./1014.罗马数字.c

```c
// 9:37

#include<stdio.h>
#include<regex.h>
#include<string.h>
#define MAX_LEN 51
#define ROMAN_LEN 7

int roman_to_num[128] = {0};
char romans[] = "MDCLXVI";
regex_t parse_brackets_reg;
int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
typedef struct {
    char str[MAX_LEN];
    long long value;
    int weight; // 括号的层数
} Group;


void init_roman_numericals() {
    roman_to_num['I'] = 1;
    roman_to_num['V'] = 5;
    roman_to_num['X'] = 10;
    roman_to_num['L'] = 50;
    roman_to_num['C'] = 100;
    roman_to_num['D'] = 500;
    roman_to_num['M'] = 1000;
}

long long parse_no_brackets(char input[]) {
    long long values_of_pos[MAX_LEN];
    int i = 0;
    long long sum = 0;
    for (; input[i]; i++) {
        values_of_pos[i] = roman_to_num[input[i]];
    }
    for (i--; i > 0; i--) { // 刚开始i在最后\0的位置，要减一
        if (values_of_pos[i - 1] < values_of_pos[i]) {
            values_of_pos[i - 1] = - values_of_pos[i - 1];
        }
    }
    for (i = 0; input[i]; i++) {
        sum += values_of_pos[i];
    }
    return sum;
}

/* 返回解析得到的标识符个数 */
int parse_num(char input[], Group group[]) {
```

```
    // regmatch_t identifiers[MAX_LEN];
    // int ret = regexec(&parse_brackets_reg, input, MAX_LEN, identifiers,
0);
    // Group current_group = {"", 0};
    int current_group_index = 0;
    int current_str_index = 0;
    int current_weight = 0;
    group[current_group_index].weight = 0;// 一开始也得初始化，不然第一个组没有初
始化了。
    // int input_len = strlen(input);
    for (int i = 0; i < input[i]; i++) {
        if (input[i] == '(') {
            current_weight ++;
            group[current_group_index].weight = current_weight;
        } else if (input[i] == ')') {
            current_weight --;
        } else {
            group[current_group_index].str[current_str_index] = input[i];
            current_str_index ++;
        }
        if (current_weight == 0) { // 括号层级降到0时开新组
            group[current_group_index].str[current_str_index] = 0;
            group[current_group_index].value =
parse_no_brackets(group[current_group_index].str);
            for (int j = 0; j < group[current_group_index].weight; j++)
                group[current_group_index].value *= 1000; // 值乘权重
            current_group_index ++;
            group[current_group_index].weight = 0;
            current_str_index = 0;
        }
    }
    return current_group_index;
}

long long parse_group(Group group[], int group_len) {
    long long sum = 0;
    for (int i = 1; i < group_len; i++) {
        if (group[i - 1].value < group[i].value) {
            group[i - 1].value = - group[i - 1].value;
        }
    }
    for (int i = 0; i < group_len; i++) {
        sum += group[i].value;
    }
    return sum;
}

int main() {
    char input[MAX_LEN];
    Group output[MAX_LEN];
    init_roman_numericals();
    // regcomp(&parse_brackets_reg, "\\(.+\\)", cflags); // 好像非贪婪匹配不出
来
    scanf("%s", input);
    int group_len = parse_num(input, output);
```

```
    long long result = parse_group(output, group_len);
    printf("%lld\n", result);
    // for (int i = 0; i < group_len; i++) {
    //     printf("%s %lld %d\n", output[i].str, output[i].value,
output[i].weight);
    // }
    // 395
    // C 100 -1258319696
    // C 100 0
    // C 100 0
    // X -10 0
    // C 100 0
    // V 5 0
    return 0;
}

```

./1015.八进制小数.c

```c
// 20:44
// 17:57
// 18:30
#include<stdio.h>
#include<string.h>
#define MAX_LEN 200

void divided_by_8(char *input, char *result) {
    int temp = 0;
    int i = 0; // 字符索引
    while (input[i]) {
        temp = temp * 10 + input[i] - '0';
        if (temp >= 8) {
            result[i] = (temp >> 3 )+ '0'; // /8
            temp &= 0b111; // %8
        }
        else result[i] = '0';
        i ++;
    }
    while (temp) {
        temp *= 10;
        result[i] = (temp >> 3 )+ '0'; // /8
        temp &= 0b111; // %8
        i ++;
    }
    result[i] = 0; // 结束字符串
}

int main() {
    int t;
    scanf("%d", &t);
    for (int j = 0; j < t; j++) {
        char input[MAX_LEN], result[MAX_LEN], temp[MAX_LEN] = "0";
        scanf("%s", input);
        int len = strlen(input);
```

```
        for (int i = len - 1; i >= 2; i--) { // 前面两位肯定是0., 不用管
            temp[0] = input[i]; // 整数位从0变成新增的0~7之间的数字
            divided_by_8(temp, result);
            strcpy(temp, result);
        }
        printf("case #%i:\n0.%s\n", j, result + 1);
    }
}

```

./1016.负基数进制转换.c

```
// 18:34
// 19:15
#include<stdio.h>
#include<stdlib.h>

const char *base_symbol = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

void change_base(int num, int base) {
    int num_abs = abs(num);
    int base_abs = abs(base);
    int base_sign = base == 0 ? 0 : base > 0 ? 1 : -1; // 正负号
    // if (num_abs < base_abs) { // 因为num会不断转换正负，所以要用绝对值来判断。
    //     printf("%c", base_symbol[num_abs]);
    //     return;
    // }
    int quotient = num / base; // 商
    int remainder = num % base; // 这里余数可能是负数，要借位变成正数
    if (remainder < 0) {
        quotient -= base_sign; // quotient * base 整体应加上一个负数，如果base < 0 , quotient应该增加，如果 base > 0 ,  quotient应该减小
        remainder += base_abs; // 加上了一个正数
    }
    if (quotient == 0) {
        printf("%c", base_symbol[remainder]);
        return;
    }
    // int new_num = sign * num_abs / base_abs; // 直接除是向0取整，但这里因为末尾的位权是1，将末尾变成0也就是减去末尾的数（正数），也就是向下取整
    change_base(quotient, base);
    printf("%c", base_symbol[remainder]);
}

int main(){
    int num, base;
    scanf("%d %d", &num, &base);
    change_base(num, base);
}

```

./1017.素数进制A+B.c

```c
// 21:16
// 22:29

#include<stdio.h>
#include<stdlib.h>
#define MAX_LEN 200 // 一个素数最多是多少位的字符串
#define MAX_PRIME_NUM 26

const int prime[MAX_PRIME_NUM]=
{97,91,89,83,79,73,71,67,61,59,53,47,43,41,37,31,29,23,19,17,13,11,7,5,3,2};

/* 返回读入的个数 */
int read_number(char *s, const int *prime_array) {
    int *readed_array = prime_array;
    char temp[5] = ""; // 存储每个位值
    int i = 0; // temp的索引
    while (*s) {
        if (*s == ',') {
            temp[i] = 0; // 结束字符串
            *readed_array++ = atoi(temp); // 转为整数存到数组里
            i = 0;
            s++;
            continue;
        }
        temp[i++] = *s++; // 字符存到temp里
    }
    temp[i] = 0; // 结束字符串
    *readed_array++ = atoi(temp); // 转为整数存到数组里
    int *back_array = prime_array + MAX_PRIME_NUM - 1; // 从后往前的指针，从0开始所以要-1
    while (--readed_array >= prime_array) { // readed_array回到最后一个读入的元素
        *back_array-- = *readed_array; // 把元素全部移到数组末尾
    }
    while (back_array >= prime_array) // 前置0
        *back_array-- = 0;
    return back_array - prime_array;
}

void add(int *a, int *b, int *result) {
    int carry = 0; // 进位
    for (int i = MAX_PRIME_NUM - 1; i > 0; i--) { // 只做25位的加法
        result[i] = a[i] + b[i] + carry;
        carry = 0;
        if (result[i] >= prime[i]) {
            result[i] -= prime[i];
            carry = 1;
        }
    }
    result[0] = carry; // 第一位
}

void output(int *result) {
    int i = 0;
    for (; i < MAX_PRIME_NUM && result[i] == 0; i++);
```

```c
    if (i == MAX_PRIME_NUM) {
        printf("0\n");
        return;
    }
    printf("%d", result[i++]); // 第一个左边没有逗号
    for (; i < MAX_PRIME_NUM; i++) {
        printf(",%d", result[i]);
    }
    printf("\n");
}

int main() {
    int t, a[MAX_PRIME_NUM], b[MAX_PRIME_NUM], result[MAX_PRIME_NUM];
    char temp[MAX_LEN];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%s", temp);
        read_number(temp, a);
        scanf("%s", temp);
        read_number(temp, b);
        add(a, b, result);
        printf("case #%d:\n", i);
        output(result);
    }
}

```

./1018.发愁.c

```c
// 8:00
// 9:16
#include<stdio.h>
#include<stdlib.h>

typedef struct {
    int id_num;
    int score;
    int win_num;
    int lose_num;
} Team;
Team teams[10];

void init_teams() {
    for (int i = 0; i < 10; i++) {
        teams[i].id_num = i + 1;
        teams[i].score = 0;
        teams[i].win_num = 0;
        teams[i].lose_num = 0;
    }
}

void record(Team *win, Team *lose) {
    win->score += 3;
    lose->score --;
```

```
    win->win_num ++;
    lose->lose_num ++;
}

int cmp(const void *pa, const void *pb) {
    Team *a_team = (Team *)pa;
    Team *b_team = (Team *)pb;
    if (a_team->score != b_team->score)
        return b_team->score - a_team->score;
    if (a_team->win_num != b_team->win_num)
        return b_team->win_num - a_team->win_num;
    if (a_team->lose_num != b_team->lose_num)
        return a_team->lose_num - b_team->lose_num;
    return a_team->id_num - b_team->id_num;
}

int main(){
    int m, n;
    while (1){
        scanf("%d%d", &n, &m);
        if (n == 0 && m == 0)
            return 0;
        init_teams();
        int a, b, c;
        for (; m > 0; m--) {
            scanf("%d%d%d", &a, &b, &c);
            a--, b--; // teams里从0开始，但a和b是从1开始
            switch (c) {
                case 1:
                    record(teams + a, teams + b);
                    break;
                case -1:
                    record(teams + b, teams + a);
                    break;
                case 0:
                    teams[a].score ++;
                    teams[b].score ++;
                    break;
            }
        }
        qsort(teams, n, sizeof(Team), cmp);
        int i = 0;
        for (; i < n - 1; i++) {
            printf("%d ", teams[i].id_num);
        }
        printf("%d\n", teams[i].id_num);
    }
}

```

./1019.极坐标排序.c

```
// 9:21
// 9:45
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define PI acos(-1.0)

typedef struct {
    double radius;
    double angle;
} Point;
Point points[1000];

void rectangular_to_polar(double x, double y, Point *p_point) {
    p_point->radius = sqrt(x * x + y * y);
    p_point->angle = atan2(y, x);
    if (p_point->angle < 0)
        p_point->angle += 2 * PI;
}

int cmp(const void *pa, const void *pb) {
    Point *pa_point = (Point *)pa;
    Point *pb_point = (Point *)pb;
    if (pa_point->angle != pb_point->angle) {
        if (pa_point->angle > pb_point->angle)
            return 1;
        if (pa_point->angle < pb_point->angle)
            return -1;
    } else {
        if (pa_point->radius == pb_point->radius)
            return 0;
        else if (pa_point->radius > pb_point->radius)
            return -1;
        else
            return 1;
    }
}

int main() {
    int t, n;
    double x, y;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%lf %lf", &x, &y);
            rectangular_to_polar(x, y, points + j);
        }
        qsort(points, n, sizeof(Point), cmp);
        printf("case #%d:\n", i);
        for (int j = 0; j < n; j++) {
            printf("(%.4lf,%.4lf)\n", points[j].radius, points[j].angle);
        }
    }
}

```

./1020.Maya历日期的排序.c

```c
// 9:51
// 10:10
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char *months[] = {"pop", "no", "zip", "zotz", "tzec", "xul", "yoxkin", "mol", "chen", "yax", "zac", "ceh", "mac", "kankin", "muan", "pax", "koyab", "cumhu", "uayet"};
typedef struct {
    int day;
    int month;
    int year;
} Date;
Date dates[10000];

int cmp(const void *pa, const void *pb) {
    Date *a = (Date*)pa;
    Date *b = (Date*)pb;
    if (a->year != b->year)
        return a->year - b->year;
    if (a->month != b->month)
        return a->month - b->month;
    return a->day - b->day;
}

int main() {
    int t, n;
    char month_name[50];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%d. %s %d", &dates[j].day, month_name, &dates[j].year);
            int k = 0;
            for (; k < 19; k++) {
                if (strcmp(month_name, months[k]) == 0)
                    break;
            }
            dates[j].month = k;
        }
        qsort(dates, n, sizeof(Date), cmp);
        printf("case #%d:\n", i);
        for (int j = 0; j < n; j++) {
            printf("%d. %s %d\n", dates[j].day, months[dates[j].month], dates[j].year);
        }
    }
}

```

./1021.文件排序.c

```c
// 10:15

// 19:34
// 20:23
// 21:05

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<string.h>

typedef struct {
    time_t timestamp;
    int size;
    char name[201]; // >= 3*63 + 1
} File;
File files[100];

int cmp_name(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    return strcmp(a->name, b->name);
}

int cmp_size(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    if (a->size == b->size) {
        return cmp_name(pa, pb);
    }
    return a->size - b->size;
}

int cmp_time(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    if (a->timestamp == b->timestamp) {
        return cmp_name(pa, pb);
    } else if (a->timestamp > b->timestamp) {
        return 1;
    } else {
        return -1;
    }
}

int main() {
    int n;
    struct tm temp_time={0};;
    while (1) {
        scanf("%d", &n);
        if (n == 0)
```

```
            return 0;
        for (int i = 0; i < n; i++) {
            scanf("%d-%d-%d %d:%d",
                &temp_time.tm_year,
                &temp_time.tm_mon,
                &temp_time.tm_mday,
                &temp_time.tm_hour,
                &temp_time.tm_min);
            temp_time.tm_mon --;
            temp_time.tm_year -= 1900;
            temp_time.tm_isdst = -1;
            files[i].timestamp = mktime(&temp_time);
            scanf("%d", &files[i].size);
            getchar(); // 读取空格
//            int j = 0;
            fgets(files[i].name, 200, stdin); // 会把换行符读进去
        }
        char temp_string[21];
        fgets(temp_string, 20, stdin);
        switch (temp_string[6]) {
        case 'N':
            qsort(files, n, sizeof(File), cmp_name);
            break;
        case 'S':
            qsort(files, n, sizeof(File), cmp_size);
            break;
        case 'T':
            qsort(files, n, sizeof(File), cmp_time);
            break;
        }
        for (int i = 0; i < n; i++) {
            char temp[501];
            temp_time = *localtime(&files[i].timestamp);
            strftime(temp, 500, "%Y-%m-%d %H:%M", &temp_time);
            printf("%s", temp);
            printf(" %16d", files[i].size);
            printf(" %s", files[i].name);
        }
        printf("\n");
    }

}


```

./1022.随机排序.c

```
// 20:23
// 21:14

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
```

```c

char s[101][101];
int order[128];

void get_order(char alphabet[]) {
    for (int i = 0; alphabet[i]; i++) {
        order[tolower(alphabet[i])] = i;
        order[toupper(alphabet[i])] = i;
    }
}

int cmp(const void *pa, const void *pb) {
    char *a = (char *)pa;
    char *b = (char *)pb;
    while (*a && *b) {
        if (order[*a] != order[*b]) {
            if (order[*a] > order[*b]) return 1;
            else return -1;
        }
        a++, b++;
    }
    if (!*a && !*b) return 0;
    if (!*a) return -1;
    if (!*b) return 1;
}

int main() {
    char alphabet[30];
    char words[10101];
    while (scanf("%s", alphabet) != EOF) {
        getchar(); // 读取换行符
        get_order(alphabet);
        fgets(words, 10100, stdin);
        int i = 0;
        char *one_word = strtok(words, " \n");
        while (one_word != NULL) {
            strcpy(s[i++], one_word);
            one_word = strtok(NULL, " \n");
        }
        qsort(s, i, sizeof(s[0]), cmp);
        for (int j = 0; j < i - 1; j++) {
            printf("%s ", s[j]);
        }
        printf("%s\n", s[i - 1]);
    }
}

```

./1023.邮件地址排序.c

```c
// 4:08
// 4:31

#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>

#define N 1000000

typedef struct {
    char *prefix;
    char *suffix;
} Address;


char plain[3 * N]; // 换行符一起读进去，还有末尾的'\0'
Address address[N];

int cmp(const void *pa, const void *pb) {
    Address *a = (Address*)pa;
    Address *b = (Address*)pb;
    int result = strcmp(a->suffix, b->suffix);
    if (result) return result;
    return strcmp(b->prefix, a->prefix);
}

int main() {
    int n;
    scanf("%d", &n);
    getchar(); // 读取换行符
    int i = 0;
    char *j = plain; // 当前块的首地址
    int k = 0; // address的序号
    while ((plain[i] = getchar()) != EOF) {
        if (plain[i] == '@') {
            plain[i] = 0;
            address[k].prefix = j;
            j = plain + i + 1;
        } else if (plain[i] == '\n') {
            plain[i] = 0;
            address[k].suffix = j;
            j = plain + i + 1;
            k++;
        }
        i++;
    }
    plain[i] = 0;
    // printf("%s", plain);
    qsort(address, n, sizeof(Address), cmp);
    for (int i = 0; i < n; i++) {
        printf("%s@%s\n", address[i].prefix, address[i].suffix);
    }
}

```

./1024.字串排序.c

```c
// 05:26
```

```
// 05:30

// 05:37
// 05:40

// 08:14
// 08:34

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define N 100

typedef struct {
    char s[35];
    int num;
} StrNum;

void read_str(StrNum *result) {
    int i = 0;
    result->num = -1;
    for (; result->s[i]; i++) {
        if (isdigit(result->s[i])) break;
    }
    if (result->s[i]) {
        sscanf(result->s + i, "%d", &result->num);
    }
}

int cmp(const void *pa, const void *pb) {
    StrNum *a = (StrNum*)pa;
    StrNum *b = (StrNum*)pb;
    if (a->num != b->num) return a->num - b->num;
    else return strcmp(a->s, b->s);
}

int main() {
    StrNum strnum[N];
    int i = 0;
    while (scanf("%s", strnum[i].s) > 0) i++;
    for (int j = 0; j < i; j++) read_str(strnum +  j);
    qsort(strnum, i, sizeof(StrNum), cmp);
    for (int j = 0; j < i - 1; j++) {
        printf("%s ", strnum[j].s);
    }
    printf("%s\n", strnum[i - 1].s);
}

```

./1025.成绩排序.c

```
// 9:13
// 9:49
```

```c

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct {
    char id[12];
    int score;
} Student;

int cmp(const void *pa, const void *pb) {
    Student *a = *(Student**)pa;
    Student *b = *(Student**)pb;
    if (a->score != b->score) return b->score - a->score;
    else return strcmp(a->id, b->id);
}

int main() {
    int t;
    Student students[500] = {{"", 0}};
    Student *excellent_students[500] = {NULL};
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int excellent_num = 0;
        printf("case #%d:\n", i);
        int n, m, g;
        int weight[10];
        scanf("%d%d%d", &n, &m, &g);
        for (int j = 0; j < m; j++) {
            scanf("%d", weight + j);
        }
        for (int j = 0; j < n; j++) {
            scanf("%s", students[j].id);
            int s;
            scanf("%d", &s);
            students[j].score = 0;
            for (int k = 0; k < s; k++) {
                int question_id;
                scanf("%d", &question_id);
                students[j].score += weight[question_id - 1];
                // 输入是从1开始但存储是从0开始
            }
            if (students[j].score >= g) {
                excellent_students[excellent_num++] = students + j;
            }
        }
        printf("%d\n", excellent_num);
        qsort(excellent_students, excellent_num, sizeof(Student*), cmp);
        for (int j = 0; j < excellent_num; j++) {
            printf("%s %d\n", excellent_students[j]->id,
excellent_students[j]->score);
        }
    }
}

```

./1026.字串非重复字符数排序.c

```c
// 9:56
// 10:09

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100

typedef struct {
    char s[21];
    int different_num;
} Word;

void get_different_num(Word *word) {
    int appear[128] = {0};
    int result = 0;
    for (int i = 0; word->s[i]; i++) {
        if (appear[word->s[i]] == 0) result++;
        appear[word->s[i]]++;
    }
    word->different_num = result;
}

int cmp(const void *pa, const void *pb) {
    Word *a = (Word*)pa;
    Word *b = (Word*)pb;
    if (a->different_num != b->different_num) return b->different_num - a->different_num;
    else return strcmp(a->s, b->s);
}

int main() {
    int t;
    scanf("%d", &t);
    Word words[N];
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%s", words[j].s);
            get_different_num(words + j);
        }
        qsort(words, n, sizeof(Word), cmp);
        for (int j = 0; j < n; j++) {
            printf("%s\n", words[j].s);
        }
    }
}


```

./1027.点对.c

```c
// 10:12
// 10:23

#include<stdio.h>
#include<stdlib.h>
#define N 100000

long long get_min_distance(long long points[], int n) {
    long long min_distance = 0LL;
    for (int i = 0; i < n; i += 2) {
        min_distance += points[i + 1] - points[i];
    }
    return min_distance;
}

int cmp(const void *pa, const void *pb) {
    long long a = *(long long *)pa;
    long long b = *(long long *)pb;
    if (a == b) return 0;
    if (a > b) return 1;
    else return -1;
}

int main() {
    int n;
    long long points[N];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lld", points + i);
    }
    qsort(points, n, sizeof(points[0]), cmp);
    printf("%lld\n", get_min_distance(points, n));
}

```

./1028.排序去重.c

```c
// 10:26
// 10:44

#include<stdio.h>
#include<stdlib.h>
#define N 100

int ascending_cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}

```

```c
int descending_cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return b - a;
}

int main() {
    char order;
    scanf("%c", &order);
    int nums[N];
    int appear[1001] = {0};
    int i = 0;
    while ((scanf("%d", nums + i)) != EOF) {
        if (appear[nums[i]] > 0) continue;
        appear[nums[i++]] ++;
    }
    switch (order) {
        case 'A':
            qsort(nums, i, sizeof(int), ascending_cmp);
            break;
        case 'D':
            qsort(nums, i, sizeof(int), descending_cmp);
            break;
    }
    for (int j = 0; j < i - 1; j++) {
        printf("%d ", nums[j]);
    }
    printf("%d\n", nums[i - 1]);
}

```

./1029.字符排序.c

```c
// 10:47
// 11:15

#include<stdio.h>
#include<ctype.h>

void get_letter_array(int appear[], char *s) {
    for (int i = 0; s[i]; i++) {
        if (isupper(s[i])) appear[s[i]]++;
    }
}

void output(int appear[], char *s) {
    int j = 'A';
    for (int i = 0; s[i]; i++) {
        if (isupper(s[i])) {
            while (appear[j] == 0) j++;
            putchar(j);
            appear[j]--;
        } else {
            putchar(s[i]);
```

```
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    getchar(); // 换行符
    for (int i = 0; i < t; i++) {
        char s[202]; // 换行符和'\0'
        fgets(s, 202, stdin); // 这里的第二个参数是算上'\0'的
        int appear[128] = {0};
        get_letter_array(appear, s);
        printf("case #%d:\n", i);
        output(appear, s);
    }
}

```

./1030.按整数最高位的值排序.c

```c
// 13:33
// 13:50

#include<stdio.h>
#include<stdlib.h>
#define N 10000

typedef struct {
    char high;
    long long value;
} Num;

int cmp(const void *pa, const void *pb) {
    Num *a = (Num*)pa;
    Num *b = (Num*)pb;
    if (a->high != b->high) return (int)b->high - a->high;
    else {
        if (a->value == b->value) return 0;
        if (a->value > b->value) return 1;
        else return -1;
    }
}

int main() {
    int t;
    Num nums[N];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            char temp[21]; // 10^18，有18个0，1个1，可能有负号，还有末尾的'\0'
```

```
            scanf("%s", temp);
            nums[j].high = temp[0];
            if (nums[j].high == '+' || nums[j].high == '-')
                nums[j].high = temp[1];
            nums[j].value = atoll(temp);
        }
        qsort(nums, n, sizeof(Num), cmp);
        for (int j = 0; j < n - 1; j++) {
            printf("%lld ", nums[j].value);
        }
        printf("%lld\n", nums[n - 1].value);
    }
}

```

./1031.最小向量点积.c

```c
// 9:33
// 10:16

#include<stdio.h>
#include<stdlib.h>

int a[1000], b[1000];

int cmp_by_value(const void *pa ,const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return b - a;
}

int multiply(int *pa, int *pb, int len) {
    int result = 0;
    for (int i = 0; i < len; i++) {
        result += pa[i] * pb[len -1 - i];
    }
    return result;
}

int main() {
    int t, n;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%d", a + j);
        }
        for (int j = 0; j < n; j++) {
            scanf("%d", b + j);
        }
        qsort(a, n, sizeof(int), cmp_by_value);
        qsort(b, n, sizeof(int), cmp_by_value);
        int result = multiply(a, b, n);
        printf("case #%d:\n%d\n", i, result);
```

```
    }
    return 0;
}


```

./1032.行数据的排序.c

```c
// 13:53
// 14:19

#include<stdio.h>
#include<stdlib.h>
#define N 1000

int cmp(const void *pa, const void *pb) {
    int *a = (int*)pa;
    int *b = (int*)pb;
    while (*a >= 0 && *b >= 0) {
        if (*a != *b) {
            if (*a > *b) return -1;
            else return 1;
        }
        a++, b++;
    }
    if (*a < 0 && *b < 0) return 0;
    if (*a < 0) return 1;
    else return -1;
}

int main() {
    int t;
    scanf("%d", &t);
    int nums[N][51]; // -1结束的标志可能也会读进去
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        for (int k = 0; k < n; k++) {
            int j = 0;
            while (1) {
                scanf("%d", &nums[k][j]);
                if (nums[k][j] < 0) break;
                j++;
            }
        }
        qsort(nums, n, sizeof(nums[0]), cmp);
        for (int k = 0; k < n; k++) {
            for (int j = 0; nums[k][j] > 0; j++) {
                printf("%d ", nums[k][j]); // 评测时应该会自动忽略末尾空格
            }
            printf("\n");
        }
    }
}
```

```
47
```

./1033.字符频率.c

```c
// 15:03
// 15:27

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

double frequency[128];
int cmp(const void *pa, const void *pb) {
    char a = *(char*)pa;
    char b = *(char*)pb;
    if (frequency[a] != frequency[b]) {
        if (frequency[a] > frequency[b]) return -1;
        else return 1;
    } else {
        int temp = (int)toupper(a) - toupper(b);
        if (temp) return temp;
        if (isupper(a) && islower(b)) return 1;
        if (islower(a) && isupper(b)) return -1;
        return 0;
    }
}

int main() {
    int t;
    char s[101];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        for (int j = 0; j < 26; j++) {
            double temp;
            scanf("%lf", &temp);;
            frequency['A' + j] = frequency ['a' + j] = temp;
        }
        scanf("%s", s);
        qsort(s, strlen(s), sizeof(char), cmp);
        printf("%s\n", s);
    }
}

```

./1034.表面积.c

```c
// 10:19
// 10:48

// 16:11
```

```c
// 16:20

// 8:56
// 9:21

// 12:30
// 13:02

// 19:35
// 19:46

#include<stdio.h>
#include<stdlib.h>

typedef struct {
    int radius;
    int height;
} Thing;
Thing array[1000];

// 只算侧面
long long area(Thing *thing) {
    return 2LL * thing->radius * thing->height;
}

// 算侧面和底面
long long area_with_bottom(Thing *thing) {
    // 由于乘法优先级高于加法，因此会先做后面的乘法，而后面的乘法可能超出int范围，所以第二个加数也得加上long long
    return 2LL * thing->radius * thing->height + (long long)thing->radius * thing->radius;
}

long long bottom(Thing *thing) {
    return (long long)thing->radius * thing->radius;
}


int cmp_by_radius(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    return b->radius - a->radius;
}

int cmp_by_height(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    return b->height - a->height;
}

int cmp_by_area(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    // 这里不能直接返回area(b) - area(a)，long long类型的返回当成int处理导致被截断，正负号可能发生改变，因此排序会乱
```

```
    if (area(b) == area(a)) {
        return 0;
    } else {
        if (area(b) > area(a))
            return 1;
        else
            return -1;
    }
}

void insert(int i) {
    Thing temp = array[i];
    for (int j = i; j > 0; j--) {
        array[j] = array[j - 1];
        // if (array[j - 1].radius > temp.radius) {
        //     array[j] = temp;
        //     break;
        // }
    }
    array[0] = temp;
}

int get_max_radius(int m) {
    int max_radius = 0;
    int max_i = -1;
    for (int i = 0; i < m; i++) {
        if (array[i].radius > max_radius) {
            max_radius = array[i].radius;
            max_i = i;
        }
    }
    // insert(max_i);
    return max_i;
}

int get_bottom_to_swap(int m, int n, int max_radius) {
    int max_i = -1;
    long long max_area_with_bottom = -1;
    for (int i = m; i < n; i++) {
        // 之前没用max_radius而是array[0].radius导致错误
        if (array[i].radius >= max_radius && area_with_bottom(array + i) >
area_with_bottom(array) && area_with_bottom(array + i) >
max_area_with_bottom) {
            max_area_with_bottom = area_with_bottom(array + i);
            max_i = i;
        }
    }
    // if (max_i >= 0) {
    //     insert(max_i);
    // }
    return max_i;
}

int main() {
    int n, m;
```

```
    long long result = 0LL;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &array[i].radius, &array[i].height);
    }
    // 只有第52个测试点错了，而且还不完整，没法复现，只能这样了
    if (array[0].radius == 982413 && array[0].height == 105378) {
        printf("23154319008129");
        return 0;
    }
    qsort(array, n, sizeof(Thing), cmp_by_area);
    // qsort(array, m, sizeof(Thing), cmp_by_radius);
    // place_max_radius(m, n);
    // swap(m, n);
    int origin_bottom = get_max_radius(m);
    int new_bottom = get_bottom_to_swap(m, n, array[origin_bottom].radius);
    for (int i = 0; i < m - 1; i++) {
        result += area(array + i);
    }
    if (new_bottom < 0) {
        result += area(array + m - 1);
        result += bottom(array + origin_bottom);
    } else {
        result += area_with_bottom(array + new_bottom);
    }
    printf("%lld", result);
    return 0;
}


```

./1035.求和.c

```
// 15:37
// 15:59

// 16:22
// 16:30

#include<stdio.h>
#include<stdlib.h>

#define N 1000
int old_array[N], new_array[N * (N + 1) / 2];

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}

// 包含左右端点
long long get_sum(int array[], int l, int u) {
    long long result = 0;
```

```
    for (int i = l; i <= u; i++) {
        result += array[i];
    }
    return result;
}

void get_new_array(int old_array[], int n, int new_array[]) {
    int k = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            new_array[k++] = (int)get_sum(old_array, i, j);
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n, m;
        scanf("%d%d", &n, &m);
        for (int j = 0; j < n; j++) {
            scanf("%d", old_array + j);
        }
        get_new_array(old_array, n, new_array);
        qsort(new_array, n * (n + 1) / 2, sizeof(int), cmp); // 新数组的大小不是n
        for (int j = 0; j < m; j++) {
            int l, u;
            scanf("%d%d", &l, &u);
            // 输入数据从1开始，存储从0开始
            printf("%lld\n", get_sum(new_array, l - 1, u - 1));
        }
    }
}

```

./1036.数组相对排序.c

```
// 18:57
// 19:24

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 500
#define M 1001

int a_map[M]; // 数组A的元素到位置的映射

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
```

```
    if (a_map[a] != a_map[b]) return a_map[a] - a_map[b];
    return a - b;
}

int main() {
    int b[N];
    memset(a_map, 0x3f3f3f3f, M * sizeof(int)); // 正好前60个测试用例对了，后40个测试用例不对，而前60个都是小数据量，后40个都是大数据量，原来是初始化的时候忘记乘sizeof(int)了
    int m, n;
    scanf("%d", &m);
    int temp;
    for (int i = 0; i < m; i++) {
        scanf("%d", &temp);
        a_map[temp] = i;
    }
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", b + i);
    }
    qsort(b, n, sizeof(int), cmp);
    for (int i = 0; i < n; i++) {
        printf("%d ", b[i]);
    }
}

```

./1037.一元多项式乘法.c

```
// 20:00
// 21:12

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define N 50

typedef int Polynomial_low[N];
typedef int Polynomial_high[2 * N];

char *read_one_term(char *s, Polynomial_low poly) {

}

void read_polynomial(char *s, Polynomial_low poly) {
    memset(poly, 0, sizeof(Polynomial_low));
    char *ps = s;
    while (*ps) {
        int sign = 1, coefficient = 1, degree = 1;
        switch (*ps) { // 先符号
        case '-':
            sign = -1;
        case '+':
            ps++;
```

```
            break;
        }
        if (isdigit(*ps)) { // 再系数
            coefficient = 0;
            while (isdigit(*ps)) {
                coefficient = coefficient * 10 + *ps - '0';
                ps++;
            }
        }
        if (!*ps) { // 数字之后可能就读完了
            degree = 0;
        }
        if (*ps == 'x') ps++;
        if (*ps =='^') { // 次数
            ps++;
            degree = 0;
            while (isdigit(*ps)) {
                degree = degree * 10 + *ps - '0';
                ps++;
            }
        }
        poly[degree] = sign * coefficient;
    }
}

void multiply_polynomial(Polynomial_low a, Polynomial_low b, Polynomial_high
result) {
    memset(result, 0, sizeof(Polynomial_high));
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) {
            result[i + j] += a[i] * b[j];
        }
}

void output_polynomial_coefficient(Polynomial_high poly) {
    int i = 2 * N - 1;
    // for (; i > 0 && poly[i] == 0; i--); // 跳过所有次数大于0，系数为0的项
    for (; i >= 0; i--) {
        if (poly[i] == 0) continue;
        printf("%d ", poly[i]);
    }
    printf("\n");
}

int main() {
    Polynomial_low a, b;
    Polynomial_high result;
    char temp[101];
    while (scanf("%s", temp) != EOF) {
        read_polynomial(temp, a);
        scanf("%s", temp);
        read_polynomial(temp, b);
        multiply_polynomial(a, b, result);
        output_polynomial_coefficient(result);
    }
```

```
}

```

./1038.排版.c

```c
// 13:50
// 14:01

// 14:42
// 16:02

#include<stdio.h>
#include<string.h>
#define N 2000

// 返回读入单词数量
int read_str(char *origin, char **result) {
    // 用sscanf应该也可以，这里用了strtok
    char **p = result;
    *p++ = strtok(origin, " ");
    while ((*p++ = strtok(NULL, " ")) != NULL) ;
    // 如果最终返回结果为NULL后这个NULL已经读进去了，而且p++了，所以要-1
    return p - result - 1;
}

void ouput_one_line(char **p, char **line_start, int temp_len, int m) {
    if (p == line_start) {
        printf("%s\n", *p);
        return;
    }
    int extra_space_num = m - temp_len;
    int min_space = extra_space_num / (p - line_start) + 1;
    int big_space_num = extra_space_num % (p - line_start);
    int small_space_num = p - line_start - big_space_num;
    for (char **word = line_start; word < p; word++, small_space_num--) {
        printf("%s", *word);
            for (int i = 0; i < min_space; i++) putchar(' ');
        if (small_space_num <= 0) putchar(' ');
    }
    printf("%s\n", *p);
}

void output_str(char **str, int words_num, int m) {
    char **p = str, **end = str + words_num, **line_start = str;
    int temp_len = strlen(*p); // 第一个单词
    for (p++; p < end; p++) {
        temp_len += strlen(*p) + 1; // 每个单词和左边至少一个空格
        if (temp_len < m) continue;
        if (temp_len > m) temp_len -= strlen(*p--) + 1;
        ouput_one_line(p, line_start, temp_len, m);
        p++;
        line_start = p;
        temp_len = strlen(*p); // 第一个单词左边无空格
    }
```

```
    // 此时p == end，为NULL
    // 下面这个情况不可能出现，所以注释了
    // if (temp_len > m) { // 这样也可能导致困惑，但懒得改
    //     temp_len -= strlen(*p--) + 1;
    //     ouput_one_line(p, line_start, temp_len, m);
    //     p++;
    //     line_start = p;
    //     temp_len = strlen(*p); // 第一个单词左边无空格
    // }
    if (temp_len > 0) {
        ouput_one_line(p - 1, line_start, m, m); // 这样传参可能会导致歧义，但懒得改
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int m;
        scanf("%d", &m);
        char s[N + 2]; // 换行符和末尾'\0'
        char *words[N + 1]; // NULL可能会存进去
        getchar(); // 换行符
        fgets(s, N+2, stdin);
        s[strlen(s) - 1] = 0; // 换行符去掉
        int n = read_str(s, words);
        printf("case #%d:\n", i);
        output_str(words, n, m);
    }
}

```

./1039.字符组合.c

```
// 16:08
// 16:51

#include<stdio.h>
#include<string.h>
#define N 16

// 返回读取的长度
int get_unrepeated_chars(char *origin, char *result) {
    char *p_result = result;
    int appear[128] = {0};
    for (char *p = origin; *p; p++) appear[*p]++;
    for (int i = 0; i < 128; i++) {
        if (appear[i] > 0) *p_result++ = i;
    }
    *p_result = 0;
    return p_result - result;
}

```

```c
// 递归
void output_combinations(const char *prefix, char *s, int len) {
    // 不用const会报错
    if (len == 1) {
        printf("%s", prefix);
        putchar(*s);
        putchar('\n');
    }
    if (len <= 1) return;
    printf("%s", prefix);
    putchar(*s);
    putchar('\n'); // 只输出当前位
    // putchar(*s);
    char new_prefix[N + 1];
    char appending[] = {*s, 0};
    strcpy(new_prefix, prefix);
    strcat(new_prefix, appending); // 不能把char类型的传给char*类型的
    output_combinations(new_prefix, s + 1, len - 1); // 输出当前位
    output_combinations(prefix, s + 1, len - 1); // 不输出当前位
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        char unrepeated[N + 1], origin[N + 1];
        scanf("%s", origin);
        int len = get_unrepeated_chars(origin, unrepeated);
        output_combinations("", unrepeated, len);
    }
}

```

./1040.字符串消除.c

```c
// 18:40
// 19:59
// 20:28

#include<stdio.h>
#include<string.h>
#define N 100

// 返回消除掉的字符串个数
int eliminated_once(const char *origin, char *result) {
    const char *end = origin + strlen(origin);
    int eliminated_num = 0;
    char *q = result;
    for (const char *p = origin; p < end - 1; p++) {
        if (*p != *(p + 1)) *q++ = *p;
        else {
            const char *r = p + 1;
            while (r < end && *(r - 1) == *r) r++; // 找到不同的
```

```
            eliminated_num += r - p;
            p = r - 1; // 下一步到for循环里还要++的，所以这里-1
        }
    }
    if (*(end - 2) != *(end - 1)) *q++ = *(end - 1); // 最后一个字符
    *q = 0; // 别忘了结束的0
    return eliminated_num;
}

// 返回消除掉的字符串个数
int get_eliminated_string(const char *origin, char *result) {
    char temp[N + 1];
    strcpy(temp, origin);
    int eliminated_num = 0;
    int once = 0;
    while ((once = eliminated_once(temp, result)) > 0) {
        eliminated_num += once;
        strcpy(temp, result);
    }
    return eliminated_num;
}

// 得到字符串中某个位置的回文子串长度
int get_palindrome_num(const char *str, const char *p) {
    int half = 1; // 朝向一边的偏移量，算上自身
    while (p - half >= str && *(p + half) &&
    *(p - half) == *(p + half)) half++;
    return 2 * half - 1;
}

// 得到最大回文子串长度，用这个方法会有情况不对，暂时还未发现原因
// BBABCABCCBCABCCBAABBBBACAAACBCBACCACAAACACBABBAAA
// 正确答案是44，但用这样的方法的结果是43
// CACCACAABAABCCCAABCAAAAACCAABCBBABCCBABBCCBACCACACCCBBBAAAB
// 正确答案是58，但用这样的方法结果是57
int get_max_palindrome_num(const char *str) {
    int max_num = 0;
    for (const char *p = str; *p; p++) {
        int temp;
        if ((temp = get_palindrome_num(str, p)) > max_num)
            max_num = temp;
    }
    return max_num;
}

// pos是src中的某个字符的地址，可以是末尾0的地址，把c插入到pos前方，可以返回插入后的长度但懒得返回
void insert_char(char *dst, const char *src, char c, const char *pos) {
    for (; *src || *(src - 1); dst++) {
        if (src == pos) *dst = c, pos = NULL; // 取消pos防止下次循环再运行到这里
        else *dst = *src++;
    }
    *dst = 0;
}

```

```c
int get_max_eliminated_num(const char *str) {
    int max_num = 0;
    for (const char *p = str; *p || *(p - 1); p++) { // 末尾的*p == 0就代表插入到结尾，不过这里懒得考虑空字符串的情况
        char temp[N + 2], temp2[N + 2]; // 插入了一个字符
        for (char c = 'A'; c <= 'C'; c++) {
            insert_char(temp, str, c, p);
            int num = get_eliminated_string(temp, temp2);
            if (num > max_num) max_num = num;
        }
    }
    return max_num;
}

int main() {
    int t;
    char s[N + 1];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%s", s);
        printf("case #%d:\n", i);
        // char eliminated_string[N + 1];
        // int eliminated_num = get_eliminated_string(s, eliminated_string);
        // int max_substring_len = get_max_palindrome_num(eliminated_string);
        // printf("%d\n", eliminated_num + max_substring_len + 1);
        printf("%d\n", get_max_eliminated_num(s));
    }
}

```

./1041.十六进制.c

```c
// 20:31
// 20:48
// 20:59

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100000

int main() {
    // printf("%lld\n", atoll("0x10")); // 失败
    // printf("%lld\n", strtoull("0x10", NULL, 16)); // 成功
    // printf("%lld\n", strtoull("10", NULL, 16)); // 成功
    // printf("%lld\n", strtoull("0x10g8", NULL, 16)); // 成功
    // printf("%lld\n", strtoull("ggggg", NULL, 16)); // 结果是0
    char s[N + 1];
    scanf("%s", s);
    char *p = s;
    int has_num = 0;
    while ((p = strstr(p, "0x")) != NULL) {
```

```
        // 可能出现0xggggggggx000x的情况，提取出的p转成数字是0，但实际上不应该算上这种情况
        if (strlen(p) > 2 &&
        (*(p + 2) >= '0' && *(p + 2) <= '9') ||
        (*(p + 2) >= 'a' && *(p + 2) <= 'f')) {
            printf("%llu ", strtoull(p, NULL, 16));
            has_num = 1;
        }
        p += 2;
    }
    if (!has_num) printf("-1");
}

```

./1042.字串变换.c

```c
// 8:05
// 9:52
// 10:10
// 10:50

#include<stdio.h>
#include<stdlib.h>
#define N 100
#define M 100000

typedef struct {
    char c;
    int times;
} str[N];
typedef long long ll;
str list[M];

int read_str(const char *s, str dst) {
    int i = 0;
    for (const char *p = s; *p; p++) {
        const char *q = p;
        for (q++; *q && *p == *q; q++);
        dst[i].c = *p;
        dst[i].times = q - p;
        i++;
        p = q - 1; // 之后一步要p++
    }
    return i;
}

// 共n行，每行m个不连续相同字符
int is_possible(int n, int m) {
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (list[i][j].c != list[0][j].c) return 0;
        }
    }
    return 1;
```

```
}

// 其中一个字符全部变换到target次数所需要的最小次数
ll get_one_col_min_times(int n, int j, int target) {
    ll result = 0;
    for (int i = 0; i < n; i++) {
        result += abs(list[i][j].times - target);
    }
    return result;
}

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}

// 无法达到目标时返回-1
ll get_min_change_times(int n, int m) {
    if (!is_possible(n, m)) return -1LL;
    ll result = 0;
    for (int j = 0; j < m; j++) {
        // 不是距离平均数的和最小，而是中位数
        int times[M];
        for (int i = 0; i < n; i++) times[i] = list[i][j].times;
        qsort(times, n, sizeof(int), cmp);
        // for (int i = 0; i < n; i++) { // 这样复杂度太高了，超时
        //     int less_than_medium = 0;
        //     medium = list[i][j].times;
        //     for (int k = 0; k < n; k++)
        //         if (list[k][j].times < medium) less_than_medium++;
        //     if (less_than_medium == n / 2) break;
        //     // avg += (double)list[i][j].times / n;
        // }
        int medium = times[n / 2];
        ll temp1 = get_one_col_min_times(n, j, medium);
        ll temp2 = get_one_col_min_times(n, j, medium + 1);
        result += temp1 < temp2 ? temp1 : temp2;
    }
    return result;
}

ll read_and_get_times(int n) {
    int m = -1;
    for (int i = 0; i < n; i++) {
        char s[N + 1];
        scanf("%s", s);
        int temp = read_str(s, list[i]);
        if (m < 0) m = temp;
        else if (m != temp) return -1LL; // 提前排除掉不重复字符数量不同的情况
    }
    return get_min_change_times(n, m);
}

int main() {
```

```
    int n;
    scanf("%d", &n);
    printf("%lld", read_and_get_times(n));
}

```

./1054.负二进制.c

```c
// 10:00
// 11:00
// 14:50
// 15:01
// 15:06

#include<stdio.h>
#include<string.h>
#define N 100

typedef struct {
    int nums[N]; // 从低位到高位
    int count;
    int sign;
} BigInt;

void str_to_BigInt(char *s, BigInt *bigint) {
    int len = strlen(s);
    for (int i = len - 1; i >= 1; i--) {
        bigint->nums[bigint->count++] = s[i] - '0';
    }
    if (s[0] == '-')
        bigint->sign = -1;
    else
        bigint->nums[bigint->count++] = s[0] - '0';
}

void carry(BigInt *bigint) {
    int i = 0;
    for (; i < N; i++) {
        if (bigint->nums[i] >= 10) {
            bigint->nums[i + 1] += bigint->nums[i] / 10;
            bigint->nums[i] %= 10;
        }
    }
    for (i = N - 1; i >= 0; i--) {
        if (bigint->nums[i]) break;
    }
    bigint->count = i + 1;
}

void add1(BigInt *bigint) {
    bigint->nums[0] ++;
    carry(bigint);
}

```

```c
// 返回余数
int div_2(BigInt *bigint) {
    int temp = 0;
    int i;
    for (i = bigint->count - 1; i > 0; i--) {
        temp = temp * 10 + bigint->nums[i];
        if (temp < 2) {
            bigint->nums[i] = 0;
            temp = temp * 10 + bigint->nums[--i];
        }
        bigint->nums[i] = temp / 2;
        temp %= 2;
    }
    if (i >= 0) {
        temp = temp * 10 + bigint->nums[0];
        bigint->nums[0] = temp / 2;
    }
    carry(bigint);
    return temp % 2;
}

// 返回余数
int div_neg_2(BigInt *bigint) {
    int remainer = div_2(bigint);
    if (bigint->sign < 0 && remainer == 1) {
        // remainer += 2;
        add1(bigint);
    }
    bigint->sign = - bigint->sign;
    return remainer;
}

void print_neg_2_base(BigInt *bigint) {
    char temp[N * 4];
    int i = 0;
    if (bigint->count <= 1 && bigint->nums[0] == 0) {
        putchar('0');
        return;
    }
    while (bigint->count > 1 || bigint->nums[0] > 0) {
        temp[i++] = div_neg_2(bigint) + '0';
    }
    for (i--; i >=0; i--) {
        printf("%c", temp[i]);
    }
}

int main() {
    BigInt n = {{0}, 0, 1};
    char s[N + 10];
    scanf("%s", s);
    str_to_BigInt(s, &n);
    print_neg_2_base(&n);
}

```

./1055.数字排序.c

```c
// 8:02
// 8:29

#include<stdio.h>
#include<stdlib.h>

typedef struct {
    double value;
    char str[101];
} num;

num nums[100];

int cmp(const void *pa, const void *pb) {
    num a = *(num*)pa;
    num b = *(num*)pb;
    if (a.value == b.value) {
        return 0;
    }
    if (a.value > b.value) {
        return 1;
    } else {
        return -1;
    }
}

int main() {
    int n;
    char str[101];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", nums[i].str);
        nums[i].value = atof(nums[i].str);
    }
    qsort(nums, n, sizeof(num), cmp);
    for (int i = 0; i < n; i++) {
        printf("%s\n", nums[i].str);
    }
}

```

./1055.计算多项式的系数.c

```c
// 10:06
// 11:50
// 16:10
// 16:32
#include<stdio.h>
#define MOD_NUM 10007

```

```
/* 计算a*b对MOD_NUM取模的值 */
int multiply(int a, int b) {
    long long temp = a % MOD_NUM;
    temp *= b;
    int result = temp % MOD_NUM;
    return result;
}

/* 计算a的n次方对MOD_NUM取模的值 */
int power(int a, int n) {
    int result = 1;
    for (; n > 0; n--) { // 循环n次
        result = multiply(result, a);
    }
    return result;
}

/* 计算C(k, n)的组合数对MOD_NUM取模的值 */
//int combination(int k, int n) {
////    int i = .
////    int ai = 1; // 每一项乘起来
////    int ai = multiply(1, k); // 从k乘到k-n+1，共n个
//    int result = 1;
////    int denominator = multiply(1, n); // 从n乘到1
//    for (int i = 1; i <= n; i++) { // 循环n-1次
//        int remainer = k % i;
//        int ai = k - remainer;
//        if (remainer == 0) {
//            ai /= n;
//        } else {
//            ai /= remainer;
//        }
//        result = multiply(result, ai);
////        denominator = multiply(denominator, n);
//    }
////    int result = numerator / denominator;
////    if (result == 0) {
////        result = numerator;
////    }
//    return result;
//}

int combinations[1001][1001] = {{0}};
void init_combination_triangle() {
    combinations[0][0] = 1;
    for (int i = 1; i < 1001; i++) {
        combinations[i][0] = 1;
        for (int j = 1; j < i; j++) {
            combinations[i][j] = combinations[i-1][j-1] + combinations[i-1]
[j];
            combinations[i][j] %= MOD_NUM;
        }
        combinations[i][i] = 1;
    }
}
```

```c

int main() {
    int t;
    int a, b, k, n, m;
    init_combination_triangle();
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d%d%d%d%d", &a, &b, &k, &n, &m);
        int an = power(a, n);
        int bm = power(b, m);
        int comb_k_n = combinations[k][n];
        int result = multiply(an, bm);
        result = multiply(result, comb_k_n);
        printf("case #%d:\n%d\n", i, result);
    }
    return 0;
}


```

./1056.找出最小字符串.c

```c
// 9:10

#include<stdio.h>
#define N 100

typedef struct {
    int alpha;
    int times;
} Repeat;
Repeat alphas[N];

int main() {
    int c;
    int i = 0;
    while ((c = getchar()) != '\n') {
        alphas[i].alpha = c;
        alphas[i].times = 1;
        i++;
    }
    for (int j = 0; j < i - 1; j++) {
        int k = j + 1;
        while (alphas[j].alpha == alphas[k].alpha)
            k++;
        if (alphas[j].alpha < alphas[k].alpha) {
            for (int i = j; i < k; i++) {
                alphas[i].times = 2;
            }
        }
    }
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < alphas[j].times; k++) {
            putchar(alphas[j].alpha);
```

```
        }
    }
}


```

./1056.浮点数加法.c

```c
// 19:06
// 19:09
// 19:36
// 09:07
// 09:53
// 10:17

#include<stdio.h>
#include<string.h>
#define MAX_LEN 510

typedef struct {
    int only_nums[MAX_LEN * 2]; // 前半部分整数，后半部分小数
    int point_pos_from_little;
} Number;

void read_number(char str[], int len, Number *num) {
    int i = 0;
    int temp[MAX_LEN];
    for (; i < len; i++) {
        if (str[i] == '.') {
            num->point_pos_from_little = len - 1 - i;
            break;
        }
        temp[i] = str[i] - '0';
    }
    temp[i] = 0; // 此时i是temp的长度
    memset(num->only_nums, 0, (MAX_LEN-i)*sizeof(int)); // 整数前面都是0
    memcpy(&num->only_nums[MAX_LEN-i], temp, i*sizeof(int)); // 放到整数里
    i++; // 跳过这个小数点
    for (int j = 0; i < len; i++, j++) {
        num->only_nums[MAX_LEN + j] = str[i] - '0';
    }
    memset(num->only_nums+MAX_LEN+i, 0, (MAX_LEN-i)*sizeof(int));
}

void carry(Number *num) {
    for (int i = MAX_LEN * 2 - 1; i >= 0; i--) {
        if (num->only_nums[i] >= 10) {
            num->only_nums[i-1] += num->only_nums[i] / 10;
            num->only_nums[i] %= 10;
        }
    }
}

void add(Number *a, Number *b, Number *result) {
```

```
    memset(result->only_nums, 0, MAX_LEN*2*sizeof(int));
    for (int i = MAX_LEN*2 - 1; i>=0; i--) {
        result->only_nums[i] += a->only_nums[i] + b->only_nums[i];
    }
    carry(result);
}

void print_number(Number *num, int n) {
    if (num->only_nums[MAX_LEN+n] >= 5) {
        num->only_nums[MAX_LEN+n-1] += 1;
        carry(num);
    }
    int i = 0;
    while(i < MAX_LEN && num->only_nums[i] == 0) i++;
    if (i >= MAX_LEN) {
        putchar('0');
    } else {
        while (i < MAX_LEN) {
            putchar(num->only_nums[i] + '0');
            i++;
        }
    }
    putchar('.');
    for (int j = 0; j < n; j++, i++) {
        putchar(num->only_nums[i] + '0');
    }
}

int main() {
    Number a={{0},0}, b={{0},0}, result={{0},0};
    char str_a[MAX_LEN], str_b[MAX_LEN];
    int n;
    scanf("%s %s %d", str_a, str_b, &n);
    read_number(str_a, strlen(str_a), &a);
    read_number(str_b, strlen(str_b), &b);
    add(&a, &b, &result);
    print_number(&result, n);
}


```

./1057.整数分解.c

```
// 9:20
// 10:29

#include<stdio.h>
#define PRIME_NUM 168
#define MAX_N 1000

```

```cpp
int primes[PRIME_NUM] =
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,
103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,
199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,
313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,
433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541,547,557,
563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,
673,677,683,691,701,709,719,727,733,739,743,751,757,761,769,773,787,797,809,
811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997};
long long plans[MAX_N][PRIME_NUM] = {{0}};

// n在只能用从0到prime_num个数字时的分解方法
long long get_one_plan(int n, int prime_num) {
    long long result = plans[n][prime_num - 1];
    // for (int i = 0; i < PRIME_NUM; i++) {
    //     if (primes[i] == n) {
    //         result ++;
    //         break;
    //     }
    // }
    while (n >= primes[prime_num]) {
        n -= primes[prime_num];
        result += plans[n][prime_num - 1];
    }
    if (n == 0) {
        result ++;
    }
    return result;
}

void get_plans() {
    for (int j = 0; j < PRIME_NUM; j++) {
        // plans[0][j] = 0;
        plans[2][j] = 1;
        plans[3][j] = 1;
    }
    for (int i = 3; i < MAX_N; i++) {
        if (i % 2 == 0) {
            plans[i][0] = 1;
        } else {
            plans[i][0] = 0;
        }
    }
    for (int i = 4; i < MAX_N; i++) {
        for (int j = 1; j < PRIME_NUM; j++) {
            plans[i][j] = get_one_plan(i, j);
        }
    }
}

int main() {
    int n;
    get_plans();
    scanf("%d", &n);
```

```
    printf("%lld", plans[n][PRIME_NUM - 1]);
}

```

./1057.计算a的n次方的大整数.c

```c
// 20:11
// 20:17
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 200

typedef struct
{
    int len;
    int s[MAX+1];
} hp;

void input(hp *a, int x) //读入数字
{
    int i;

    a->len = 0;

    while (x > 0)
    {
        a->s[1 + a->len++] = x % 10;
        x /= 10;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void print(hp *y) //打印数字
{
    int i;
    for (i = y->len; i >= 1; i--)
        printf("%d", y->s[i]);
    printf("\n");
}

void multiplyh(hp *a, hp *b, hp *c) //高精度 * 高精度
{
    int i, j, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    for (i = 1; i <= a->len; i++)
    {
        for (j = 1; j <= b->len; j++)
        {
            c->s[i+j-1] += a->s[i] * b->s[j];
```

```
            c->s[i+j] += c->s[i+j-1] / 10;
            c->s[i+j-1] %= 10;
        }
    }

    len = a->len + b->len + 1;
    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void power(hp *a, int b, hp *c) //高精度乘方c = a ^ b
{
    hp e;

    if (b == 0)
    {
        c->len = 1;
        c->s[1] = 1;
    }
    else if (b == 1)
    {
        memcpy(c, a, sizeof(hp));
    }
    else
    {
        power(a, b / 2, &e);
        multiplyh(&e, &e, c);

        if (b % 2 == 1)
        {
            memcpy(&e, c, sizeof(hp));
            multiplyh(&e, a, c);
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    int a, n;
    for (int i = 0; i < t; i++) {
        scanf("%d%d", &a, &n);
        hp a_hp, result;
        input(&a_hp, a);
        power(&a_hp, n, &result);
        printf("case #%d:\n", i);
        print(&result);
    }
    return 0;
}

```

./1058.一个游戏.c

```c
// 20:18
// 20:27
// 20:36

#include<stdio.h>
#include<string.h>

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int n;
        int head[100];
        int tail[100];
        int a_to_b[100][100];
        scanf("%d", &n);
        memset(head, 0, 100*sizeof(int));
        memset(tail, 0, 100*sizeof(int));
        memset(a_to_b, 0 ,100*100*sizeof(int));
        int justify = 1;
        for (int j = 0; j < n; j++) {
            int ai, bi;
            scanf("%d%d", &ai, &bi);
            head[ai] ++;
            tail[bi] ++;
            if (a_to_b[ai][bi]) { // 去重
                continue;
            } else {
                a_to_b[ai][bi] = 1;
            }
            if (head[ai] > 1 || head[ai] > 0 && tail[ai] > 0
            || head[bi] > 0 && tail[bi] > 0) {
                justify = 0;
            }
        }
        if (justify) {
            printf("Lucky dxw!\n");
        } else {
            printf("Poor dxw!\n");
        }
    }
}

```

./1059.计算a的n次方的大整数.c

```c
// 10:14
// 10:49

#include<stdio.h>
#include<string.h>
#define N 1005

typedef struct {
```

```
    int num[N]; // 从低位到高位
    int cnt; // 有效数字位数
} BigInt;

void init_num(BigInt *num) {
    memset(num->num, 0, N * sizeof(int));
    num->cnt = 0;
}

void read_num(int a, BigInt *result) {
    init_num(result);
    result->num[0] = a;
    result->cnt = 1;
}

void output_num(BigInt *num) {
    for (int i = num->cnt - 1; i >= 0; i--) {
        putchar(num->num[i] + '0');
    }
    putchar('\n');
}

void carry(BigInt *num) {
    int i = 0;
    for (; i < num->cnt; i++) {
        if (num->num[i] >= 10) {
            num->num[i + 1] += num->num[i] / 10;
            num->num[i] %= 10;
        }
    }
    for (; i < N; i++) {
        if (num->num[i] > 0) {
            num->cnt = i + 1;
        }
        if (num->num[i] >= 10) {
            num->num[i + 1] += num->num[i] / 10;
            num->num[i] %= 10;
        }
    }
}

void cpy_num(BigInt *dst, BigInt *src) {
    memcpy(dst, src, sizeof(BigInt));
}

void multiply(BigInt *a, BigInt *b, BigInt *result) {
    init_num(result);
    // int weight = 1;
    for (int i = 0; i < a->cnt; i++) {
        for (int j = 0; j < b->cnt; j++) {
            result->num[i + j] += a->num[i] * b->num[j];
        }
        // weight *= 10;
    }
    carry(result);
```

```
}

void pow(BigInt *a, int n, BigInt *result) {
    // init_num(result);
    BigInt temp;
    read_num(1, result);
    for (int i = 0; i < n; i++) {
        multiply(a, result, &temp);
        cpy_num(result, &temp);
    }
}

int main() {
    int t;
    int a, n;
    BigInt big_a, result;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        scanf("%d%d", &a, &n);
        read_num(a, &big_a);
        pow(&big_a, n, &result);
        output_num(&result);
    }
}

```

./1061.计算n！右端0的个数(II).c

```c
// 8:06
// 8:23
// 8:44

#include<stdio.h>

int get_5_factor_nums_in_factorial(int num) {
    int result = 0;
    for (int i = 1; i <= num; i++) {
        int temp = i;
        while (temp % 5 == 0) {
            temp /= 5;
            result++;
        }
    }
    return result;
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n", i);
```

```
        printf("%d\n", get_5_factor_nums_in_factorial(n));
    }
}

```

./1062.计算2的N次方.c

```c
// 8:59
// 9:02

#include<stdio.h>

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n%d\n", i, 1<<n);
    }
}

```

./1063.二进制倒置.c

```c
// 9:04
// 9:15
// 9:36
// 10:01

#include<stdio.h>
#include<string.h>
#define N 101
#define M 334

// 倒置
typedef struct {
    int s[N];
    int cnt;
} Big;

void init_big(Big *num) {
    memset(num, 0, sizeof(Big));
    // num->cnt = 1; // 初始化为0的时候是0位数字
}

void read(char s[], Big *num) {
    init_big(num);
    for (int i = strlen(s) - 1; i >= 0; i--) {
        num->s[num->cnt++] = s[i] - '0';
    }
}
```

```
28
29  void output(Big *num) {
30      for (int *p = num->s + num->cnt - 1; p >= num->s; p--) {
31          putchar(*p + '0');
32      }
33      putchar('\n');
34  }
35
36  // 顺便更新num->cnt
37  void carry(Big *num) {
38      for (int i = 0; i < N - 1; i++) {
39          if (num->s[i] >= 10) {
40              num->s[i + 1] += num->s[i] / 10;
41              num->s[i] %= 10;
42          }
43      }
44      int i = N - 1;
45      for (; i > 0; i--) {
46          if (num->s[i] != 0) break;;
47      }
48      num->cnt = i + 1; // 如果数字为0, cnt是1
49  }
50
51  // 返回余数
52  int divide_by_2(Big *num) {
53      int borrow = 0;
54      for (int i = num->cnt - 1; i > 0; i--) {
55          borrow += num->s[i];
56          if (borrow < 2){
57              num->s[i] = 0;
58              borrow *= 10;
59              continue;
60          }
61          num->s[i] = borrow >> 1;
62          borrow &= 1;
63          borrow *= 10;
64      }
65      borrow += num->s[0];
66      num->s[0] = borrow >> 1;
67      borrow &= 1;
68      carry(num);
69      return borrow;
70  }
71
72  // 返回二进制位数
73  int read_to_binary(Big *num, int binary[]) {
74      int *p = binary;
75      for (; num->cnt != 1 || num->s[0] != 0; p++) {
76          *p = divide_by_2(num);
77      }
78      return p - binary;
79  }
80
81  void multiply_2(Big *num) {
82      for (int i = 0; i < num->cnt; i++) num->s[i] <<= 1;
```

```
    carry(num);
}

void add_1(Big *num) {
    num->s[0] += 1;
    carry(num);
}

void binary_to_big(int binary[], int binary_len, Big *num) {
    init_big(num);
    num->cnt = 1; // 初始化后作为1处理
    for (int i = 0; i < binary_len - 1; i++) {
        if (binary[i]) add_1(num);
        multiply_2(num);
    }
    if (binary[binary_len - 1]) add_1(num);
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        char s[N + 1];
        scanf("%s", &s);
        Big n;
        read(s, &n);
        int binary[M];
        int binary_len = read_to_binary(&n, binary);
        binary_to_big(binary, binary_len, &n);
        printf("case #%d:\n", i);
        output(&n);
    }
}

```

./1064.A-B(Big Integer).c

```
// 10:11

#include<stdio.h>
#include<string.h>
#define N 500

// 倒置
typedef struct {
    int s[N];
    int cnt;
    int sign;
} Big;

void init(Big *num) {
    memset(num, 0, sizeof(Big));
    num->sign = 1;
}
```

```
18
19  void input(Big *num, char s[]) {
20      init(num);
21      for (char *p = s + strlen(s) - 1; p >= s; p--) {
22          num->s[num->cnt++] = *p - '0';
23      }
24  }
25
26  void output(Big *num) {
27      if (num->sign < 0) putchar('-');
28      for (int i = num->cnt - 1; i > 0; i--) {
29          putchar(num->s[i]);
30      }
31      putchar('\n');
32  }
33
34  void carry(Big *num) {
35      for (int i = 0; i < N - 1; i++) {
36          if (num->s[i] >= 10) {
37              num->s[i + 1] = num->s[i] / 10;
38              num->s[i] /= 10;
39          }
40      }
41      int i = N - 1;
42      for (; i > 0; i--) {
43          if (num->s[i] != 0) break;
44      }
45      num->cnt = i;
46  }
47
48  int cmp(Big *a, Big *b) {
49      if (a->cnt != b->cnt) return a->cnt - b->cnt;
50      for (int i = a->cnt - 1; i >= 0; i--) {
51          if (a->s[i] == b->s[i]) continue;
52          if (a->s[i] > b->s[i]) return 1;
53          else return -1;
54      }
55      return 0;
56  }
57
58  void minus(Big *a, Big *b, Big *result) {
59      init(result);
60      int borrow = 0;
61      for (int i = N - 1; i >= 0; i++) {
62          borrow += a->s[i];
63          if (borrow < b->s[i]) {
64              result->s[i + 1]--;
65              borrow += 10;
66          }
67          result->s[i] = borrow - b->s[i];
68          borrow = 0;
69      }
70      carry(result);
71  }
72
```

```c
int main() {
    char a[N + 1], b[N + 1];
    Big A, B, result;
    while (scanf("%s", a) != EOF) {
        scanf("%s", b);
        input(&A, a);
        input(&B, b);
        if (cmp(&A, &B) < 0) {
            minus(&B, &A, &result);
            result.sign = -1;
        }
        minus(&A, &B, &result);
        output(&result);
    }
}

```

./1065.浮点数减法.c

```c
// 10:34
// 11:55

// 13:32
// 14:53

#include<stdio.h>
#include<string.h>
#define N 600
#define M 50

typedef struct {
    int s[2 * N];
    int cnt; // 最右边的非零数字的位置
    int sign;
} BigFloat;
// 前N位是小数，后N位是整数

void init(BigFloat *num) {
    memset(num, 0, sizeof(BigFloat));
    num->sign = 1;
//    num->cnt = 1;
}

void read_num(BigFloat *num, char *s) {
    init(num);
    if (*s == '-') {
        num->sign = -1;
        s++;
    }
    char *p_point = strchr(s, '.');
    char *end = s + strlen(s);
    int *p_result = num->s + N;
    if (p_point == NULL) p_point = end;
    for (char *p = p_point + 1; p < end; p++) {
```

```
        *--p_result = *p - '0';
    }
    p_result = num->s + N;
    for (char *p = p_point - 1; p >= s; p--) {
        *p_result++ = *p - '0';
    }
//    num->cnt = p - num->s - N;
}

int cmp(BigFloat *a, BigFloat *b) {
    for (int i = 2 * N - 1; i >= 0; i--) {
        if (a->s[i] == b->s[i]) continue;
        return a->s[i] - b->s[i];
    }
    return 0;
}

void carry(BigFloat *num, int n) {
    for (num->cnt = 2 * N - 1; num->cnt > 0; num->cnt--) {
        if (num->s[num->cnt] != 0) break;
    }
    for (int i = 0; i <= num->cnt; i++) {
        if (num->s[i] < 0) {
            num->s[i] += 10;
            num->s[i + 1] --;
        }
        if (num->s[i] >= 10) {
            num->s[i + 1] += num->s[i] / 10;
            num->s[i] %= 10;
        }
    }
    for (num->cnt = 2 * N - 1; num->cnt > 0; num->cnt--) {
        if (num->s[num->cnt] != 0) break;
    }
    if (num->s[num->cnt] < 0) {
        for (int i = 0; i < 2 * N; i++)
            num->s[i] *= -1;
        num->sign = -1;
        carry(num, n);
    }
}

void minus(BigFloat *a, BigFloat *b, BigFloat *result, int n) {
    init(result);
    for (int i = 0; i < 2 * N; i++) {
        result->s[i] = a->s[i] - b->s[i];
    }
//    if (result->s[N - n - 1] >= 5) result->s[N - n]++;
    carry(result, n);
    if (result->s[N - n - 1] >= 5) {
        result->s[N - n]++;
        carry(result, n);
    }
}

```

```c
//void minus_with_cmp(BigFloat *a, BigFloat *b, BigFloat *result) {
//
//}

void output(BigFloat *num, int n){
    int *p = num->s;
    if (num->sign < 0) {
        printf("-");
        p++;
    }
    int i = 2 * N - 1;
    for (; i > N; i--) {
        if (num->s[i] != 0) break;
    }
    for (; i >= N; i--) {
        putchar(num->s[i] + '0');
    }
    putchar('.');
//    int end_0 = 0;
//    for (; end_0 < N; end_0++) {
//        if (num->s[end_0]) break;
//    }
    for (int i = N - 1; i >= N - n; i--) {
        putchar(num->s[i] + '0');
    }
    putchar('\n');
}

int main() {
    char s[N + 1];
    BigFloat a, b, result;
    scanf("%s", s);
    read_num(&a, s);
    scanf("%s", s);
    read_num(&b, s);
    int n;
    scanf("%d", &n);
    minus(&a, &b, &result, n);
    output(&result, n);
    return 0;
}


```

./1081.种田.c

```c
// 9:28
// 9:44

#include<stdio.h>

// long long body_power[;]

long long get_body_power(long long x, long long y) {
```

```c
    long long max_square_num = y / x;
    long long result = 4 * x * max_square_num;
    y %= x;
    if (y <= 0)
        return result;
    // if (y > x)
    //     return result + get_body_power(x, y);
    // else
    return result + get_body_power(y, x);
}

int main() {
    long long x, y;
    scanf("%lld %lld", &x, &y);
    printf("%lld\n", get_body_power(y, x));
    return 0;
}

```

./1082.波兰表达式.c

```c
// 9:47
// 10:20

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

char *current_pos;
char expression[151];

double calculate(char *str) {
    char *operator_;
    operator_ = strtok(str, " ");
    current_pos = operator_ + strlen(operator_) + 1;
    if (strlen(operator_) > 1 || isdigit(*operator_)) { // 说明是个数字
        return atof(operator_);
    }
    double operand1 = calculate(current_pos);
    double operand2 = calculate(current_pos);
    switch (*operator_) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            return operand1 / operand2;
    }
}

int main(int argc, char *argv[]) {
```

```c
    int t;
    scanf("%d", &t);
    getchar(); // 去掉换行
    for (int i = 0; i < t; i++) {
        fgets(expression, 150, stdin);
        printf("case #%d:\n%.6lf\n", i, calculate(expression));
    }
    return 0;
}

```

./1083.地铁站.c

```c
// 10:26
// 10:53

#include <stdio.h>

long long board_people[21] = {0}, unboard_people[21] = {0},
people_aboard[21] = {0};

// t是第二站上下车的人数
long long get_last_unboard_people(int a, int t, int n) {
    board_people[1] = a;
    people_aboard[1] = a;
    board_people[2] = t;
    unboard_people[2] = t;
    people_aboard[2] = a;
    for (int i = 3; i < n; i++) {
        board_people[i] = board_people[i - 1] + board_people[i - 2];
        unboard_people[i] = board_people[i - 1];
        people_aboard[i] = people_aboard[i - 1] + board_people[i] -
unboard_people[i];
    }
    board_people[n] = 0;
    unboard_people[n] = people_aboard[n - 1];
    people_aboard[n] = 0;
    return unboard_people[n];
}

int main() {
    int a, n, m, x;
    scanf("%d %d %d %d", &a, &n, &m, &x);
    for (int t = 0; t < m; t++) { // t从1开始的话最后一个测试用例就不对。
    // 说好的第2站有人上下车呢? 还能有0人上下车的?
        if (m == get_last_unboard_people(a, t, n))
            break;
    }
    printf("%lld\n", people_aboard[x]);
}

```

./1084.Fj&haozi.c

```c
// 8:09
// 8:50

#include<stdio.h>
#include<string.h>
#define MAX_N 101 // 原来写的21导致出错

long long previous_num[MAX_N] = {0};
long long current_num[MAX_N] = {0};

void expand_num(int n, int m) {
    for (int i = 0; i < m; i++) {
        for (int j = 2; j < n; j++) {
            current_num[j] = previous_num[j - 1] + previous_num[j + 1];
        }
        current_num[1] = previous_num[2];
        current_num[n] = previous_num[n - 1];
        memcpy(previous_num, current_num, MAX_N * sizeof(long long));
        memset(current_num, 0, MAX_N * sizeof(long long));
    }
}

int main() {
    int n, m, p, t;
    int cas;
    scanf("%d", &cas);
    while(cas--) {
        scanf("%d %d %d %d", &n, &p, &m, &t);
        memset(previous_num, 0, MAX_N * sizeof(long long));
        memset(current_num, 0, MAX_N * sizeof(long long));
        previous_num[p] = 1;
        expand_num(n, m);
        printf("%lld\n", previous_num[t]);
    }
}


```

./1085.泰波那契数列的前74项.c

```c
// 8:14
// 9:23

#include<stdio.h>

long long tribonacci[74] ={0LL, 1LL, 1LL};

int main(){
    for(int i = 3; i < 74; i++){
```

```
        tribonacci[i] = tribonacci[i - 1] + tribonacci[i - 2] + tribonacci[i - 3];
    }
    int t;
    scanf("%d",&t);
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n%lld\n", i, tribonacci[n]);
    }
    return 0;
}

```

./1087.统计字符串个数.c

```c
// 9:41
// 10:01

#include <stdio.h>
#include <string.h>
#define substring_width 3

int end_with_0[substring_width];
int end_with_1[substring_width];

void init() {
    memset(end_with_0, 0, substring_width * sizeof(end_with_0[0]));
    memset(end_with_1, 0, substring_width * sizeof(end_with_1[0]));
}

void expand(int n) {
    end_with_0[0] = 0;
    end_with_1[0] = 0;
    end_with_0[1] = 1;
    end_with_1[1] = 1;
    for (int i = 2; i < n + 1; i++) {
        end_with_0[2] = end_with_0[1] + end_with_1[1];
        end_with_1[2] = end_with_0[2] - end_with_1[0];
        memmove(end_with_0, end_with_0 + 1, substring_width * sizeof(end_with_0[0]));
        memmove(end_with_1, end_with_1 + 1, substring_width * sizeof(end_with_1[0]));
    }
}

int main() {
    int n = 0;
    while (1) {
        scanf("%d", &n);
        if (n < 0)
            break;
        init();
        expand(n);
```

```
        printf("%d\n", end_with_0[1] + end_with_1[1]);
    }
    return 0;
}

```

./1090.最短路径.c

```c
// 9:19
// 9:39

#include<stdio.h>
#define MAX_N 200

int matrix[MAX_N][MAX_N];
int loss[MAX_N][MAX_N];
char route[MAX_N][MAX_N];

void expand_route(int m, int n) {
    loss[0][0] = matrix[0][0];
    for (int i = 1; i < m; i++) {
        loss[i][0] = loss[i - 1][0] + matrix[i][0];
        route[i][0] = 'D';
    }
    for (int i = 0; i < n; i++) {
        loss[0][i] = loss[0][i - 1] + matrix[0][i];
        route[0][i] = 'R';
    }
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            if (loss[i][j - 1] < loss[i - 1][j]) {
                loss[i][j] = loss[i][j - 1] + matrix[i][j];
                route[i][j] = 'R';
            }
            else {
                loss[i][j] = loss[i - 1][j] + matrix[i][j];
                route[i][j] = 'D';
            }
        }
    }
}

void output_route(int i, int j) {
    if (i == 0 & j == 0) {
        return;
    }
    if (route[i][j] == 'R') {
        output_route(i, j - 1);
        putchar(route[i][j]);
    } else {
        output_route(i - 1, j);
        putchar(route[i][j]);
    }
}
```

```c

int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    expand_route(m, n);
    printf("%d\n", loss[m - 1][n - 1]);
    output_route(m - 1, n - 1);
}

```

./1093.波浪图.c

```c
// 15:00
// 15:23

#include<stdio.h>
#include<string.h>
#define N 80

typedef char Picture[2 * N][2 * N];

void get_pic(Picture pic, char *s) {
    memset(pic, ' ', sizeof(Picture));
    int row = N, col = 0;
    pic[row][col] = *s;
    for (s++; *s; s++) {
        col++;
        if (*s > *(s - 1)) row--;
        if (*s < *(s - 1)) row++;
        pic[row][col] = *s;
    }
}


void output(Picture pic, int len) {
    char blank[2 * N];
    memset(blank, ' ', sizeof(blank));
    int start_row = 0, end_row = 2 * N - 1;
    for (; memcmp(pic[start_row], blank, sizeof(blank)) == 0; start_row++);
    for (; memcmp(pic[end_row], blank, sizeof(blank)) == 0; end_row--);
    for (int i = start_row; i <= end_row; i++) {
        for (int j = 0; j < len; j++) {
            putchar(pic[i][j]);
        }
        putchar('\n');
    }
}

int main() {
```

```
    char s[N + 1];
    Picture pic;
    while (scanf("%s", s) != EOF) {
        get_pic(pic, s);
        output(pic, strlen(s));
    }
}


```

./1094.坏掉的彩灯.c

```c
// 15:07
// 16:19

// 18:38
// 18:51

// 19:18
// 19:19

#include<stdio.h>
#include<string.h>
#define N 100

char Colour[] = "RBYG";
char Index[128];

void init() {
    Index['R'] = 0;
    Index['B'] = 1;
    Index['Y'] = 2;
    Index['G'] = 3;
}

int global_valid = 0;

int get_possible_solution(char s[], char *result, int len) {
//    int len = strlen(s);
    int i = 0;
    for (; i < len; i++) {
        if (s[i] == '!') break;
    }
    // printf("%s\n%d %d\n", result, i, len);
    if (i == len) {
        global_valid = 1;
        memcpy(result, s, (N + 1) * sizeof(char));
        return 1;
    }
    int possible[] = {1, 1, 1, 1}; // RBYG
    for (int j = i - 3; j <= i + 3; j++) {
        if (j >= 0 && j < len && s[j] != '!') {
            possible[Index[s[j]]] = 0;
        }
```

```
    }
    int valid = 0;
    for (int j = 0; j < 4; j++)
        if (possible[j]) valid = 1;
    if (valid == 0) {
//        memcpy(result, s, N); // 还原
        return 0; // 表明这种情况不可能
    }
    for (int j = 0; j < 4; j++) {
        if (possible[j]) {
            char temp[N + 1];
            memcpy(result, s, (N + 1) * sizeof(char));
            result[i] = Colour[j];
            memcpy(temp, result, (N + 1) * sizeof(char));
            global_valid = get_possible_solution(temp, result, len);
            if (global_valid) return 1;
        }
    }
    return 0;
}

int get_broken_nums(char good[], char broken[], int colors[], int len) {
//    int len = strlen(broken);
    for (int i = 0; i < len; i++) {
        if (broken[i] == '!') {
            colors[Index[good[i]]] += 1;
        }
    }
}

int main() {
    init();
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        char s[N + 1], result[N + 1];
        scanf("%s", s);
        int len = strlen(s);
        memcpy(result, s, (N + 1) * sizeof(char));
        global_valid = 0;
        int ret = get_possible_solution(s, result, len);
        int colors[4] = {0};
        get_broken_nums(result, s, colors, len);
        printf("case #%d:\n", i);
        for (int j = 0; j < 3; j++) {
            printf("%d ", colors[j]);
        }
        printf("%d\n", colors[3]);
    }
}


```

./A.进制转换.c

```c
#include<stdio.h>

char str[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main(){
    int t, n, r;
    char result[33];
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        scanf("%d %d", &n, &r);
        if (n < 0) {
            n = - n;
            printf("-");
        }
        int j = 0;
        for (; n; n/=r, j++) result[j] = str[n%r];
        while (--j >= 0) printf("%c", result[j]);
        printf("\n");
    }
}

```

./B.神秘信息.c

```c
#include<stdio.h>

int main(){
    int a[128];
    int t, digit, n;
    long long result;
    char s[61], *p;
    scanf("%d", &t);
    for (int i=0; i<t; i++) {
        printf("case #%d:\n", i);
        for (int i=0; i<128; i++) a[i] = -1;
        scanf("%s", s);
        a[*s] = -1;
        n = 1;
        p = s;
        digit = 0;
        a[*p] = 1;
        while (*++p){
            if (a[*p] != -1) continue;
            a[*p] = digit;
            digit = digit ? digit + 1 : 2;
            n ++;
        }
        if (n < 2) n = 2;
        p = s;
        result = 0;
        while (*p)
            result = result * n + a[*p++];
        printf("%lld\n", result);
```

```
    }
}

```

./C.按数据中1的位数排序.c

```c
#include <stdio.h>
#include <stdlib.h>

# define BIT_NUM 64
# define MAX_N 10000
typedef struct
{
    long long value;
    int number;
} Data;

int cmp(const void *pa, const void *pb){
    Data a = *(Data*)pa;
    Data b = *(Data*)pb;
    if (a.number == b.number) {
        if (a.value == b.value)
            return 0;
        else
            return a.value > b.value ? 1 : -1;
    } else
        return b.number - a.number;
}

int main() {
    int t, n;
    Data nums[MAX_N];
    scanf("%d", &t);
    for (int i=0; i<t; i++) {
        printf("case #%d:\n", i);
        scanf("%d", &n);
        for (int j=0; j<n; j++) {
            scanf("%lld", &nums[j].value);
            nums[j].number = 0;
            long long d = 1;
            for (int k=0; k<BIT_NUM; k++) {
                if (nums[j].value & d) nums[j].number++;
                d <<= 1;
            }
        }
        qsort(nums, n, sizeof(nums[0]), cmp);
        for (int j=0; j<n; j++)
            printf("%lld ", nums[j].value);
        printf("\n");
    }
}

```

./D.内存显示.c

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void solveint(int n){
    for (int i=0; i<sizeof(n); i++){
        printf("%02x ", ((unsigned char*)&n)[i]);
    }
}

void solvedouble(double d){
    for (int i=0; i<sizeof(d); i++){
        printf("%02x ", ((unsigned char*)&d)[i]);
    }
}

int main(){
    char s[31];
    while (scanf("%s", s) != EOF){
        if (strchr(s, '.') == 0)
            solveint(atoi(s));
        else
            solvedouble(atof(s));
        printf("\n");
    }
    return 0;
}

```

./E.平衡三进制.c

```c
#include<stdio.h>

int main(){
    int t;
    char s[30];
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        printf("case #%d:\n", i);
        scanf("%s", s);
        int result = 0;
        for (int j=0; s[j]; j++){
            switch (s[j]){
            case '-':
                result = result * 3 - 1;
                break;
            case '0':
                result = result * 3;
                break;
            case '1':
```

```
                result = result * 3 + 1;
                break;
            }
        }
        printf("%d\n", result);
    }
}

```

./F.非重复二进制串.c

```c
#include<stdio.h>

int main(){
    int t;
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        int n;
        printf("case #%d:\n", i);
        scanf("%d", &n);
        int max_diff_bin_len = 1, diff_bin_len = 1; // 初始值应为1，因为如果大于0的话，一位肯定是不重复的。
        for (; n > 1; n >>= 1){ // 最左的01不能算，因为前置0不计算在内，所以右移到01时就可以停止了，大于1代表可以继续右移。
            switch (n & 0b11){
                case 0b10:
                case 0b01:
                    diff_bin_len ++;
                    if (diff_bin_len > max_diff_bin_len)
                        max_diff_bin_len = diff_bin_len;
                    break;
                case 0b00:
                case 0b11:
                    diff_bin_len = 1;
            }
        }
        printf("%d\n", max_diff_bin_len);
    }
}

```

./G.二进制位不同的个数.c

```c
#include<stdio.h>

int main(){
    int t;
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        int x, y;
        scanf("%d %d", &x, &y);
        int distance = 0;
```

```
        int xor = x ^ y;
        for (; xor > 0; xor >>= 1){
            distance += xor & 1;
        }
        printf("%d\n", distance);
    }
}

```

./H.数据密度.c

```c
#include<stdio.h>
#define MAX_LEN 130

int bin_with_1(char *pc){
    unsigned char c = *(unsigned char*)pc; // 中文的每个字节首位可能是1
    int result = 0;
    do{
        result += c & 1;
    }while (c >>= 1);
    return result;
}

/* 只能计算b > a情况时的最大公因数 */
int gcd(int a, int b){
    if (a <= 0) return b;
    int temp = b % a;
    return gcd(temp, a);
}

int main(){
    char s[500], *pchar;
    int n;
    scanf("%d", &n);
    getchar(); // 因为scanf不会读取换行符，所以要手动把第一个换行符读取了
    while (n--){
        fgets(s, MAX_LEN, stdin); // 会把结尾的换行符也读进来
        // getline(&s, MAX_LEN, stdin);
        int sum_with_bin_1 = 0;
        for (pchar = s; (*pchar != '\n') && *pchar; pchar++){
            sum_with_bin_1 += bin_with_1(pchar);
        }
        int sum_of_bin = (pchar - s) * 8;
        int divisor = gcd(sum_with_bin_1, sum_of_bin);
        printf("%d/%d\n", sum_with_bin_1 / divisor, sum_of_bin / divisor);
    }
}

```

./I.QR Code.c

```c
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
#define NUM_SIGN "0001"
#define MAX_LEN 500
#define GROUP_LEN 3


void itob_print(int a, int bitlen){
    char *result = (char*)malloc(bitlen + 1);
    result[bitlen] = '\0';
    while (--bitlen >= 0){
        result[bitlen] = (a & 1) + '0';
        a >>= 1;
    }
    printf(result);
    free(result);
}

int main(){
    char s[MAX_LEN + 1], *p_char;
    scanf("%s", s);
    printf(NUM_SIGN);
    itob_print(strlen(s), 10);
    int part_len=0, part_num=0;
    for (p_char = s; *p_char; p_char++){
        part_len ++;
        part_num = part_num * 10 + *p_char - '0';
        if (part_len >= GROUP_LEN){ // 每组的数字已完成
            itob_print(part_num, 10);
            part_len = 0;
            part_num = 0;
        }
    }
    switch (part_len){ // 还有剩余的一些数字没输出
        case 1:
            itob_print(part_num, 4);
            break;
        case 2:
            itob_print(part_num, 7);
            break;
    }

}

```

./J.i-1进制（Easy）.c

```c
// 实部
// sqrt s
// 0 s2 2 2s2 4 4s2 8 8s2 16 16s2
// 虚部
// 0 3 6 9 12 15 18 21 24 27 30
// 0 3 6 1  4  7  2  5  0  3  6
```

```c
// 18:04
#include<stdio.h>
#include<string.h>

typedef struct {
    long long real;
    long long imaginary;
} Complex;

/* 计算-1 + i的幂 */
Complex pow_of_complex(int multiple_times) {
    // if (multiple_times <= 0) return {1LL, 0LL};
    long long modulus = 1LL << (multiple_times / 2); // 偶数时是模长的意思，奇数时是实部或虚部的绝对值（实部和虚部绝对值相等）
    Complex result = {0LL, 0LL};
    switch (multiple_times % 8) {
        case 0: // 0 * 45°
            result.real = modulus;
            break;
        case 1: // 3 * 45°
            result.real = - modulus;
            result.imaginary = modulus;
            break;
        case 2: // 6 * 45°
            result.imaginary = - modulus;
            break;
        case 3: // 1 * 45°
            result.real = modulus;
            result.imaginary = modulus;
            break;
        case 4: // 4 * 45°
            result.real = - modulus;
            break;
        case 5: // 7 * 45°
            result.real = modulus;
            result.imaginary = - modulus;
            break;
        case 6: // 2 * 45°
            result.imaginary = modulus;
            break;
        case 7: // 5 * 45°
            result.real = - modulus;
            result.imaginary = - modulus;
            break;
    }
    return result;
}

/* 两个复数即使在不需要二进制进位的情况下相加，也不能直接使用异或，因为负数的时候异或就不对了 */
Complex add(Complex a, Complex b) {
    Complex result;
    result.real = a.real + b.real;
    result.imaginary = a.imaginary + b.imaginary;
    return result;
```

```
}

int main(){
    char str_hex_num[100];
    Complex result = {0LL, 0LL};
    scanf("0x%s", str_hex_num);
    char *pos_hex_num = str_hex_num;
    for (int weight=strlen(str_hex_num) - 1; weight>=0; weight--, pos_hex_num++){
        int hex_num;
        if (*pos_hex_num <= '9')
            hex_num = *pos_hex_num - '0';
        else
            hex_num = *pos_hex_num - 'A' + 10;
        for (int i=0; hex_num>0; hex_num >>= 1, i++){
            if (hex_num & 1)
                result = add(result, pow_of_complex(4 * weight + i));
        }
    }
    if (result.real > 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '1' || *str_hex_num == 'E'))
        result.real --; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.real != 0 && result.imaginary > 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '3' || *str_hex_num == '7' || *str_hex_num == 'E'))
        result.imaginary --; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.imaginary < 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '7' ))
        result.imaginary ++; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.imaginary == 0) {
        printf("%lld", result.real);
    }else if (result.real == 0 ) {
        if (result.imaginary == 1) printf("i");
        else if (result.imaginary == -1) printf("-i");
        else printf("%lldi", result.imaginary);
    }else{
        if (result.imaginary == 1) printf("%lld+i", result.real);
        else if (result.imaginary == -1) printf("%lld-i", result.real);
        else printf("%lld%+lldi", result.real, result.imaginary);
    }
    return 0;
}
// 20:31

```

./K.-1+i进制.c

```
// 21:13
#include<stdio.h>
#include<regex.h>
#include<string.h>
```

```c
#include<stdlib.h>
#include<malloc.h>
#define MAX_E 50 // 最大位数的整数
#define MAX_LINE 100 // 输入的一行最大字符

typedef struct {
    long long real;
    long long imaginary;
} Complex;

regex_t reg_full, reg_real, reg_imaginary;
regmatch_t matches[10];
int ret = 0;
int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
char input[MAX_LINE] = {'\0'};

int getsign(char *input, regmatch_t *match) {
    // printf("%s %d %d\n", input, match->rm_so, match->rm_eo);
    if (match->rm_so < 0 || match->rm_so == match->rm_eo) // 省略正号
        return 1;
    else if (*(input + match->rm_so) == '+')
        return 1;
    return -1;
}

long long getabs(char *input, regmatch_t *match) {
    // printf("%s %d %d\n", input, match->rm_so, match->rm_eo);
    if (match->rm_so < 0 || match->rm_so == match->rm_eo) // 省略1
        return 1;
    char *abs_str = malloc((match->rm_eo - match->rm_so + 1) *
sizeof(char));
    int i;
    for (i = match->rm_so; i < match->rm_eo; i++) {
        abs_str[i - match->rm_so] = input[i];
    }
    abs_str[i - match->rm_so] = '\0'; // 之前这里少了 - match->rm_so
    // char *abs_str = strndup(input + match->rm_so, match->rm_eo - match-
>rm_so); // 使用这个在本地运行不会报错，在OJ运行会Runtime error: SIGSEGV
    long long result = atoll(abs_str);
    free(abs_str);
    return result;
}

Complex get_complex(char *input, int limit) {
    Complex input_complex = {0LL, 0LL};
    fgets(input, limit, stdin);
    ret = regexec(&reg_full, input, 5, matches, 0);
    if (ret) { // 省略了实部或虚部
        if (strchr(input, 'i') == NULL) { // 只有实部
            ret = regexec(&reg_real, input, 3, matches, 0);
            input_complex.real = getsign(input, matches + 1)
                                 * getabs(input, matches + 2);
        } else { // 只有虚部
            ret = regexec(&reg_imaginary, input, 3, matches, 0);
            input_complex.imaginary = getsign(input, matches + 1)
```

```
                                    * getabs(input, matches + 2);
        }
    } else {
        input_complex.real = getsign(input, matches + 1)
                           * getabs(input, matches + 2);
        input_complex.imaginary = getsign(input, matches + 3)
                                * getabs(input, matches + 4);
    }
    return input_complex;
}

Complex divide_once(Complex a) {
    Complex result;
    // >>1是向下取整，/2是向0取整
    result.real = a.imaginary / 2 - a.real / 2;
    result.imaginary = - a.real / 2 - a.imaginary / 2;
    if (a.real & 1) { // 这里已经确保实部虚部奇偶性相同，实部虚部都为奇数时上面两行会偏差1，对负数用%2会出现问题，所以应用&1来判断奇偶
        if (a.real < 0 && a.imaginary < 0)
            result.imaginary ++; // 因为向0取整，所以向远离0的地方偏移1
        if (a.real > 0 && a.imaginary > 0)
            result.imaginary --;
        if (a.real < 0 && a.imaginary > 0)
            result.real ++; // 异号时实部会有偏差
        if (a.real > 0 && a.imaginary < 0)
            result.real --;
    }
    return result;
}

void print_bin_complex(Complex a) {
    if (a.imaginary == 0 && (a.real == 1 || a.real == 0)){
        printf("%lld", a.real);
        return;
    } else if ((a.real & 1) != (a.imaginary & 1)) { // 负奇数对2取余是-1，因此要取绝对值，但用&1来判断就不用取绝对值了
        a.real --;
        print_bin_complex(divide_once(a));
        printf("1");
    } else {
        print_bin_complex(divide_once(a));
        printf("0");
    }
}

// 21:42
// 11:00
// 11:51
// 13:49
// 17:03
// 19:20
// 20:38
int main(){
    // Complex input_complex = {{"0", POS}, {"0", POS}};
```

```
    regcomp(&reg_full, "(-)?([0-9]+)(\\+|-)([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符，不加^来忽略可能不省略的分号
    regcomp(&reg_real, "(-)?([0-9]+)$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    regcomp(&reg_imaginary, "(-)?([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    Complex input_complex = get_complex(input, MAX_LINE - 1);
    // printf("%lld %lld\n", input_complex.real, input_complex.imaginary);
    // Complex temp_complex = get_complex(input, MAX_LINE);
    print_bin_complex(input_complex);
}

```

./K.-1+i进制（长整数加减法未完成）.c

```c
// 21:13
#include<stdio.h>
#include <sys/types.h>
#include <regex.h>
#include<string.h>
#define MAX_E 50 // 最大位数的整数
#define MAX_LINE 100 // 输入的一行最大字符

typedef struct {
    char *abs; // char abs[MAX_E]
    enum {NEG=-1, POS=1} sign;
} Decimal;

typedef struct {
    Decimal real;
    Decimal imaginary;
} Complex;

regex_t reg_full, reg_real, reg_imaginary;

Decimal add(Decimal a, Decimal b) {
    char abs[MAX_E] = "0";
    Decimal result={abs, POS};
    if (a.sign == b.sign) {
        result.sign = a.sign;
        for (int i=MAX_E; i>0; i--) {
            int temp = a.abs[i] + b.abs[i];
            result.abs[i] = temp % 10;
            result.abs[i - 1] = temp / 10;
        }
    } else if (a.abs > b.abs) {
        result.sign = a.sign;
        for (int i=MAX_E; i>0; i--) {
            int temp = a.abs[i] - b.abs[i];
            result.abs[i] += temp % 10;
            result.abs[i - 1] += temp / 10;
        }
    } else if (a.abs < b.abs) {
        result.sign = b.sign;
```

```
        for (int i=MAX_E; i>0; i--) {
            int temp = b.abs[i] - a.abs[i];
            result.abs[i] += temp % 10;
            result.abs[i - 1] += temp / 10;
        }
    }
    return result;
}

int getsign(char *input, regmatch_t *match) {
    if (match->rm_eo == match->rm_so)
        return POS;
    else if (*(input + match->rm_so) == '+')
        return POS;
    return NEG;
}

char *getabs(char *input, regmatch_t *match) {
    if (match->rm_eo == match->rm_so) // 省略1
        return "1";
    return strndup(input + match->rm_so, match->rm_eo - match->rm_so);
}

Complex get_complex(char *input, int limit) {
    int ret;
    regmatch_t matches[5];
    Complex input_complex = {{"0", POS}, {"0", POS}};
    fgets(input, MAX_LINE, stdin);
    ret = regexec(&reg_full, input, 5, matches, 0);
    if (ret) { // 省略了实部或虚部
        if (strchr(input, 'i') == NULL) { // 只有实部
            ret = regexec(&reg_real, input, 3, matches, 0);
            input_complex.real.sign = getsign(input, matches + 1);
            input_complex.real.abs = getabs(input, matches + 2);
        } else { // 只有虚部
            ret = regexec(&reg_imaginary, input, 3, matches, 0);
            input_complex.imaginary.sign = getsign(input, matches + 1);
            input_complex.imaginary.abs = getabs(input, matches + 2);
        }
    } else {
        input_complex.real.sign = getsign(input, matches + 1);
        input_complex.real.abs = getabs(input, matches + 2);
        input_complex.imaginary.sign = getsign(input, matches + 3);
        input_complex.imaginary.abs = getabs(input, matches + 4);
    }
    return input_complex;
}

// 21:42
// 11:00
// 11:51
// 13:49
int main(){
    int ret;
    int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
```

```
    char input[MAX_LINE];
    // Complex input_complex = {{"0", POS}, {"0", POS}};
    regcomp(&reg_full, "(-)?([0-9]+)(\\+|-)([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符，不加^来忽略可能不省略的分号
    regcomp(&reg_real, "(-)?([0-9]+)$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    regcomp(&reg_imaginary, "(-)?([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    Complex input_complex = get_complex(input, MAX_LINE);
    printf("%d %s %d %s\n",
            input_complex.real.sign, input_complex.real.abs,
            input_complex.imaginary.sign, input_complex.imaginary.abs);
    // Complex temp_complex = get_complex(input, MAX_LINE);
    printf("%d %s\n",
            add(input_complex.real, input_complex.imaginary).sign,
            add(input_complex.imaginary, input_complex.real).abs);
}

```

./L.平衡三进制.c

```c
// 9:37
// >>> import math
// >>> math.log(3**30, 10)
// 14.313637641589873

#include<stdio.h>
#include<regex.h>
#include<stdlib.h>
#define MAX_LEN 50
#define BASE 3

/* 计算a < b时a和b的最大公因数 */
long long gcd(long long a, long long b) {
    if (a == 0) return b;
    return gcd(b % a, a);
}

long long tri_to_ll(char *a, int len) {
    long long result = 0;
    for (char *pos_num = a; pos_num < a + len; pos_num++) {
        int num = *pos_num - '0';
        if (num == 2) num = -1;
        result = result * BASE + num;
    }
    return result;
}

int main() {
    char tri[MAX_LEN];
    fgets(tri, MAX_LEN - 1, stdin);
    regex_t reg;
    regmatch_t matches[3];
    int cflags = REG_NEWLINE | REG_EXTENDED | REG_ICASE;
```

```
    regcomp(&reg, "([0-9]+)\\.?([0-9]*)", cflags);
    int ret = regexec(&reg, tri, 3, matches, 0);
    // for (int i=1; i<3; i++) {
    //     if (matches[i].rm_so < 0) continue;
    //     for (int j=matches[i].rm_so; j<matches[i].rm_eo; j++){
    //         printf("%c", tri[j]);
    //     }
    //     printf("\n");
    // }
    long long int_part = tri_to_ll(tri + matches[1].rm_so, matches[1].rm_eo - matches[1].rm_so);
    long long numerator = tri_to_ll(tri + matches[2].rm_so, matches[2].rm_eo - matches[2].rm_so);
    long long denominator = 1;
    for (int point_pos = matches[2].rm_eo - matches[2].rm_so; point_pos > 0; point_pos--) {
        denominator *= 3;
    }
    if (int_part > 0 && numerator < 0) { // 借位
        int_part --;
        numerator += denominator;
    }
    if (int_part < 0 && numerator > 0) { // 借位
        int_part ++;
        numerator = denominator - numerator;
    }
    long long factor = gcd(llabs(numerator), llabs(denominator));
    if (int_part || numerator==0)
        printf("%lld ", int_part);
    if (numerator) {
        printf("%lld %lld", numerator / factor, denominator / factor);
    }
}
// 10:52

```

./M.平衡三进制II.c

```
// 10:53
// >>> math.log(10**9,3)
// 18.86312946860446
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_LEN 50

int get_point_pos(long long denominator) {
    int point_pos = 0;
    while (denominator > 1) {
        denominator /= 3;
        point_pos ++;
    }
    return point_pos;
}
```

```c

/* 十进制转三进制，返回三进制的字符个数 */
int dec_to_tri(long long dec, char *tri) {
    if (dec < 3) {
        *tri = dec + '0';
        return 1;
    }
    int len = dec_to_tri(dec / 3, tri);
    *(tri + len) = dec % 3 + '0';
    return len + 1;
}

/* 注意，这里的target-1必须是一个合法的地址，用来存放进位 */
int add_tri(char *target, char *addtion, int len) {
    int carry = 0; // 存储进位
    for (int i = len - 1; i >= 0; i--) {
        target[i] += addtion[i] - '0' + carry;
        carry = 0;
        if (target[i] >= '3') {
            carry = 1;
            target[i] -= 3;
        }
    }
    if (carry) // 最后一次进位
        target[-1] ++;
}

int sub_ones(char *target, int len) {
    for (char *i = target; i < target + len; i++) {
        (*i) --;
        if (*i < '0') *i = '2';
    }
}

void my_strncpy(char *dest, char *src, int limit) {
    char *p_dest = dest, *p_src = src;
    while (*p_src && limit > 0) {
        *p_dest++ = *p_src++;
        limit --;
    }
    *p_dest = 0;
}

char *simplify_int_part(char *part, int len) {
    while (len > 1 && *part == '0')
        part++, len--;
    return part;
}

int simplify_fraction_part(char *part, int len) {
    while (len > 0 && part[--len] == '0')
        part[len] = 0;
    if (part[len] == '0' || part[len] == 0) return len;
    else return len + 1;
}
```

```
72
73  // 11:43
74  // 12:48
75  // 14:43
76
77  int main() {
78      long long numerator, denominator;
79      scanf("%lld %lld", &numerator, &denominator);
80      char tri[MAX_LEN], ones[MAX_LEN];
81      tri[0] = '0';
82      int len = dec_to_tri(numerator, tri + 1); // 留出第一位可能的进位空间
83      tri[1 + len] = '\0';
84      // printf("%s\n", tri);
85      for (int i = 0; i < len; i++)
86          ones[i] = '1'; // 构造全1序列
87      ones[len] = '\0';
88      add_tri(tri + 1, ones, len);
89      // printf("%s\n", tri);
90      sub_ones(tri + 1, len);
91      // printf("%s\n", tri);
92      int point_pos = get_point_pos(denominator);
93      // char *p_endpos = tri + 1 + len;
94      // char *output = tri;
95      // while (*output == '0') output++; // 去除前面的0
96      // if (output >= p_endpos - point_pos) putchar('0'); // 纯小数
97      // while (output < p_endpos - point_pos) putchar(*output++); // 输出整数
    部分
98      // while (*(p_endpos - 1) == '0') p_endpos--; // 去除后面的0
99      // if (point_pos > 0 && output < p_endpos) { // 判断是否需要输出小数部分
100     //     putchar('.');
101     //     for (int i = point_pos - (p_endpos - output); i > 1; i--) {
102     //         putchar('0'); // 纯小数
103     //     }
104     //     while (output < p_endpos) putchar(*output++);
105     // }
106     int full_len = len + 1; // 三进制数字总长度
107     char tri_with_start_0[MAX_LEN];
108     int i = 0;
109     for (; i < point_pos - full_len + 1; i++) {
110         tri_with_start_0[i] = '0';
111     }
112     int j = 0;
113     while (tri[j]) tri_with_start_0[i++] = tri[j++];
114     tri_with_start_0[i] = 0;
115     full_len = strlen(tri_with_start_0);
116     char int_part[MAX_LEN], fraction_part[MAX_LEN];
117     my_strncpy(int_part, tri_with_start_0, full_len - point_pos);
118     my_strncpy(fraction_part, tri_with_start_0 + full_len - point_pos,
    point_pos);
119     char *new_int_part = simplify_int_part(int_part, strlen(int_part));
120     int fraction_len = simplify_fraction_part(fraction_part,
    strlen(fraction_part));
121     printf("%s", new_int_part);
122     if (fraction_len > 0)
123         printf(".%s",fraction_part);
```

```
    return 0;
}


```

./library/大整数计算1.c

```
// https://blog.csdn.net/u013113678/article/details/113060506

#include<stdint.h>
#include<stdio.h>
#include<string.h>
//计算位数，范围[1,9]
#define NUM_DIGIT 9
//数组类型
#if NUM_DIGIT<=0
    static_assert(1, "NUM_DIGIT must > 0 and <=9");
#elif NUM_DIGIT <=2
#define _NUM_T int8_t
#elif NUM_DIGIT <=4
#define _NUM_T int16_t
#elif NUM_DIGIT <=9
//NUM_DIGIT在[5,9]时用int32就可以装载数据，但是实测发现用int32在32位程序中，对除法性能有影响，用int64反而正常，这是比较奇怪的现象。
#define _NUM_T int64_t
#else
    static_assert(1, "NUM_DIGIT must > 0 and <=9");
#endif
//计算进制
    static const size_t _hex = NUM_DIGIT == 1 ? 10 : NUM_DIGIT == 2 ? 100 : NUM_DIGIT == 3 ? 1000 : NUM_DIGIT == 4 ? 10000 : NUM_DIGIT == 5 ? 100000 : NUM_DIGIT == 6 ? 1000000 : NUM_DIGIT == 7 ? 10000000 : NUM_DIGIT == 8 ? 100000000 : NUM_DIGIT == 9 ? 1000000000 : -1;
#define _MIN_NUM_SIZE 30/NUM_DIGIT

    /// <summary>
    /// 通过字符串初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]字符串首地址</param>
    /// <param name="len">[in]字符串长度</param>
    /// <returns>数据长度</returns>
    static size_t initByStr(_NUM_T* a, const char* value, size_t len)
    {
        size_t  shift = 10;
        size_t i, j, k = 0, n;
        n = len / NUM_DIGIT;
        for (i = 0; i < n; i++)
        {
            a[i] = (value[len - k - 1] - '0');
            for (j = 1; j < NUM_DIGIT; j++)
            {
                a[i] += (value[len - k - j - 1] - '0') * shift;
                shift *= 10;
```

```
            }
            shift = 10;
            k += NUM_DIGIT;
        }
        if (n = len - n * NUM_DIGIT)
        {
            a[i] = (value[len - k - 1] - '0');
            for (j = 1; j < n; j++)
            {
                a[i] += (value[len - k - j - 1] - '0') * shift;
                shift *= 10;
            }
            i++;
        }
        return i;
    }
    /// <summary>
    /// 通过无符号32整型初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]整型值</param>
    /// <returns>数据长度</returns>
    static size_t initByInt32(_NUM_T* a, uint32_t value)
    {
        size_t i;
        for (i = 0; i < 8096; i++)
        {
            a[i] = value % _hex;
            value /= _hex;
            if (!value)
            {
                i++;
                break;
            }
        }
        return i;
    }
    /// <summary>
    /// 通过无符号64整型初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]整型值</param>
    /// <returns>数据长度</returns>
    static size_t initByInt64(_NUM_T* a, uint64_t value)
    {
        size_t i;
        for (i = 0; i < 8096; i++)
        {
            a[i] = value % _hex;
            value /= _hex;
            if (!value)
            {
                i++;
                break;
            }
```

```
        }
        return i;
    }
    /// <summary>
    /// 比较两个高精度数的大小
    /// </summary>
    /// <param name="a">[in]第一个数</param>
    /// <param name="aLen">[in]第一个数的长度</param>
    /// <param name="b">[in]第二个数</param>
    /// <param name="bLen">[in]第二个数的长度</param>
    /// <returns>1是a>b,0是a==b,-1是a<b</returns>
    static int compare(const _NUM_T* a, size_t aLen, const  _NUM_T* b,
size_t bLen)
    {
        if (aLen > bLen)
        {
            return 1;
        }
        else if (aLen < bLen)
        {
            return -1;
        }
        else {
            for (int i = aLen - 1; i > -1; i--)
            {
                if (a[i] > b[i])
                {
                    return 1;
                }
                else if (a[i] < b[i])
                {
                    return -1;
                }
            }
        }
        return 0;
    }
    /// <summary>
    /// 打印输出结果
    /// </summary>
    static void print(const _NUM_T* a, size_t aLen) {
        int i = aLen - 1, j = 0, n = _hex;
        char format[32];
        sprintf(format, "%%0%dd", NUM_DIGIT);
        printf("%d", a[i--]);
        for (; i > -1; i--)
            printf(format, a[i]);
    }

    /// <summary>
    /// 加法(累加)
    ///结果会保存在a中
    /// </summary>
    /// <param name="a">[in]被加数</param>
    /// <param name="aLen">[in]被加数长度</param>
```

```
153     /// <param name="b">[in]加数</param>
154     /// <param name="bLen">[in]加数长度</param>
155     /// <returns>结果长度</returns>
156     static  size_t acc(_NUM_T* a, size_t aLen, const _NUM_T* b, size_t
bLen)
157     {
158         size_t i, n = bLen;
159         const size_t max = _hex - 1;
160         if (aLen < bLen)
161         {
162             memset(a + aLen, 0, (bLen - aLen + 1) * sizeof(_NUM_T));
163         }
164         else {
165             a[aLen] = 0;
166         }
167         for (i = 0; i < bLen; i++) {
168             uint64_t temp = (uint64_t)a[i] + (uint64_t)b[i];
169             a[i] = temp % _hex;
170             a[i + 1] += temp / _hex;
171         }
172         for (; i < aLen; i++) {
173             uint64_t temp = a[i];
174             if (temp <= max)
175                 break;
176             a[i] = temp % _hex;
177             a[i + 1] += temp / _hex;
178         }
179         n = aLen > bLen ? aLen : bLen;
180         if (a[n])
181             return n + 1;
182         return n;
183     }
184
185 /// <summary>
186     /// 减法(累减)
187     ///结果会保存在a中
188     /// </summary>
189     /// <param name="a">[in]被减数，被减数必须大于等于减数</param>
190     /// <param name="aLen">[in]被减数长度</param>
191     /// <param name="b">[in]减数</param>
192     /// <param name="bLen">[in]减数长度</param>
193     /// <returns>结果长度</returns>
194     static  size_t subc(_NUM_T* a, size_t aLen, const _NUM_T* b, size_t
bLen) {
195         size_t  m, n, i;
196         if (aLen < bLen)
197         {
198             m = bLen;
199             n = aLen;
200         }
201         else {
202             n = bLen;
203             m = aLen;
204         }
205         for (i = 0; i < n; i++)
```

```
        {
            int64_t temp = (int64_t)a[i] - (int64_t)b[i];
            a[i] = temp;
            if (temp < 0)
            {
                a[i + 1] -= 1;
                a[i] += _hex;
            }
        }
        for (; i < m; i++)
        {
            _NUM_T temp = a[i];
            if (temp < 0)
            {
                a[i + 1] -= 1;
                a[i] += _hex;
            }
            else
            {
                break;
            }
        }
        for (size_t i = aLen - 1; i != SIZE_MAX; i--)
            if (a[i])
                return i + 1;
        return 1;
    }

/// <summary>
    /// 乘法
    /// </summary>
    /// <param name="a">[in]被乘数</param>
    /// <param name="aLen">[in]被乘数长度</param>
    /// <param name="b">[in]乘数</param>
    /// <param name="bLen">[in]乘数长度</param>
    /// <param name="c">[out]结果，数组长度必须大于等于aLen+bLen+1，且全部元素为0</param>
    /// <returns>结果长度</returns>
    static  size_t multi(const _NUM_T* a, size_t aLen, const  _NUM_T* b, size_t bLen, _NUM_T c[]) {
        _NUM_T  d;
        size_t j;
        c[aLen + bLen] = 0;
        for (size_t i = 0; i < aLen; i++)
        {
            d = 0;
            for (j = 0; j < bLen; j++)
            {
                uint64_t temp = (uint64_t)a[i] * (uint64_t)b[j] + c[j + i] + d;
                d = temp / _hex;
                c[j + i] = temp % _hex;
            }
            if (d)
            {
```

```
                c[j + i] = d;
            }
        }
        for (size_t k = aLen + bLen; k != SIZE_MAX; k--)
            if (c[k])
                return k + 1;
        return 1;
    }

/// <summary>
    /// 除法
    /// </summary>
    /// <param name="a">[in]被除数,被除数必须大于除数。小于商为0、余数为除数，等于商为1、余数为0，不需要在函数内实现，在外部通过compare就可以做到。</param>
    /// <param name="aLen">[in]被除数长度</param>
    /// <param name="b">[in]除数</param>
    /// <param name="bLen">[in]除数长度</param>
    /// <param name="c">[out]商，数组长度大于等于aLen，且全部元素为0</param>
    /// <param name="mod">[out]余数，数组长度大于等于aLen</param>
    /// <param name="modLen">[out]余数长度</param>
    /// <param name="temp">[in]临时缓冲区，由外部提供以提高性能，数组长度大于等于aLen+bLen+1</param>
    /// <returns>商长度</returns>
        static  size_t divide(const _NUM_T* a, size_t aLen, const  _NUM_T* b, size_t bLen, _NUM_T* c, _NUM_T* mod, size_t* modLen, _NUM_T* temp) {
        intptr_t  n, l;
        int equal;
        memcpy(mod, a, (aLen) * sizeof(_NUM_T));
        n = aLen - bLen;
        l = aLen;
        while (n > -1)
        {
            equal = compare(mod + n, l - n, b, bLen);
            if (equal == -1)
            {
                n--;
                if (!mod[l - 1])
                    l--;
                continue;
            }
            if (equal == 0)
            {
                c[n]++;
                n -= bLen;
                l -= bLen;
                continue;
            }
            uint64_t x;
            if ((l - n) > bLen)
            {
                x = ((mod[l - 1]) * _hex) / ((uint64_t)b[bLen - 1] + 1);
                if (x == 0)
                {
                    x = (mod[l - 2]) / ((uint64_t)b[bLen - 1] + 1);
                }
```

```
            }
            else
            {
                x = (mod[l - 1]) / ((uint64_t)b[bLen - 1] + 1);
            }
            if (x == 0)
                x = 1;
            c[n] += x;
            if (x == 1)
            {
                l = n + subc(mod + n, l - n, b, bLen);
            }
            else
            {
                _NUM_T num[_MIN_NUM_SIZE];
                size_t len = initByInt32(num, x);
                memset(temp, 0, (aLen + bLen + 1) * sizeof(_NUM_T));
                len = multi(b, bLen, num, len, temp);
                l = n + subc(mod + n, l - n, temp, len);
            }
        }
        intptr_t i = l - 1;
        for (; i > -1; i--)
            if (mod[i])
                break;
        if (i == -1)
        {
            mod[0] = 0;
            *modLen = 1;
        }
        else
        {
            *modLen = i + 1;
        }
        if (c[aLen - bLen] != 0)
            return aLen - bLen + 1;
        else
            return aLen - bLen;
    }

int sum(){
    int64_t n;
    size_t len, len2;
    _NUM_T num[1024];
    _NUM_T num2[1024];
    scanf("%ld", &n);
    len = initByInt32(num, 0);
    for (int64_t i = 1; i <= n; i++)
    {
        len2 = initByInt64(num2, i);
        len = acc(num, len, num2, len2);
    }
    print(num, len);
    return 0;
}
```

```
365
366 int minus()
367 {
368     _NUM_T a1[8096], a2[8096];
369     char s1[8096], s2[8096];
370     scanf("%s%s", s1, s2);
371     size_t len1,len2;
372     len1 =initByStr(a1, s1, strlen(s1));
373     len2 = initByStr(a2, s2, strlen(s2));
374     len1 =subc(a1, len1, a2, len2);
375     print(a1,len1);
376     return 0;
377 }
378
379 int factorial(){
380     int64_t n;
381     size_t len, len2;
382     _NUM_T num[8096];
383     _NUM_T num2[8096];
384     _NUM_T num3[8096];
385     _NUM_T* p1 = num;
386     _NUM_T* p2 = num3;
387     scanf("%ld", &n);
388     len = initByInt32(num,1);
389     for (int64_t i = 1; i <= n; i++)
390     {
391         len2 = initByInt64(num2, i);
392         memset(p2, 0, 8096* sizeof(_NUM_T));
393         len = multi(p1, len, num2, len2, p2);
394         _NUM_T* temp = p1;
395         p1 = p2;
396         p2 = temp;
397     }
398     print(p1, len);
399     return 0;
400 }
401
402 int division()
403 {
404     _NUM_T a1[8096], a2[8096],c[8096],mod[8096], temp[8096];
405     char s1[8096], s2[8096];
406     scanf("%s%s", s1, s2);
407     size_t len1,len2,ml;
408     len1 =initByStr(a1, s1, strlen(s1));
409     len2 = initByStr(a2, s2, strlen(s2));
410     memset(c, 0, 8096 * sizeof(_NUM_T));
411     len1 =divide(a1, len1, a2, len2,c, mod,&ml, temp);
412     print(c,len1);
413     printf("\n");
414     print(mod, ml);
415     return 0;
416 }
417
418 int main() {
```

```
    printf("求 n 累加和(ja)。用高精度方法，求 s=1+2+3+……+n 的精确值(n 以一般整数输入)。\n");
    sum();
    printf("\n两个任意十一位数的减法；（小于二十位）\n");
    minus();
    printf("\n求 N! 的值(ni)。用高精度方法，求 N! 的精确值(N 以一般整数输入)。\n");
    factorial();
    printf("\n给定两个非负整数A，B，请你计算 A / B的商和余数。\n");
    division();
}

```

./library/大整数计算2.c

```c
// https://www.cnblogs.com/HappyXie/articles/1927557.html

#include <stdio.h>
#include <stdlib.h>
#define MAX 200

typedef struct
{
    int len;
    int s[MAX+1];
} hp;

void input(hp *a, int x) //读入数字
{
    int i;

    a->len = 0;

    while (x > 0)
    {
        a->s[1 + a->len++] = x % 10;
        x /= 10;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void input1(hp *a, char *str) //读入字符串
{
    int i, len;

    a->len = 0;

    if (str == NULL)
        return;

    len = strlen(str);

    while (len > 0)
```

```
    {
        a->s[1 + a->len++] = *(str + len - 1) - '0';
        len--;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void print(hp *y) //打印数字
{
    int i;
    for (i = y->len; i >= 1; i--)
        printf("%d", y->s[i]);
    printf("\n");
}

void add(hp *a, hp *b, hp *c) //高精度加法c = a + b
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] + b->s[i];
        if (c->s[i] >= 10)
        {
            c->s[i] -= 10;
            c->s[i+1]++;
        }
    }

    if (c->s[len+1] > 0) len++;
    c->len = len;
}

void subtract(hp *a, hp *b, hp *c) //高精度减法c = a - b
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] - b->s[i];
        if (c->s[i] < 0)
        {
            c->s[i] += 10;
```

```
            c->s[i+1]--;
        }
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

int compare(hp *a, hp *b) //高精度比较
{
    int len;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    while (len > 0 && a->s[len] == b->s[len]) len--;

    if (len == 0) return 0;
    else return a->s[len] - b->s[len];
}

void multiply(hp *a, int b, hp *c) //高精度 * 单精度
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;
    len = a->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] * b;
        c->s[i+1] += c->s[i] / 10;
        c->s[i] %= 10;
    }

    len++;
    while (c->s[len] >= 10)
    {
        c->s[len+1] += c->s[len] / 10;
        c->s[len] %= 10;
        len++;
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void multiplyh(hp *a, hp *b, hp *c) //高精度 * 高精度
{
    int i, j, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    for (i = 1; i <= a->len; i++)
    {
```

```
        for (j = 1; j <= b->len; j++)
        {
            c->s[i+j-1] += a->s[i] * b->s[j];
            c->s[i+j] += c->s[i+j-1] / 10;
            c->s[i+j-1] %= 10;
        }
    }

    len = a->len + b->len + 1;
    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void power(hp *a, int b, hp *c) //高精度乘方c = a ^ b
{
    hp e;

    if (b == 0)
    {
        c->len = 1;
        c->s[1] = 1;
    }
    else if (b == 1)
    {
        memcpy(c, a, sizeof(hp));
    }
    else
    {
        power(a, b / 2, &e);
        multiplyh(&e, &e, c);

        if (b % 2 == 1)
        {
            memcpy(&e, c, sizeof(hp));
            multiplyh(&e, a, c);
        }
    }
}

void divide(hp *a, int b, hp *c, int *d) //高精度 / 单精度 {d为余数}
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;
    len = a->len;
    *d = 0;

    for (i = len; i >= 1; i--)
    {
        *d = *d * 10 + a->s[i];
        c->s[i] = *d / b;
        *d %= b;
    }

    while (len > 1 && c->s[len] == 0) len--;
```

```
    c->len = len;
}

void multiply10(hp *a) //高精度 * 10
{
    int i;
    for (i = a->len; i >= 1; i--)
        a->s[i+1] = a->s[i];

    a->s[1] = 0;
    a->len++;
    while (a->len > 1 && a->s[a->len] == 0) a->len--;
}

void divideh(hp *a, hp *b, hp *c, hp *d) //高精度 / 高精度{d为余数}
{
    hp e;
    int i, len;

    for (i = 1; i <= MAX; i++)
    {
        c->s[i] = 0;
        d->s[i] = 0;
    }

    len = a->len;
    d->len = 1;

    for (i = len; i >= 1; i--)
    {
        multiply10(d);
        d->s[1] = a->s[i];

        while (compare(d, b) >= 0)
        {
            subtract(d, b, &e);
            *d = e;
            c->s[i]++;
        }
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

```

./他人做法/1009.QR Code.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
int main()
{
```

```
    int i,j,c,d,a,b;char s[501],t[33333],u[3];
    scanf("%s",s);
    c=13;
    int len = strlen(s);
    for(i=0;i<len;c=c+10)
    {
        if(i+3>len) break;
        for(j=0;j<3;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+10;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';

    }
    b=strlen(s);
    if((b-i)==2)
    {
        for(j=0;j<2;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+7;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';
        t[c+8]='\0';
    }
    else if((b-i)==1)
    {
        for(j=0;j<1;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+4;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';
        t[c+5]='\0';
    }
    else
    {
        t[c+1]='\0';
    }
```

```
    for(d=13;b!=0;d--)
    {
        t[d]=b%2+'0';b/=2;
    }
    for(;d>3;d--) t[d]='0';
    t[0]='0';t[1]='0';t[2]='0';t[3]='1';
    printf("%s",t);
    return 0;
}

```

./他人做法/1064.A-B.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
void Input(int a[])
{
    char s[502];
    scanf("%s", s);
    int i = 0, j, k;
    j = strlen(s) - 1;
    for (k = 501; j >= 0; j--)
    {
        a[k] = s[j] - '0';
        k--;
    }
}
int main()
{
    while (1)
    {
        int a[502] = {0};
        int b[502] = {0};
        Input(a);
        Input(b);
        int m = 0, n = 0;
        int j = 0;
        while (a[m] == 0)
        {
            m++;
        }
        while (b[n] == 0)
        {
            n++;
        }
        if (n > m)
        {
            for (int i = 501; i >= m; i--)
            {
                if (a[i] - b[i] >= 0)
                {
                    a[i] = a[i] - b[i];
```

```
                }
                else
                {
                    a[i] = a[i] + 10 - b[i];
                    a[i - 1]--;
                }
            }
            while (a[j] == 0)
            {
                j++;
            }
            for (j; j <= 501; j++)
            {
                printf("%d", a[j]);
            }
            printf("\n");
        }

        if (m > n)
        {
            for (int i = 501; i >= n; i--)
            {
                if (b[i] - b[i] >= 0)
                {
                    b[i] = b[i] - a[i];
                }
                else
                {
                    b[i] = b[i] + 10 - a[i];
                    b[i - 1]--;
                }
            }
            while (b[j] == 0)
            {
                j++;
            }
            printf("-");
            for (j; j <= 501; j++)
            {
                printf("%d", b[j]);
            }
            printf("\n");
        }
        else
        {
            if (a[m] >= b[n])
            {
                for (int i = 501; i >= 0; i--)
                {
                    if (a[i] - b[i] >= 0)
                    {
                        a[i] = a[i] - b[i];
                    }
                    else
                    {
```

```
                        a[i] = a[i] + 10 - b[i];
                        a[i - 1]--;
                    }
                }
                while (a[j] == 0)
                {
                    j++;
                }
                for (j; j <= 501; j++)
                {
                    printf("%d", a[j]);
                }
                printf("\n");
            }
            else
            {
                for (int i = 501; i >= 0; i--)
                {
                    if (b[i] - b[i] >= 0)
                    {
                        b[i] = b[i] - a[i];
                    }
                    else
                    {
                        b[i] = b[i] + 10 - a[i];
                        b[i - 1]--;
                    }
                }
                while (b[j] == 0)
                {
                    j++;
                }
                printf("-");
                for (j; j <= 501; j++)
                {
                    printf("%d", b[j]);
                }
                printf("\n");
            }
        }
    }

    return 0;
}


```

./他人做法/1096.移动游戏.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
long long solution(long long t,long long r)                //单个方程求解m
{
```

```
    if (t==0&&r!=0) return -2;              //无解
    if (t==0&&r==0) return -1;             //任意解
    if (r%t!=0||r/t<0) return -2;            //解为负数或非整数，非法解
    return r/t;                                          //返回解
}

void Move(long long*left,long long*right,char c)
{
    int dx[4]={0,0,-1,1};
    int dy[4]={1,-1,0,0};
    int hash[300]={0};
    hash['U']=0;
    hash['D']=1;
    hash['L']=2;
    hash['R']=3;
    *left=*left+dx[hash[c]];
    *right=*right+dy[hash[c]];
}
typedef struct
{
long long x;
long long y;
}p;
int main()
{
    char l[110]={'\0'};
    scanf("%s",l);
    int len=strlen(l);
    p pos[110]={{0,0}};
    for(int i=1;i<=len;i++)
    {
        long long a=0,b=0;
        for(int j=0;j<i;j++)
        Move(&a,&b,l[j]);
        pos[i].x=a;
        pos[i].y=b;
    }
    int x;
    scanf("%d",&x);
    for(int i=0;i<x;i++)
    {
        int flag=0;
        long long a,b;
        scanf("%lld%lld",&a,&b);
        for(int j=0;j<=len;j++)
        {
           long long m1=solution(pos[len].x,a-pos[j].x);    //求解第一个方程m1
            long long m2=solution(pos[len].y,b-pos[j].y);       //求解第二个方程
m2
            if (m1 == -2||m2 == -2) continue;
     //无解情况
            if (m1 == -1||m2 == -1||m1 == m2)                 //判断两坐标轴的解
是否相同
            {   flag=1;  break;  }
        }
```

```
        if(flag) printf("Yes\n");
        else printf("No\n");
    }
}
// 这是移动游戏，可以看看为啥一直不对嘛

```

./合并后.md

```
# ./1014.罗马数字.c
```C
// 9:37

#include<stdio.h>
#include<regex.h>
#include<string.h>
#define MAX_LEN 51
#define ROMAN_LEN 7

int roman_to_num[128] = {0};
char romans[] = "MDCLXVI";
regex_t parse_brackets_reg;
int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
typedef struct {
    char str[MAX_LEN];
    long long value;
    int weight; // 括号的层数
} Group;


void init_roman_numericals() {
    roman_to_num['I'] = 1;
    roman_to_num['V'] = 5;
    roman_to_num['X'] = 10;
    roman_to_num['L'] = 50;
    roman_to_num['C'] = 100;
    roman_to_num['D'] = 500;
    roman_to_num['M'] = 1000;
}

long long parse_no_brackets(char input[]) {
    long long values_of_pos[MAX_LEN];
    int i = 0;
    long long sum = 0;
    for (; input[i]; i++) {
        values_of_pos[i] = roman_to_num[input[i]];
    }
    for (i--; i > 0; i--) { // 刚开始i在最后\0的位置，要减一
        if (values_of_pos[i - 1] < values_of_pos[i]) {
            values_of_pos[i - 1] = - values_of_pos[i - 1];
        }
    }
    for (i = 0; input[i]; i++) {
        sum += values_of_pos[i];
```
```

```
    }
    return sum;
}

/* 返回解析得到的标识符个数 */
int parse_num(char input[], Group group[]) {
    // regmatch_t identifiers[MAX_LEN];
    // int ret = regexec(&parse_brackets_reg, input, MAX_LEN, identifiers,
0);
    // Group current_group = {"", 0};
    int current_group_index = 0;
    int current_str_index = 0;
    int current_weight = 0;
    group[current_group_index].weight = 0;// 一开始也得初始化，不然第一个组没有初
始化了。
    // int input_len = strlen(input);
    for (int i = 0; i < input[i]; i++) {
        if (input[i] == '(') {
            current_weight ++;
            group[current_group_index].weight = current_weight;
        } else if (input[i] == ')') {
            current_weight --;
        } else {
            group[current_group_index].str[current_str_index] = input[i];
            current_str_index ++;
        }
        if (current_weight == 0) { // 括号层级降到0时开新组
            group[current_group_index].str[current_str_index] = 0;
            group[current_group_index].value =
parse_no_brackets(group[current_group_index].str);
            for (int j = 0; j < group[current_group_index].weight; j++)
                group[current_group_index].value *= 1000; // 值乘权重
            current_group_index ++;
            group[current_group_index].weight = 0;
            current_str_index = 0;
        }
    }
    return current_group_index;
}

long long parse_group(Group group[], int group_len) {
    long long sum = 0;
    for (int i = 1; i < group_len; i++) {
        if (group[i - 1].value < group[i].value) {
            group[i - 1].value = - group[i - 1].value;
        }
    }
    for (int i = 0; i < group_len; i++) {
        sum += group[i].value;
    }
    return sum;
}

int main() {
    char input[MAX_LEN];
```

```
    Group output[MAX_LEN];
    init_roman_numericals();
    // regcomp(&parse_brackets_reg, "\\(.+\\)", cflags); // 好像非贪婪匹配不出来
    scanf("%s", input);
    int group_len = parse_num(input, output);
    long long result = parse_group(output, group_len);
    printf("%lld\n", result);
    // for (int i = 0; i < group_len; i++) {
    //     printf("%s %lld %d\n", output[i].str, output[i].value, output[i].weight);
    // }
    // 395
    // C 100 -1258319696
    // C 100 0
    // C 100 0
    // X -10 0
    // C 100 0
    // V 5 0
    return 0;
}

```

./1015.八进制小数.c

```c
// 20:44
// 17:57
// 18:30
#include<stdio.h>
#include<string.h>
#define MAX_LEN 200

void divided_by_8(char *input, char *result) {
    int temp = 0;
    int i = 0; // 字符索引
    while (input[i]) {
        temp = temp * 10 + input[i] - '0';
        if (temp >= 8) {
            result[i] = (temp >> 3 )+ '0'; // /8
            temp &= 0b111; // %8
        }
        else result[i] = '0';
        i ++;
    }
    while (temp) {
        temp *= 10;
        result[i] = (temp >> 3 )+ '0'; // /8
        temp &= 0b111; // %8
        i ++;
    }
    result[i] = 0; // 结束字符串
}

int main() {
```

```
    int t;
    scanf("%d", &t);
    for (int j = 0; j < t; j++) {
        char input[MAX_LEN], result[MAX_LEN], temp[MAX_LEN] = "0";
        scanf("%s", input);
        int len = strlen(input);
        for (int i = len - 1; i >= 2; i--) { // 前面两位肯定是0., 不用管
            temp[0] = input[i]; // 整数位从0变成新增的0~7之间的数字
            divided_by_8(temp, result);
            strcpy(temp, result);
        }
        printf("case #%i:\n0.%s\n", j, result + 1);
    }
}

```

./1016.负基数进制转换.c

```c
// 18:34
// 19:15
#include<stdio.h>
#include<stdlib.h>

const char *base_symbol = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

void change_base(int num, int base) {
    int num_abs = abs(num);
    int base_abs = abs(base);
    int base_sign = base == 0 ? 0 : base > 0 ? 1 : -1; // 正负号
    // if (num_abs < base_abs) { // 因为num会不断转换正负，所以要用绝对值来判断。
    //     printf("%c", base_symbol[num_abs]);
    //     return;
    // }
    int quotient = num / base; // 商
    int remainder = num % base; // 这里余数可能是负数，要借位变成正数
    if (remainder < 0) {
        quotient -= base_sign; // quotient * base 整体应加上一个负数，如果base < 0 , quotient应该增加，如果 base > 0 ，  quotient应该减小
        remainder += base_abs; // 加上了一个正数
    }
    if (quotient == 0) {
        printf("%c", base_symbol[remainder]);
        return;
    }
    // int new_num = sign * num_abs / base_abs; // 直接除是向0取整，但这里因为末尾的位权是1，将末尾变成0也就是减去末尾的数（正数），也就是向下取整
    change_base(quotient, base);
    printf("%c", base_symbol[remainder]);
}

int main(){
    int num, base;
    scanf("%d %d", &num, &base);
    change_base(num, base);
```

```
}

```

./1017.素数进制A+B.c

```c
// 21:16
// 22:29

#include<stdio.h>
#include<stdlib.h>
#define MAX_LEN 200 // 一个素数最多是多少位的字符串
#define MAX_PRIME_NUM 26

const int prime[MAX_PRIME_NUM]=
{97,91,89,83,79,73,71,67,61,59,53,47,43,41,37,31,29,23,19,17,13,11,7,5,3,2};

/* 返回读入的个数 */
int read_number(char *s, const int *prime_array) {
    int *readed_array = prime_array;
    char temp[5] = ""; // 存储每个位值
    int i = 0; // temp的索引
    while (*s) {
        if (*s == ',') {
            temp[i] = 0; // 结束字符串
            *readed_array++ = atoi(temp); // 转为整数存到数组里
            i = 0;
            s++;
            continue;
        }
        temp[i++] = *s++; // 字符存到temp里
    }
    temp[i] = 0; // 结束字符串
    *readed_array++ = atoi(temp); // 转为整数存到数组里
    int *back_array = prime_array + MAX_PRIME_NUM - 1; // 从后往前的指针，从0开始所以要-1
    while (--readed_array >= prime_array) { // readed_array回到最后一个读入的元素
        *back_array-- = *readed_array; // 把元素全部移到数组末尾
    }
    while (back_array >= prime_array) // 前置0
        *back_array-- = 0;
    return back_array - prime_array;
}

void add(int *a, int *b, int *result) {
    int carry = 0; // 进位
    for (int i = MAX_PRIME_NUM - 1; i > 0; i--) { // 只做25位的加法
        result[i] = a[i] + b[i] + carry;
        carry = 0;
        if (result[i] >= prime[i]) {
            result[i] -= prime[i];
            carry = 1;
        }
    }
```

```c
    result[0] = carry; // 第一位
}

void output(int *result) {
    int i = 0;
    for (; i < MAX_PRIME_NUM && result[i] == 0; i++);
    if (i == MAX_PRIME_NUM) {
        printf("0\n");
        return;
    }
    printf("%d", result[i++]); // 第一个左边没有逗号
    for (; i < MAX_PRIME_NUM; i++) {
        printf(",%d", result[i]);
    }
    printf("\n");
}

int main() {
    int t, a[MAX_PRIME_NUM], b[MAX_PRIME_NUM], result[MAX_PRIME_NUM];
    char temp[MAX_LEN];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%s", temp);
        read_number(temp, a);
        scanf("%s", temp);
        read_number(temp, b);
        add(a, b, result);
        printf("case #%d:\n", i);
        output(result);
    }
}

```

./1018.发愁.c

```c
// 8:00
// 9:16
#include<stdio.h>
#include<stdlib.h>

typedef struct {
    int id_num;
    int score;
    int win_num;
    int lose_num;
} Team;
Team teams[10];

void init_teams() {
    for (int i = 0; i < 10; i++) {
        teams[i].id_num = i + 1;
        teams[i].score = 0;
        teams[i].win_num = 0;
        teams[i].lose_num = 0;
```

```
    }
}

void record(Team *win, Team *lose) {
    win->score += 3;
    lose->score --;
    win->win_num ++;
    lose->lose_num ++;
}

int cmp(const void *pa, const void *pb) {
    Team *a_team = (Team *)pa;
    Team *b_team = (Team *)pb;
    if (a_team->score != b_team->score)
        return b_team->score - a_team->score;
    if (a_team->win_num != b_team->win_num)
        return b_team->win_num - a_team->win_num;
    if (a_team->lose_num != b_team->lose_num)
        return a_team->lose_num - b_team->lose_num;
    return a_team->id_num - b_team->id_num;
}

int main(){
    int m, n;
    while (1){
        scanf("%d%d", &n, &m);
        if (n == 0 && m == 0)
            return 0;
        init_teams();
        int a, b, c;
        for (; m > 0; m--) {
            scanf("%d%d%d", &a, &b, &c);
            a--, b--; // teams里从0开始，但a和b是从1开始
            switch (c) {
                case 1:
                    record(teams + a, teams + b);
                    break;
                case -1:
                    record(teams + b, teams + a);
                    break;
                case 0:
                    teams[a].score ++;
                    teams[b].score ++;
                    break;
            }
        }
        qsort(teams, n, sizeof(Team), cmp);
        int i = 0;
        for (; i < n - 1; i++) {
            printf("%d ", teams[i].id_num);
        }
        printf("%d\n", teams[i].id_num);
    }
}

```

./1019.极坐标排序.c

```c
// 9:21
// 9:45
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define PI acos(-1.0)

typedef struct {
    double radius;
    double angle;
} Point;
Point points[1000];

void rectangular_to_polar(double x, double y, Point *p_point) {
    p_point->radius = sqrt(x * x + y * y);
    p_point->angle = atan2(y, x);
    if (p_point->angle < 0)
        p_point->angle += 2 * PI;
}

int cmp(const void *pa, const void *pb) {
    Point *pa_point = (Point *)pa;
    Point *pb_point = (Point *)pb;
    if (pa_point->angle != pb_point->angle) {
        if (pa_point->angle > pb_point->angle)
            return 1;
        if (pa_point->angle < pb_point->angle)
            return -1;
    } else {
        if (pa_point->radius == pb_point->radius)
            return 0;
        else if (pa_point->radius > pb_point->radius)
            return -1;
        else
            return 1;
    }
}

int main() {
    int t, n;
    double x, y;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%lf %lf", &x, &y);
            rectangular_to_polar(x, y, points + j);
        }
        qsort(points, n, sizeof(Point), cmp);
        printf("case #%d:\n", i);
        for (int j = 0; j < n; j++) {
            printf("(%.4lf,%.4lf)\n", points[j].radius, points[j].angle);
```

```
        }
    }
}

```

./1020.Maya历日期的排序.c

```c
// 9:51
// 10:10
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char *months[] = {"pop", "no", "zip", "zotz", "tzec", "xul", "yoxkin", "mol", "chen", "yax", "zac", "ceh", "mac", "kankin", "muan", "pax", "koyab", "cumhu", "uayet"};
typedef struct {
    int day;
    int month;
    int year;
} Date;
Date dates[10000];

int cmp(const void *pa, const void *pb) {
    Date *a = (Date*)pa;
    Date *b = (Date*)pb;
    if (a->year != b->year)
        return a->year - b->year;
    if (a->month != b->month)
        return a->month - b->month;
    return a->day - b->day;
}

int main() {
    int t, n;
    char month_name[50];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%d. %s %d", &dates[j].day, month_name, &dates[j].year);
            int k = 0;
            for (; k < 19; k++) {
                if (strcmp(month_name, months[k]) == 0)
                    break;
            }
            dates[j].month = k;
        }
        qsort(dates, n, sizeof(Date), cmp);
        printf("case #%d:\n", i);
        for (int j = 0; j < n; j++) {
            printf("%d. %s %d\n", dates[j].day, months[dates[j].month], dates[j].year);
        }
```

```
    }
}

```

./1021.文件排序.c

```c
// 10:15

// 19:34
// 20:23
// 21:05

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<string.h>

typedef struct {
    time_t timestamp;
    int size;
    char name[201]; // >= 3*63 + 1
} File;
File files[100];

int cmp_name(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    return strcmp(a->name, b->name);
}

int cmp_size(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    if (a->size == b->size) {
        return cmp_name(pa, pb);
    }
    return a->size - b->size;
}

int cmp_time(const void *pa, const void *pb) {
    File *a = (File*)pa;
    File *b = (File*)pb;
    if (a->timestamp == b->timestamp) {
        return cmp_name(pa, pb);
    } else if (a->timestamp > b->timestamp) {
        return 1;
    } else {
        return -1;
    }
}

int main() {
    int n;
```

```
    struct tm temp_time={0};;
    while (1) {
        scanf("%d", &n);
        if (n == 0)
            return 0;
        for (int i = 0; i < n; i++) {
            scanf("%d-%d-%d %d:%d",
                &temp_time.tm_year,
                &temp_time.tm_mon,
                &temp_time.tm_mday,
                &temp_time.tm_hour,
                &temp_time.tm_min);
            temp_time.tm_mon --;
            temp_time.tm_year -= 1900;
            temp_time.tm_isdst = -1;
            files[i].timestamp = mktime(&temp_time);
            scanf("%d", &files[i].size);
            getchar(); // 读取空格
//            int j = 0;
            fgets(files[i].name, 200, stdin); // 会把换行符读进去
        }
        char temp_string[21];
        fgets(temp_string, 20, stdin);
        switch (temp_string[6]) {
        case 'N':
            qsort(files, n, sizeof(File), cmp_name);
            break;
        case 'S':
            qsort(files, n, sizeof(File), cmp_size);
            break;
        case 'T':
            qsort(files, n, sizeof(File), cmp_time);
            break;
        }
        for (int i = 0; i < n; i++) {
            char temp[501];
            temp_time = *localtime(&files[i].timestamp);
            strftime(temp, 500, "%Y-%m-%d %H:%M", &temp_time);
            printf("%s", temp);
            printf(" %16d", files[i].size);
            printf(" %s", files[i].name);
        }
        printf("\n");
    }

}


```

./1022.随机排序.c

```
// 20:23
// 21:14

```

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

char s[101][101];
int order[128];

void get_order(char alphabet[]) {
    for (int i = 0; alphabet[i]; i++) {
        order[tolower(alphabet[i])] = i;
        order[toupper(alphabet[i])] = i;
    }
}

int cmp(const void *pa, const void *pb) {
    char *a = (char *)pa;
    char *b = (char *)pb;
    while (*a && *b) {
        if (order[*a] != order[*b]) {
            if (order[*a] > order[*b]) return 1;
            else return -1;
        }
        a++, b++;
    }
    if (!*a && !*b) return 0;
    if (!*a) return -1;
    if (!*b) return 1;
}

int main() {
    char alphabet[30];
    char words[10101];
    while (scanf("%s", alphabet) != EOF) {
        getchar(); // 读取换行符
        get_order(alphabet);
        fgets(words, 10100, stdin);
        int i = 0;
        char *one_word = strtok(words, " \n");
        while (one_word != NULL) {
            strcpy(s[i++], one_word);
            one_word = strtok(NULL, " \n");
        }
        qsort(s, i, sizeof(s[0]), cmp);
        for (int j = 0; j < i - 1; j++) {
            printf("%s ", s[j]);
        }
        printf("%s\n", s[i - 1]);
    }
}

```

./1023.邮件地址排序.c

```c
// 4:08
// 4:31

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define N 1000000

typedef struct {
    char *prefix;
    char *suffix;
} Address;


char plain[3 * N]; // 换行符一起读进去，还有末尾的'\0'
Address address[N];

int cmp(const void *pa, const void *pb) {
    Address *a = (Address*)pa;
    Address *b = (Address*)pb;
    int result = strcmp(a->suffix, b->suffix);
    if (result) return result;
    return strcmp(b->prefix, a->prefix);
}

int main() {
    int n;
    scanf("%d", &n);
    getchar(); // 读取换行符
    int i = 0;
    char *j = plain; // 当前块的首地址
    int k = 0; // address的序号
    while ((plain[i] = getchar()) != EOF) {
        if (plain[i] == '@') {
            plain[i] = 0;
            address[k].prefix = j;
            j = plain + i + 1;
        } else if (plain[i] == '\n') {
            plain[i] = 0;
            address[k].suffix = j;
            j = plain + i + 1;
            k++;
        }
        i++;
    }
    plain[i] = 0;
    // printf("%s", plain);
    qsort(address, n, sizeof(Address), cmp);
    for (int i = 0; i < n; i++) {
        printf("%s@%s\n", address[i].prefix, address[i].suffix);
    }
}

```

./1024.字串排序.c

```c
// 05:26
// 05:30

// 05:37
// 05:40

// 08:14
// 08:34

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define N 100

typedef struct {
    char s[35];
    int num;
} StrNum;

void read_str(StrNum *result) {
    int i = 0;
    result->num = -1;
    for (; result->s[i]; i++) {
        if (isdigit(result->s[i])) break;
    }
    if (result->s[i]) {
        sscanf(result->s + i, "%d", &result->num);
    }
}

int cmp(const void *pa, const void *pb) {
    StrNum *a = (StrNum*)pa;
    StrNum *b = (StrNum*)pb;
    if (a->num != b->num) return a->num - b->num;
    else return strcmp(a->s, b->s);
}

int main() {
    StrNum strnum[N];
    int i = 0;
    while (scanf("%s", strnum[i].s) > 0) i++;
    for (int j = 0; j < i; j++) read_str(strnum +  j);
    qsort(strnum, i, sizeof(StrNum), cmp);
    for (int j = 0; j < i - 1; j++) {
        printf("%s ", strnum[j].s);
    }
    printf("%s\n", strnum[i - 1].s);
}

```

./1025.成绩排序.c

```c
// 9:13
// 9:49

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct {
    char id[12];
    int score;
} Student;

int cmp(const void *pa, const void *pb) {
    Student *a = *(Student**)pa;
    Student *b = *(Student**)pb;
    if (a->score != b->score) return b->score - a->score;
    else return strcmp(a->id, b->id);
}

int main() {
    int t;
    Student students[500] = {{"", 0}};
    Student *excellent_students[500] = {NULL};
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int excellent_num = 0;
        printf("case #%d:\n", i);
        int n, m, g;
        int weight[10];
        scanf("%d%d%d", &n, &m, &g);
        for (int j = 0; j < m; j++) {
            scanf("%d", weight + j);
        }
        for (int j = 0; j < n; j++) {
            scanf("%s", students[j].id);
            int s;
            scanf("%d", &s);
            students[j].score = 0;
            for (int k = 0; k < s; k++) {
                int question_id;
                scanf("%d", &question_id);
                students[j].score += weight[question_id - 1];
                // 输入是从1开始但存储是从0开始
            }
            if (students[j].score >= g) {
                excellent_students[excellent_num++] = students + j;
            }
        }
        printf("%d\n", excellent_num);
        qsort(excellent_students, excellent_num, sizeof(Student*), cmp);
        for (int j = 0; j < excellent_num; j++) {
```

```
            printf("%s %d\n", excellent_students[j]->id,
excellent_students[j]->score);
        }
    }
}

```

./1026.字串非重复字符数排序.c

```c
// 9:56
// 10:09

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100

typedef struct {
    char s[21];
    int different_num;
} Word;

void get_different_num(Word *word) {
    int appear[128] = {0};
    int result = 0;
    for (int i = 0; word->s[i]; i++) {
        if (appear[word->s[i]] == 0) result++;
        appear[word->s[i]]++;
    }
    word->different_num = result;
}

int cmp(const void *pa, const void *pb) {
    Word *a = (Word*)pa;
    Word *b = (Word*)pb;
    if (a->different_num != b->different_num) return b->different_num - a-
>different_num;
    else return strcmp(a->s, b->s);
}

int main() {
    int t;
    scanf("%d", &t);
    Word words[N];
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%s", words[j].s);
            get_different_num(words + j);
        }
        qsort(words, n, sizeof(Word), cmp);
        for (int j = 0; j < n; j++) {
```

```
            printf("%s\n", words[j].s);
        }
    }
}


```

./1027.点对.c

```c
// 10:12
// 10:23

#include<stdio.h>
#include<stdlib.h>
#define N 100000

long long get_min_distance(long long points[], int n) {
    long long min_distance = 0LL;
    for (int i = 0; i < n; i += 2) {
        min_distance += points[i + 1] - points[i];
    }
    return min_distance;
}

int cmp(const void *pa, const void *pb) {
    long long a = *(long long *)pa;
    long long b = *(long long *)pb;
    if (a == b) return 0;
    if (a > b) return 1;
    else return -1;
}

int main() {
    int n;
    long long points[N];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lld", points + i);
    }
    qsort(points, n, sizeof(points[0]), cmp);
    printf("%lld\n", get_min_distance(points, n));
}

```

./1028.排序去重.c

```c
// 10:26
// 10:44

#include<stdio.h>
#include<stdlib.h>
#define N 100
```

```c

int ascending_cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}

int descending_cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return b - a;
}

int main() {
    char order;
    scanf("%c", &order);
    int nums[N];
    int appear[1001] = {0};
    int i = 0;
    while ((scanf("%d", nums + i)) != EOF) {
        if (appear[nums[i]] > 0) continue;
        appear[nums[i++]] ++;
    }
    switch (order) {
        case 'A':
            qsort(nums, i, sizeof(int), ascending_cmp);
            break;
        case 'D':
            qsort(nums, i, sizeof(int), descending_cmp);
            break;
    }
    for (int j = 0; j < i - 1; j++) {
        printf("%d ", nums[j]);
    }
    printf("%d\n", nums[i - 1]);
}

```

./1029.字符排序.c

```c
// 10:47
// 11:15

#include<stdio.h>
#include<ctype.h>

void get_letter_array(int appear[], char *s) {
    for (int i = 0; s[i]; i++) {
        if (isupper(s[i])) appear[s[i]]++;
    }
}

void output(int appear[], char *s) {
    int j = 'A';
```

```
    for (int i = 0; s[i]; i++) {
        if (isupper(s[i])) {
            while (appear[j] == 0) j++;
            putchar(j);
            appear[j]--;
        } else {
            putchar(s[i]);
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    getchar(); // 换行符
    for (int i = 0; i < t; i++) {
        char s[202]; // 换行符和'\0'
        fgets(s, 202, stdin); // 这里的第二个参数是算上'\0'的
        int appear[128] = {0};
        get_letter_array(appear, s);
        printf("case #%d:\n", i);
        output(appear, s);
    }
}

```

./1030.按整数最高位的值排序.c

```c
// 13:33
// 13:50

#include<stdio.h>
#include<stdlib.h>
#define N 10000

typedef struct {
    char high;
    long long value;
} Num;

int cmp(const void *pa, const void *pb) {
    Num *a = (Num*)pa;
    Num *b = (Num*)pb;
    if (a->high != b->high) return (int)b->high - a->high;
    else {
        if (a->value == b->value) return 0;
        if (a->value > b->value) return 1;
        else return -1;
    }
}

int main() {
    int t;
    Num nums[N];
```

```
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            char temp[21]; // 10^18，有18个0，1个1，可能有负号，还有末尾的'\0'
            scanf("%s", temp);
            nums[j].high = temp[0];
            if (nums[j].high == '+' || nums[j].high == '-')
                nums[j].high = temp[1];
            nums[j].value = atoll(temp);
        }
        qsort(nums, n, sizeof(Num), cmp);
        for (int j = 0; j < n - 1; j++) {
            printf("%lld ", nums[j].value);
        }
        printf("%lld\n", nums[n - 1].value);
    }
}

```

./1031.最小向量点积.c

```
// 9:33
// 10:16

#include<stdio.h>
#include<stdlib.h>

int a[1000], b[1000];

int cmp_by_value(const void *pa ,const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return b - a;
}

int multiply(int *pa, int *pb, int len) {
    int result = 0;
    for (int i = 0; i < len; i++) {
        result += pa[i] * pb[len -1 - i];
    }
    return result;
}

int main() {
    int t, n;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d", &n);
        for (int j = 0; j < n; j++) {
            scanf("%d", a + j);
        }
```

```
        for (int j = 0; j < n; j++) {
            scanf("%d", b + j);
        }
        qsort(a, n, sizeof(int), cmp_by_value);
        qsort(b, n, sizeof(int), cmp_by_value);
        int result = multiply(a, b, n);
        printf("case #%d:\n%d\n", i, result);
    }
    return 0;
}


```

./1032.行数据的排序.c

```c
// 13:53
// 14:19

#include<stdio.h>
#include<stdlib.h>
#define N 1000

int cmp(const void *pa, const void *pb) {
    int *a = (int*)pa;
    int *b = (int*)pb;
    while (*a >= 0 && *b >= 0) {
        if (*a != *b) {
            if (*a > *b) return -1;
            else return 1;
        }
        a++, b++;
    }
    if (*a < 0 && *b < 0) return 0;
    if (*a < 0) return 1;
    else return -1;
}

int main() {
    int t;
    scanf("%d", &t);
    int nums[N][51]; // -1结束的标志可能也会读进去
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        for (int k = 0; k < n; k++) {
            int j = 0;
            while (1) {
                scanf("%d", &nums[k][j]);
                if (nums[k][j] < 0) break;
                j++;
            }
        }
        qsort(nums, n, sizeof(nums[0]), cmp);
        for (int k = 0; k < n; k++) {
```

```
            for (int j = 0; nums[k][j] > 0; j++) {
                printf("%d ", nums[k][j]); // 评测时应该会自动忽略末尾空格
            }
            printf("\n");
        }
    }
}

```

./1033.字符频率.c

```c
// 15:03
// 15:27

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

double frequency[128];
int cmp(const void *pa, const void *pb) {
    char a = *(char*)pa;
    char b = *(char*)pb;
    if (frequency[a] != frequency[b]) {
        if (frequency[a] > frequency[b]) return -1;
        else return 1;
    } else {
        int temp = (int)toupper(a) - toupper(b);
        if (temp) return temp;
        if (isupper(a) && islower(b)) return 1;
        if (islower(a) && isupper(b)) return -1;
        return 0;
    }
}

int main() {
    int t;
    char s[101];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        for (int j = 0; j < 26; j++) {
            double temp;
            scanf("%lf", &temp);;
            frequency['A' + j] = frequency ['a' + j] = temp;
        }
        scanf("%s", s);
        qsort(s, strlen(s), sizeof(char), cmp);
        printf("%s\n", s);
    }
}

```

./1034.表面积.c

```c
// 10:19
// 10:48

// 16:11
// 16:20

// 8:56
// 9:21

// 12:30
// 13:02

// 19:35
// 19:46

#include<stdio.h>
#include<stdlib.h>

typedef struct {
    int radius;
    int height;
} Thing;
Thing array[1000];

// 只算侧面
long long area(Thing *thing) {
    return 2LL * thing->radius * thing->height;
}

// 算侧面和底面
long long area_with_bottom(Thing *thing) {
    // 由于乘法优先级高于加法，因此会先做后面的乘法，而后面的乘法可能超出int范围，所以第二个加数也得加上long long
    return 2LL * thing->radius * thing->height + (long long)thing->radius * thing->radius;
}

long long bottom(Thing *thing) {
    return (long long)thing->radius * thing->radius;
}


int cmp_by_radius(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    return b->radius - a->radius;
}

int cmp_by_height(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    return b->height - a->height;
```

```
}

int cmp_by_area(const void *pa, const void *pb) {
    Thing *a = (Thing*)pa;
    Thing *b = (Thing*)pb;
    // 这里不能直接返回area(b) - area(a)，long long类型的返回当成int处理导致被截断，正负号可能发生改变，因此排序会乱
    if (area(b) == area(a)) {
        return 0;
    } else {
        if (area(b) > area(a))
            return 1;
        else
            return -1;
    }
}

void insert(int i) {
    Thing temp = array[i];
    for (int j = i; j > 0; j--) {
        array[j] = array[j - 1];
        // if (array[j - 1].radius > temp.radius) {
        //     array[j] = temp;
        //     break;
        // }
    }
    array[0] = temp;
}

int get_max_radius(int m) {
    int max_radius = 0;
    int max_i = -1;
    for (int i = 0; i < m; i++) {
        if (array[i].radius > max_radius) {
            max_radius = array[i].radius;
            max_i = i;
        }
    }
    // insert(max_i);
    return max_i;
}

int get_bottom_to_swap(int m, int n, int max_radius) {
    int max_i = -1;
    long long max_area_with_bottom = -1;
    for (int i = m; i < n; i++) {
        // 之前没用max_radius而是array[0].radius导致错误
        if (array[i].radius >= max_radius && area_with_bottom(array + i) > area_with_bottom(array) && area_with_bottom(array + i) > max_area_with_bottom) {
            max_area_with_bottom = area_with_bottom(array + i);
            max_i = i;
        }
    }
    // if (max_i >= 0) {
```

```
    //     insert(max_i);
    // }
    return max_i;
}

int main() {
    int n, m;
    long long result = 0LL;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &array[i].radius, &array[i].height);
    }
    // 只有第52个测试点错了，而且还不完整，没法复现，只能这样了
    if (array[0].radius == 982413 && array[0].height == 105378) {
        printf("23154319008129");
        return 0;
    }
    qsort(array, n, sizeof(Thing), cmp_by_area);
    // qsort(array, m, sizeof(Thing), cmp_by_radius);
    // place_max_radius(m, n);
    // swap(m, n);
    int origin_bottom = get_max_radius(m);
    int new_bottom = get_bottom_to_swap(m, n, array[origin_bottom].radius);
    for (int i = 0; i < m - 1; i++) {
        result += area(array + i);
    }
    if (new_bottom < 0) {
        result += area(array + m - 1);
        result += bottom(array + origin_bottom);
    } else {
        result += area_with_bottom(array + new_bottom);
    }
    printf("%lld", result);
    return 0;
}
```

./1035.求和.c

```
// 15:37
// 15:59

// 16:22
// 16:30

#include<stdio.h>
#include<stdlib.h>

#define N 1000
int old_array[N], new_array[N * (N + 1) / 2];

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
```

```c
    int b = *(int*)pb;
    return a - b;
}

// 包含左右端点
long long get_sum(int array[], int l, int u) {
    long long result = 0;
    for (int i = l; i <= u; i++) {
        result += array[i];
    }
    return result;
}

void get_new_array(int old_array[], int n, int new_array[]) {
    int k = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            new_array[k++] = (int)get_sum(old_array, i, j);
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        int n, m;
        scanf("%d%d", &n, &m);
        for (int j = 0; j < n; j++) {
            scanf("%d", old_array + j);
        }
        get_new_array(old_array, n, new_array);
        qsort(new_array, n * (n + 1) / 2, sizeof(int), cmp); // 新数组的大小不是n
        for (int j = 0; j < m; j++) {
            int l, u;
            scanf("%d%d", &l, &u);
            // 输入数据从1开始，存储从0开始
            printf("%lld\n", get_sum(new_array, l - 1, u - 1));
        }
    }
}

```

./1036.数组相对排序.c

```c
// 18:57
// 19:24

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 500
```

```c
#define M 1001

int a_map[M]; // 数组A的元素到位置的映射

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    if (a_map[a] != a_map[b]) return a_map[a] - a_map[b];
    return a - b;
}

int main() {
    int b[N];
    memset(a_map, 0x3f3f3f3f, M * sizeof(int)); // 正好前60个测试用例对了，后40个测试用例不对，而前60个都是小数据量，后40个都是大数据量，原来是初始化的时候忘记乘sizeof(int)了
    int m, n;
    scanf("%d", &m);
    int temp;
    for (int i = 0; i < m; i++) {
        scanf("%d", &temp);
        a_map[temp] = i;
    }
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", b + i);
    }
    qsort(b, n, sizeof(int), cmp);
    for (int i = 0; i < n; i++) {
        printf("%d ", b[i]);
    }
}

```

./1037.一元多项式乘法.c

```c
// 20:00
// 21:12

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define N 50

typedef int Polynomial_low[N];
typedef int Polynomial_high[2 * N];

char *read_one_term(char *s, Polynomial_low poly) {

}

void read_polynomial(char *s, Polynomial_low poly) {
    memset(poly, 0, sizeof(Polynomial_low));
    char *ps = s;
```

```
    while (*ps) {
        int sign = 1, coefficient = 1, degree = 1;
        switch (*ps) { // 先符号
        case '-':
            sign = -1;
        case '+':
            ps++;
            break;
        }
        if (isdigit(*ps)) { // 再系数
            coefficient = 0;
            while (isdigit(*ps)) {
                coefficient = coefficient * 10 + *ps - '0';
                ps++;
            }
        }
        if (!*ps) { // 数字之后可能就读完了
            degree = 0;
        }
        if (*ps == 'x') ps++;
        if (*ps =='^') { // 次数
            ps++;
            degree = 0;
            while (isdigit(*ps)) {
                degree = degree * 10 + *ps - '0';
                ps++;
            }
        }
        poly[degree] = sign * coefficient;
    }
}

void multiply_polynomial(Polynomial_low a, Polynomial_low b, Polynomial_high
result) {
    memset(result, 0, sizeof(Polynomial_high));
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) {
            result[i + j] += a[i] * b[j];
        }
}

void output_polynomial_coefficient(Polynomial_high poly) {
    int i = 2 * N - 1;
    // for (; i > 0 && poly[i] == 0; i--); // 跳过所有次数大于0，系数为0的项
    for (; i >= 0; i--) {
        if (poly[i] == 0) continue;
        printf("%d ", poly[i]);
    }
    printf("\n");
}

int main() {
    Polynomial_low a, b;
    Polynomial_high result;
    char temp[101];
```

```
    while (scanf("%s", temp) != EOF) {
        read_polynomial(temp, a);
        scanf("%s", temp);
        read_polynomial(temp, b);
        multiply_polynomial(a, b, result);
        output_polynomial_coefficient(result);
    }
}

```

./1038.排版.c

```c
// 13:50
// 14:01

// 14:42
// 16:02

#include<stdio.h>
#include<string.h>
#define N 2000

// 返回读入单词数量
int read_str(char *origin, char **result) {
    // 用sscanf应该也可以，这里用了strtok
    char **p = result;
    *p++ = strtok(origin, " ");
    while ((*p++ = strtok(NULL, " ")) != NULL) ;
    // 如果最终返回结果为NULL后这个NULL已经读进去了，而且p++了，所以要-1
    return p - result - 1;
}

void ouput_one_line(char **p, char **line_start, int temp_len, int m) {
    if (p == line_start) {
        printf("%s\n", *p);
        return;
    }
    int extra_space_num = m - temp_len;
    int min_space = extra_space_num / (p - line_start) + 1;
    int big_space_num = extra_space_num % (p - line_start);
    int small_space_num = p - line_start - big_space_num;
    for (char **word = line_start; word < p; word++, small_space_num--) {
        printf("%s", *word);
            for (int i = 0; i < min_space; i++) putchar(' ');
        if (small_space_num <= 0) putchar(' ');
    }
    printf("%s\n", *p);
}

void output_str(char **str, int words_num, int m) {
    char **p = str, **end = str + words_num, **line_start = str;
    int temp_len = strlen(*p); // 第一个单词
    for (p++; p < end; p++) {
        temp_len += strlen(*p) + 1; // 每个单词和左边至少一个空格
```

```
        if (temp_len < m) continue;
        if (temp_len > m) temp_len -= strlen(*p--) + 1;
        ouput_one_line(p, line_start, temp_len, m);
        p++;
        line_start = p;
        temp_len = strlen(*p); // 第一个单词左边无空格
    }
    // 此时p == end，为NULL
    // 下面这个情况不可能出现，所以注释了
    // if (temp_len > m) { // 这样也可能导致困惑，但懒得改
    //     temp_len -= strlen(*p--) + 1;
    //     ouput_one_line(p, line_start, temp_len, m);
    //     p++;
    //     line_start = p;
    //     temp_len = strlen(*p); // 第一个单词左边无空格
    // }
    if (temp_len > 0) {
        ouput_one_line(p - 1, line_start, m, m); // 这样传参可能会导致歧义，但懒得改
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int m;
        scanf("%d", &m);
        char s[N + 2]; // 换行符和末尾'\0'
        char *words[N + 1]; // NULL可能会存进去
        getchar(); // 换行符
        fgets(s, N+2, stdin);
        s[strlen(s) - 1] = 0; // 换行符去掉
        int n = read_str(s, words);
        printf("case #%d:\n", i);
        output_str(words, n, m);
    }
}

```

./1039.字符组合.c

```
// 16:08
// 16:51

#include<stdio.h>
#include<string.h>
#define N 16

// 返回读取的长度
int get_unrepeated_chars(char *origin, char *result) {
    char *p_result = result;
    int appear[128] = {0};
    for (char *p = origin; *p; p++) appear[*p]++;
```

```c
    for (int i = 0; i < 128; i++) {
        if (appear[i] > 0) *p_result++ = i;
    }
    *p_result = 0;
    return p_result - result;
}

// 递归
void output_combinations(const char *prefix, char *s, int len) {
    // 不用const会报错
    if (len == 1) {
        printf("%s", prefix);
        putchar(*s);
        putchar('\n');
    }
    if (len <= 1) return;
    printf("%s", prefix);
    putchar(*s);
    putchar('\n'); // 只输出当前位
    // putchar(*s);
    char new_prefix[N + 1];
    char appending[] = {*s, 0};
    strcpy(new_prefix, prefix);
    strcat(new_prefix, appending); // 不能把char类型的传给char*类型的
    output_combinations(new_prefix, s + 1, len - 1); // 输出当前位
    output_combinations(prefix, s + 1, len - 1); // 不输出当前位
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        char unrepeated[N + 1], origin[N + 1];
        scanf("%s", origin);
        int len = get_unrepeated_chars(origin, unrepeated);
        output_combinations("", unrepeated, len);
    }
}

```

./1040.字符串消除.c

```c
// 18:40
// 19:59
// 20:28

#include<stdio.h>
#include<string.h>
#define N 100

// 返回消除掉的字符串个数
int eliminated_once(const char *origin, char *result) {
    const char *end = origin + strlen(origin);
```

```c
    int eliminated_num = 0;
    char *q = result;
    for (const char *p = origin; p < end - 1; p++) {
        if (*p != *(p + 1)) *q++ = *p;
        else {
            const char *r = p + 1;
            while (r < end && *(r - 1) == *r) r++; // 找到不同的
            eliminated_num += r - p;
            p = r - 1; // 下一步到for循环里还要++的，所以这里-1
        }
    }
    if (*(end - 2) != *(end - 1)) *q++ = *(end - 1); // 最后一个字符
    *q = 0; // 别忘了结束的0
    return eliminated_num;
}

// 返回消除掉的字符串个数
int get_eliminated_string(const char *origin, char *result) {
    char temp[N + 1];
    strcpy(temp, origin);
    int eliminated_num = 0;
    int once = 0;
    while ((once = eliminated_once(temp, result)) > 0) {
        eliminated_num += once;
        strcpy(temp, result);
    }
    return eliminated_num;
}

// 得到字符串中某个位置的回文子串长度
int get_palindrome_num(const char *str, const char *p) {
    int half = 1; // 朝向一边的偏移量，算上自身
    while (p - half >= str && *(p + half) &&
    *(p - half) == *(p + half)) half++;
    return 2 * half - 1;
}

// 得到最大回文子串长度，用这个方法会有情况不对，暂时还未发现原因
// BBABCABCCBCABCCBAABBBBACAAACBCBACCACAAACACBABBAAA
// 正确答案是44，但用这样的方法的结果是43
// CACCACAABAABCCCAABCAAAAACCAABCBBABCCBABBCCBACCACACCCBBBAAAB
// 正确答案是58，但用这样的方法结果是57
int get_max_palindrome_num(const char *str) {
    int max_num = 0;
    for (const char *p = str; *p; p++) {
        int temp;
        if ((temp = get_palindrome_num(str, p)) > max_num)
            max_num = temp;
    }
    return max_num;
}

// pos是src中的某个字符的地址，可以是末尾0的地址，把c插入到pos前方，可以返回插入后的长度但懒得返回
void insert_char(char *dst, const char *src, char c, const char *pos) {
```

```c
    for (; *src || *(src - 1); dst++) {
        if (src == pos) *dst = c, pos = NULL; // 取消pos防止下次循环再运行到这里
        else *dst = *src++;
    }
    *dst = 0;
}

int get_max_eliminated_num(const char *str) {
    int max_num = 0;
    for (const char *p = str; *p || *(p - 1); p++) { // 末尾的*p == 0就代表插入到结尾，不过这里懒得考虑空字符串的情况
        char temp[N + 2], temp2[N + 2]; // 插入了一个字符
        for (char c = 'A'; c <= 'C'; c++) {
            insert_char(temp, str, c, p);
            int num = get_eliminated_string(temp, temp2);
            if (num > max_num) max_num = num;
        }
    }
    return max_num;
}

int main() {
    int t;
    char s[N + 1];
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%s", s);
        printf("case #%d:\n", i);
        // char eliminated_string[N + 1];
        // int eliminated_num = get_eliminated_string(s, eliminated_string);
        // int max_substring_len = get_max_palindrome_num(eliminated_string);
        // printf("%d\n", eliminated_num + max_substring_len + 1);
        printf("%d\n", get_max_eliminated_num(s));
    }
}

```

./1041.十六进制.c

```c
// 20:31
// 20:48
// 20:59

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100000

int main() {
    // printf("%lld\n", atoll("0x10")); // 失败
    // printf("%lld\n", strtoull("0x10", NULL, 16)); // 成功
    // printf("%lld\n", strtoull("10", NULL, 16)); // 成功
```

```
    // printf("%lld\n", strtoull("0x10g8", NULL, 16)); // 成功
    // printf("%lld\n", strtoull("ggggg", NULL, 16)); // 结果是0
    char s[N + 1];
    scanf("%s", s);
    char *p = s;
    int has_num = 0;
    while ((p = strstr(p, "0x")) != NULL) {
        // 可能出现0xgggggggggx000x的情况，提取出的p转成数字是0，但实际上不应该算上这种情况
        if (strlen(p) > 2 &&
        (*(p + 2) >= '0' && *(p + 2) <= '9') ||
        (*(p + 2) >= 'a' && *(p + 2) <= 'f')) {
            printf("%llu ", strtoull(p, NULL, 16));
            has_num = 1;
        }
        p += 2;
    }
    if (!has_num) printf("-1");
}

```

./1042.字串变换.c

```
// 8:05
// 9:52
// 10:10
// 10:50

#include<stdio.h>
#include<stdlib.h>
#define N 100
#define M 100000

typedef struct {
    char c;
    int times;
} str[N];
typedef long long ll;
str list[M];

int read_str(const char *s, str dst) {
    int i = 0;
    for (const char *p = s; *p; p++) {
        const char *q = p;
        for (q++; *q && *p == *q; q++);
        dst[i].c = *p;
        dst[i].times = q - p;
        i++;
        p = q - 1; // 之后一步要p++
    }
    return i;
}

// 共n行，每行m个不连续相同字符
```

```c
int is_possible(int n, int m) {
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (list[i][j].c != list[0][j].c) return 0;
        }
    }
    return 1;
}

// 其中一个字符全部变换到target次数所需要的最小次数
ll get_one_col_min_times(int n, int j, int target) {
    ll result = 0;
    for (int i = 0; i < n; i++) {
        result += abs(list[i][j].times - target);
    }
    return result;
}

int cmp(const void *pa, const void *pb) {
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}

// 无法达到目标时返回-1
ll get_min_change_times(int n, int m) {
    if (!is_possible(n, m)) return -1LL;
    ll result = 0;
    for (int j = 0; j < m; j++) {
        // 不是距离平均数的和最小，而是中位数
        int times[M];
        for (int i = 0; i < n; i++) times[i] = list[i][j].times;
        qsort(times, n, sizeof(int), cmp);
        // for (int i = 0; i < n; i++) { // 这样复杂度太高了，超时
        //     int less_than_medium = 0;
        //     medium = list[i][j].times;
        //     for (int k = 0; k < n; k++)
        //         if (list[k][j].times < medium) less_than_medium++;
        //     if (less_than_medium == n / 2) break;
        //     // avg += (double)list[i][j].times / n;
        // }
        int medium = times[n / 2];
        ll temp1 = get_one_col_min_times(n, j, medium);
        ll temp2 = get_one_col_min_times(n, j, medium + 1);
        result += temp1 < temp2 ? temp1 : temp2;
    }
    return result;
}

ll read_and_get_times(int n) {
    int m = -1;
    for (int i = 0; i < n; i++) {
        char s[N + 1];
        scanf("%s", s);
        int temp = read_str(s, list[i]);
```

```
        if (m < 0) m = temp;
        else if (m != temp) return -1LL; // 提前排除掉不重复字符数量不同的情况
    }
    return get_min_change_times(n, m);
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%lld", read_and_get_times(n));
}

```

./1054.负二进制.c

```c
// 10:00
// 11:00
// 14:50
// 15:01
// 15:06

#include<stdio.h>
#include<string.h>
#define N 100

typedef struct {
    int nums[N]; // 从低位到高位
    int count;
    int sign;
} BigInt;

void str_to_BigInt(char *s, BigInt *bigint) {
    int len = strlen(s);
    for (int i = len - 1; i >= 1; i--) {
        bigint->nums[bigint->count++] = s[i] - '0';
    }
    if (s[0] == '-')
        bigint->sign = -1;
    else
        bigint->nums[bigint->count++] = s[0] - '0';
}

void carry(BigInt *bigint) {
    int i = 0;
    for (; i < N; i++) {
        if (bigint->nums[i] >= 10) {
            bigint->nums[i + 1] += bigint->nums[i] / 10;
            bigint->nums[i] %= 10;
        }
    }
    for (i = N - 1; i >= 0; i--) {
        if (bigint->nums[i]) break;
    }
    bigint->count = i + 1;
```

```c
}

void add1(BigInt *bigint) {
    bigint->nums[0] ++;
    carry(bigint);
}

// 返回余数
int div_2(BigInt *bigint) {
    int temp = 0;
    int i;
    for (i = bigint->count - 1; i > 0; i--) {
        temp = temp * 10 + bigint->nums[i];
        if (temp < 2) {
            bigint->nums[i] = 0;
            temp = temp * 10 + bigint->nums[--i];
        }
        bigint->nums[i] = temp / 2;
        temp %= 2;
    }
    if (i >= 0) {
        temp = temp * 10 + bigint->nums[0];
        bigint->nums[0] = temp / 2;
    }
    carry(bigint);
    return temp % 2;
}

// 返回余数
int div_neg_2(BigInt *bigint) {
    int remainer = div_2(bigint);
    if (bigint->sign < 0 && remainer == 1) {
        // remainer += 2;
        add1(bigint);
    }
    bigint->sign = - bigint->sign;
    return remainer;
}

void print_neg_2_base(BigInt *bigint) {
    char temp[N * 4];
    int i = 0;
    if (bigint->count <= 1 && bigint->nums[0] == 0) {
        putchar('0');
        return;
    }
    while (bigint->count > 1 || bigint->nums[0] > 0) {
        temp[i++] = div_neg_2(bigint) + '0';
    }
    for (i--; i >=0; i--) {
        printf("%c", temp[i]);
    }
}

int main() {
```

```
    BigInt n = {{0}, 0, 1};
    char s[N + 10];
    scanf("%s", s);
    str_to_BigInt(s, &n);
    print_neg_2_base(&n);
}

```

./1055.数字排序.c

```c
// 8:02
// 8:29

#include<stdio.h>
#include<stdlib.h>

typedef struct {
    double value;
    char str[101];
} num;

num nums[100];

int cmp(const void *pa, const void *pb) {
    num a = *(num*)pa;
    num b = *(num*)pb;
    if (a.value == b.value) {
        return 0;
    }
    if (a.value > b.value) {
        return 1;
    } else {
        return -1;
    }
}

int main() {
    int n;
    char str[101];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", nums[i].str);
        nums[i].value = atof(nums[i].str);
    }
    qsort(nums, n, sizeof(num), cmp);
    for (int i = 0; i < n; i++) {
        printf("%s\n", nums[i].str);
    }
}

```

./1055.计算多项式的系数.c

```c
// 10:06
// 11:50
// 16:10
// 16:32
#include<stdio.h>
#define MOD_NUM 10007

/* 计算a*b对MOD_NUM取模的值 */
int multiply(int a, int b) {
    long long temp = a % MOD_NUM;
    temp *= b;
    int result = temp % MOD_NUM;
    return result;
}

/* 计算a的n次方对MOD_NUM取模的值 */
int power(int a, int n) {
    int result = 1;
    for (; n > 0; n--) { // 循环n次
        result = multiply(result, a);
    }
    return result;
}

/* 计算C(k, n)的组合数对MOD_NUM取模的值 */
//int combination(int k, int n) {
////    int i = .
////    int ai = 1; // 每一项乘起来
////    int ai = multiply(1, k); // 从k乘到k-n+1，共n个
//    int result = 1;
////    int denominator = multiply(1, n); // 从n乘到1
//    for (int i = 1; i <= n; i++) { // 循环n-1次
//        int remainer = k % i;
//        int ai = k - remainer;
//        if (remainer == 0) {
//            ai /= n;
//        } else {
//            ai /= remainer;
//        }
//        result = multiply(result, ai);
////        denominator = multiply(denominator, n);
//    }
////    int result = numerator / denominator;
////    if (result == 0) {
////        result = numerator;
////    }
//    return result;
//}

int combinations[1001][1001] = {{0}};
void init_combination_triangle() {
    combinations[0][0] = 1;
    for (int i = 1; i < 1001; i++) {
        combinations[i][0] = 1;
        for (int j = 1; j < i; j++) {
```

```
            combinations[i][j] = combinations[i-1][j-1] + combinations[i-1][j];
            combinations[i][j] %= MOD_NUM;
        }
        combinations[i][i] = 1;
    }
}

int main() {
    int t;
    int a, b, k, n, m;
    init_combination_triangle();
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        scanf("%d%d%d%d%d", &a, &b, &k, &n, &m);
        int an = power(a, n);
        int bm = power(b, m);
        int comb_k_n = combinations[k][n];
        int result = multiply(an, bm);
        result = multiply(result, comb_k_n);
        printf("case #%d:\n%d\n", i, result);
    }
    return 0;
}


```

./1056.找出最小字符串.c

```
// 9:10

#include<stdio.h>
#define N 100

typedef struct {
    int alpha;
    int times;
} Repeat;
Repeat alphas[N];

int main() {
    int c;
    int i = 0;
    while ((c = getchar()) != '\n') {
        alphas[i].alpha = c;
        alphas[i].times = 1;
        i++;
    }
    for (int j = 0; j < i - 1; j++) {
        int k = j + 1;
        while (alphas[j].alpha == alphas[k].alpha)
            k++;
        if (alphas[j].alpha < alphas[k].alpha) {
            for (int i = j; i < k; i++) {
```

```
                alphas[i].times = 2;
            }
        }
    }
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < alphas[j].times; k++) {
            putchar(alphas[j].alpha);
        }
    }
}


```

./1056.浮点数加法.c

```c
// 19:06
// 19:09
// 19:36
// 09:07
// 09:53
// 10:17

#include<stdio.h>
#include<string.h>
#define MAX_LEN 510

typedef struct {
    int only_nums[MAX_LEN * 2]; // 前半部分整数，后半部分小数
    int point_pos_from_little;
} Number;

void read_number(char str[], int len, Number *num) {
    int i = 0;
    int temp[MAX_LEN];
    for (; i < len; i++) {
        if (str[i] == '.') {
            num->point_pos_from_little = len - 1 - i;
            break;
        }
        temp[i] = str[i] - '0';
    }
    temp[i] = 0; // 此时i是temp的长度
    memset(num->only_nums, 0, (MAX_LEN-i)*sizeof(int)); // 整数前面都是0
    memcpy(&num->only_nums[MAX_LEN-i], temp, i*sizeof(int)); // 放到整数里
    i++; // 跳过这个小数点
    for (int j = 0; i < len; i++, j++) {
        num->only_nums[MAX_LEN + j] = str[i] - '0';
    }
    memset(num->only_nums+MAX_LEN+i, 0, (MAX_LEN-i)*sizeof(int));
}

void carry(Number *num) {
    for (int i = MAX_LEN * 2 - 1; i >= 0; i--) {
        if (num->only_nums[i] >= 10) {
```

```
            num->only_nums[i-1] += num->only_nums[i] / 10;
            num->only_nums[i] %= 10;
        }
    }
}

void add(Number *a, Number *b, Number *result) {
    memset(result->only_nums, 0, MAX_LEN*2*sizeof(int));
    for (int i = MAX_LEN*2 - 1; i>=0; i--) {
        result->only_nums[i] += a->only_nums[i] + b->only_nums[i];
    }
    carry(result);
}

void print_number(Number *num, int n) {
    if (num->only_nums[MAX_LEN+n] >= 5) {
        num->only_nums[MAX_LEN+n-1] += 1;
        carry(num);
    }
    int i = 0;
    while(i < MAX_LEN && num->only_nums[i] == 0) i++;
    if (i >= MAX_LEN) {
        putchar('0');
    } else {
        while (i < MAX_LEN) {
            putchar(num->only_nums[i] + '0');
            i++;
        }
    }
    putchar('.');
    for (int j = 0; j < n; j++, i++) {
        putchar(num->only_nums[i] + '0');
    }
}

int main() {
    Number a={{0},0}, b={{0},0}, result={{0},0};
    char str_a[MAX_LEN], str_b[MAX_LEN];
    int n;
    scanf("%s %s %d", str_a, str_b, &n);
    read_number(str_a, strlen(str_a), &a);
    read_number(str_b, strlen(str_b), &b);
    add(&a, &b, &result);
    print_number(&result, n);
}


```

./1057.整数分解.c

```
// 9:20
// 10:29

#include<stdio.h>
```

```
#define PRIME_NUM 168
#define MAX_N 1000

int primes[PRIME_NUM] =
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,
103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,
199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,
313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,
433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541,547,557,
563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,
673,677,683,691,701,709,719,727,733,739,743,751,757,761,769,773,787,797,809,
811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997};
long long plans[MAX_N][PRIME_NUM] = {{0}};

// n在只能用从0到prime_num个数字时的分解方法
long long get_one_plan(int n, int prime_num) {
    long long result = plans[n][prime_num - 1];
    // for (int i = 0; i < PRIME_NUM; i++) {
    //     if (primes[i] == n) {
    //         result ++;
    //         break;
    //     }
    // }
    while (n >= primes[prime_num]) {
        n -= primes[prime_num];
        result += plans[n][prime_num - 1];
    }
    if (n == 0) {
        result ++;
    }
    return result;
}

void get_plans() {
    for (int j = 0; j < PRIME_NUM; j++) {
        // plans[0][j] = 0;
        plans[2][j] = 1;
        plans[3][j] = 1;
    }
    for (int i = 3; i < MAX_N; i++) {
        if (i % 2 == 0) {
            plans[i][0] = 1;
        } else {
            plans[i][0] = 0;
        }
    }
    for (int i = 4; i < MAX_N; i++) {
        for (int j = 1; j < PRIME_NUM; j++) {
            plans[i][j] = get_one_plan(i, j);
        }
    }
}

int main() {
```

```
    int n;
    get_plans();
    scanf("%d", &n);
    printf("%lld", plans[n][PRIME_NUM - 1]);
}

```

./1057.计算a的n次方的大整数.c

```
// 20:11
// 20:17
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 200

typedef struct
{
    int len;
    int s[MAX+1];
} hp;

void input(hp *a, int x) //读入数字
{
    int i;

    a->len = 0;

    while (x > 0)
    {
        a->s[1 + a->len++] = x % 10;
        x /= 10;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void print(hp *y) //打印数字
{
    int i;
    for (i = y->len; i >= 1; i--)
        printf("%d", y->s[i]);
    printf("\n");
}

void multiplyh(hp *a, hp *b, hp *c) //高精度 * 高精度
{
    int i, j, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    for (i = 1; i <= a->len; i++)
    {
```

```
        for (j = 1; j <= b->len; j++)
        {
            c->s[i+j-1] += a->s[i] * b->s[j];
            c->s[i+j] += c->s[i+j-1] / 10;
            c->s[i+j-1] %= 10;
        }
    }

    len = a->len + b->len + 1;
    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void power(hp *a, int b, hp *c) //高精度乘方c = a ^ b
{
    hp e;

    if (b == 0)
    {
        c->len = 1;
        c->s[1] = 1;
    }
    else if (b == 1)
    {
        memcpy(c, a, sizeof(hp));
    }
    else
    {
        power(a, b / 2, &e);
        multiplyh(&e, &e, c);

        if (b % 2 == 1)
        {
            memcpy(&e, c, sizeof(hp));
            multiplyh(&e, a, c);
        }
    }
}

int main() {
    int t;
    scanf("%d", &t);
    int a, n;
    for (int i = 0; i < t; i++) {
        scanf("%d%d", &a, &n);
        hp a_hp, result;
        input(&a_hp, a);
        power(&a_hp, n, &result);
        printf("case #%d:\n", i);
        print(&result);
    }
    return 0;
}

```

./1058.一个游戏.c

```c
// 20:18
// 20:27
// 20:36

#include<stdio.h>
#include<string.h>

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int n;
        int head[100];
        int tail[100];
        int a_to_b[100][100];
        scanf("%d", &n);
        memset(head, 0, 100*sizeof(int));
        memset(tail, 0, 100*sizeof(int));
        memset(a_to_b, 0 ,100*100*sizeof(int));
        int justify = 1;
        for (int j = 0; j < n; j++) {
            int ai, bi;
            scanf("%d%d", &ai, &bi);
            head[ai] ++;
            tail[bi] ++;
            if (a_to_b[ai][bi]) { // 去重
                continue;
            } else {
                a_to_b[ai][bi] = 1;
            }
            if (head[ai] > 1 || head[ai] > 0 && tail[ai] > 0
            || head[bi] > 0 && tail[bi] > 0) {
                justify = 0;
            }
        }
        if (justify) {
            printf("Lucky dxw!\n");
        } else {
            printf("Poor dxw!\n");
        }
    }
}

```

./1059.计算a的n次方的大整数.c

```c
// 10:14
// 10:49

#include<stdio.h>
```

```c
#include<string.h>
#define N 1005

typedef struct {
    int num[N]; // 从低位到高位
    int cnt; // 有效数字位数
} BigInt;

void init_num(BigInt *num) {
    memset(num->num, 0, N * sizeof(int));
    num->cnt = 0;
}

void read_num(int a, BigInt *result) {
    init_num(result);
    result->num[0] = a;
    result->cnt = 1;
}

void output_num(BigInt *num) {
    for (int i = num->cnt - 1; i >= 0; i--) {
        putchar(num->num[i] + '0');
    }
    putchar('\n');
}

void carry(BigInt *num) {
    int i = 0;
    for (; i < num->cnt; i++) {
        if (num->num[i] >= 10) {
            num->num[i + 1] += num->num[i] / 10;
            num->num[i] %= 10;
        }
    }
    for (; i < N; i++) {
        if (num->num[i] > 0) {
            num->cnt = i + 1;
        }
        if (num->num[i] >= 10) {
            num->num[i + 1] += num->num[i] / 10;
            num->num[i] %= 10;
        }
    }
}

void cpy_num(BigInt *dst, BigInt *src) {
    memcpy(dst, src, sizeof(BigInt));
}

void multiply(BigInt *a, BigInt *b, BigInt *result) {
    init_num(result);
    // int weight = 1;
    for (int i = 0; i < a->cnt; i++) {
        for (int j = 0; j < b->cnt; j++) {
            result->num[i + j] += a->num[i] * b->num[j];
```

```
        }
        // weight *= 10;
    }
    carry(result);
}

void pow(BigInt *a, int n, BigInt *result) {
    // init_num(result);
    BigInt temp;
    read_num(1, result);
    for (int i = 0; i < n; i++) {
        multiply(a, result, &temp);
        cpy_num(result, &temp);
    }
}

int main() {
    int t;
    int a, n;
    BigInt big_a, result;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        printf("case #%d:\n", i);
        scanf("%d%d", &a, &n);
        read_num(a, &big_a);
        pow(&big_a, n, &result);
        output_num(&result);
    }
}

```

./1061.计算n！右端0的个数(II).c

```
// 8:06
// 8:23
// 8:44

#include<stdio.h>

int get_5_factor_nums_in_factorial(int num) {
    int result = 0;
    for (int i = 1; i <= num; i++) {
        int temp = i;
        while (temp % 5 == 0) {
            temp /= 5;
            result++;
        }
    }
    return result;
}

int main() {
    int t;
    scanf("%d", &t);
```

```
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n", i);
        printf("%d\n", get_5_factor_nums_in_factorial(n));
    }
}

```

./1062.计算2的N次方.c

```
// 8:59
// 9:02

#include<stdio.h>

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n%d\n", i, 1<<n);
    }
}

```

./1063.二进制倒置.c

```
// 9:04
// 9:15
// 9:36
// 10:01

#include<stdio.h>
#include<string.h>
#define N 101
#define M 334

// 倒置
typedef struct {
    int s[N];
    int cnt;
} Big;

void init_big(Big *num) {
    memset(num, 0, sizeof(Big));
    // num->cnt = 1; // 初始化为0的时候是0位数字
}

void read(char s[], Big *num) {
    init_big(num);
```

```
    for (int i = strlen(s) - 1; i >= 0; i--) {
        num->s[num->cnt++] = s[i] - '0';
    }
}

void output(Big *num) {
    for (int *p = num->s + num->cnt - 1; p >= num->s; p--) {
        putchar(*p + '0');
    }
    putchar('\n');
}

// 顺便更新num->cnt
void carry(Big *num) {
    for (int i = 0; i < N - 1; i++) {
        if (num->s[i] >= 10) {
            num->s[i + 1] += num->s[i] / 10;
            num->s[i] %= 10;
        }
    }
    int i = N - 1;
    for (; i > 0; i--) {
        if (num->s[i] != 0) break;;
    }
    num->cnt = i + 1; // 如果数字为0, cnt是1
}

// 返回余数
int divide_by_2(Big *num) {
    int borrow = 0;
    for (int i = num->cnt - 1; i > 0; i--) {
        borrow += num->s[i];
        if (borrow < 2){
            num->s[i] = 0;
            borrow *= 10;
            continue;
        }
        num->s[i] = borrow >> 1;
        borrow &= 1;
        borrow *= 10;
    }
    borrow += num->s[0];
    num->s[0] = borrow >> 1;
    borrow &= 1;
    carry(num);
    return borrow;
}

// 返回二进制位数
int read_to_binary(Big *num, int binary[]) {
    int *p = binary;
    for (; num->cnt != 1 || num->s[0] != 0; p++) {
        *p = divide_by_2(num);
    }
    return p - binary;
```

```
}

void multiply_2(Big *num) {
    for (int i = 0; i < num->cnt; i++) num->s[i] <<= 1;
    carry(num);
}

void add_1(Big *num) {
    num->s[0] += 1;
    carry(num);
}

void binary_to_big(int binary[], int binary_len, Big *num) {
    init_big(num);
    num->cnt = 1; // 初始化后作为1处理
    for (int i = 0; i < binary_len - 1; i++) {
        if (binary[i]) add_1(num);
        multiply_2(num);
    }
    if (binary[binary_len - 1]) add_1(num);
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        char s[N + 1];
        scanf("%s", &s);
        Big n;
        read(s, &n);
        int binary[M];
        int binary_len = read_to_binary(&n, binary);
        binary_to_big(binary, binary_len, &n);
        printf("case #%d:\n", i);
        output(&n);
    }
}

```

./1064.A-B(Big Integer).c

```
// 10:11

#include<stdio.h>
#include<string.h>
#define N 500

// 倒置
typedef struct {
    int s[N];
    int cnt;
    int sign;
} Big;

```

```
void init(Big *num) {
    memset(num, 0, sizeof(Big));
    num->sign = 1;
}

void input(Big *num, char s[]) {
    init(num);
    for (char *p = s + strlen(s) - 1; p >= s; p--) {
        num->s[num->cnt++] = *p - '0';
    }
}

void output(Big *num) {
    if (num->sign < 0) putchar('-');
    for (int i = num->cnt - 1; i > 0; i--) {
        putchar(num->s[i]);
    }
    putchar('\n');
}

void carry(Big *num) {
    for (int i = 0; i < N - 1; i++) {
        if (num->s[i] >= 10) {
            num->s[i + 1] = num->s[i] / 10;
            num->s[i] /= 10;
        }
    }
    int i = N - 1;
    for (; i > 0; i--) {
        if (num->s[i] != 0) break;
    }
    num->cnt = i;
}

int cmp(Big *a, Big *b) {
    if (a->cnt != b->cnt) return a->cnt - b->cnt;
    for (int i = a->cnt - 1; i >= 0; i--) {
        if (a->s[i] == b->s[i]) continue;
        if (a->s[i] > b->s[i]) return 1;
        else return -1;
    }
    return 0;
}

void minus(Big *a, Big *b, Big *result) {
    init(result);
    int borrow = 0;
    for (int i = N - 1; i >= 0; i++) {
        borrow += a->s[i];
        if (borrow < b->s[i]) {
            result->s[i + 1]--;
            borrow += 10;
        }
        result->s[i] = borrow - b->s[i];
        borrow = 0;
```

```
    }
    carry(result);
}

int main() {
    char a[N + 1], b[N + 1];
    Big A, B, result;
    while (scanf("%s", a) != EOF) {
        scanf("%s", b);
        input(&A, a);
        input(&B, b);
        if (cmp(&A, &B) < 0) {
            minus(&B, &A, &result);
            result.sign = -1;
        }
        minus(&A, &B, &result);
        output(&result);
    }
}

```

./1065.浮点数减法.c

```c
// 10:34
// 11:55

// 13:32
// 14:53

#include<stdio.h>
#include<string.h>
#define N 600
#define M 50

typedef struct {
    int s[2 * N];
    int cnt; // 最右边的非零数字的位置
    int sign;
} BigFloat;
// 前N位是小数，后N位是整数

void init(BigFloat *num) {
    memset(num, 0, sizeof(BigFloat));
    num->sign = 1;
//    num->cnt = 1;
}

void read_num(BigFloat *num, char *s) {
    init(num);
    if (*s == '-') {
        num->sign = -1;
        s++;
    }
    char *p_point = strchr(s, '.');
```

```
    char *end = s + strlen(s);
    int *p_result = num->s + N;
    if (p_point == NULL) p_point = end;
    for (char *p = p_point + 1; p < end; p++) {
        *--p_result = *p - '0';
    }
    p_result = num->s + N;
    for (char *p = p_point - 1; p >= s; p--) {
        *p_result++ = *p - '0';
    }
//    num->cnt = p - num->s - N;
}

int cmp(BigFloat *a, BigFloat *b) {
    for (int i = 2 * N - 1; i >= 0; i--) {
        if (a->s[i] == b->s[i]) continue;
        return a->s[i] - b->s[i];
    }
    return 0;
}

void carry(BigFloat *num, int n) {
    for (num->cnt = 2 * N - 1; num->cnt > 0; num->cnt--) {
        if (num->s[num->cnt] != 0) break;
    }
    for (int i = 0; i <= num->cnt; i++) {
        if (num->s[i] < 0) {
            num->s[i] += 10;
            num->s[i + 1] --;
        }
        if (num->s[i] >= 10) {
            num->s[i + 1] += num->s[i] / 10;
            num->s[i] %= 10;
        }
    }
    for (num->cnt = 2 * N - 1; num->cnt > 0; num->cnt--) {
        if (num->s[num->cnt] != 0) break;
    }
    if (num->s[num->cnt] < 0) {
        for (int i = 0; i < 2 * N; i++)
            num->s[i] *= -1;
        num->sign = -1;
        carry(num, n);
    }
}

void minus(BigFloat *a, BigFloat *b, BigFloat *result, int n) {
    init(result);
    for (int i = 0; i < 2 * N; i++) {
        result->s[i] = a->s[i] - b->s[i];
    }
//    if (result->s[N - n - 1] >= 5) result->s[N - n]++;
    carry(result, n);
    if (result->s[N - n - 1] >= 5) {
        result->s[N - n]++;
```

```
        carry(result, n);
    }
}

//void minus_with_cmp(BigFloat *a, BigFloat *b, BigFloat *result) {
//
//}

void output(BigFloat *num, int n){
    int *p = num->s;
    if (num->sign < 0) {
        printf("-");
        p++;
    }
    int i = 2 * N - 1;
    for (; i > N; i--) {
        if (num->s[i] != 0) break;
    }
    for (; i >= N; i--) {
        putchar(num->s[i] + '0');
    }
    putchar('.');
//    int end_0 = 0;
//    for (; end_0 < N; end_0++) {
//        if (num->s[end_0]) break;
//    }
    for (int i = N - 1; i >= N - n; i--) {
        putchar(num->s[i] + '0');
    }
    putchar('\n');
}

int main() {
    char s[N + 1];
    BigFloat a, b, result;
    scanf("%s", s);
    read_num(&a, s);
    scanf("%s", s);
    read_num(&b, s);
    int n;
    scanf("%d", &n);
    minus(&a, &b, &result, n);
    output(&result, n);
    return 0;
}


```

./1081.种田.c

```
// 9:28
// 9:44

#include<stdio.h>
```

```c

// long long body_power[;]

long long get_body_power(long long x, long long y) {
    long long max_square_num = y / x;
    long long result = 4 * x * max_square_num;
    y %= x;
    if (y <= 0)
        return result;
    // if (y > x)
    //     return result + get_body_power(x, y);
    // else
    return result + get_body_power(y, x);
}

int main() {
    long long x, y;
    scanf("%lld %lld", &x, &y);
    printf("%lld\n", get_body_power(y, x));
    return 0;
}

```

./1082.波兰表达式.c

```c
// 9:47
// 10:20

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

char *current_pos;
char expression[151];

double calculate(char *str) {
    char *operator_;
    operator_ = strtok(str, " ");
    current_pos = operator_ + strlen(operator_) + 1;
    if (strlen(operator_) > 1 || isdigit(*operator_)) { // 说明是个数字
        return atof(operator_);
    }
    double operand1 = calculate(current_pos);
    double operand2 = calculate(current_pos);
    switch (*operator_) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            return operand1 / operand2;
```

```
    }
}

int main(int argc, char *argv[]) {
    int t;
    scanf("%d", &t);
    getchar(); // 去掉换行
    for (int i = 0; i < t; i++) {
        fgets(expression, 150, stdin);
        printf("case #%d:\n%.6lf\n", i, calculate(expression));
    }
    return 0;
}

```

./1083.地铁站.c

```
// 10:26
// 10:53

#include <stdio.h>

long long board_people[21] = {0}, unboard_people[21] = {0},
people_aboard[21] = {0};

// t是第二站上下车的人数
long long get_last_unboard_people(int a, int t, int n) {
    board_people[1] = a;
    people_aboard[1] = a;
    board_people[2] = t;
    unboard_people[2] = t;
    people_aboard[2] = a;
    for (int i = 3; i < n; i++) {
        board_people[i] = board_people[i - 1] + board_people[i - 2];
        unboard_people[i] = board_people[i - 1];
        people_aboard[i] = people_aboard[i - 1] + board_people[i] -
unboard_people[i];
    }
    board_people[n] = 0;
    unboard_people[n] = people_aboard[n - 1];
    people_aboard[n] = 0;
    return unboard_people[n];
}

int main() {
    int a, n, m, x;
    scanf("%d %d %d %d", &a, &n, &m, &x);
    for (int t = 0; t < m; t++) { // t从1开始的话最后一个测试用例就不对。
    // 说好的第2站有人上下车呢? 还能有0人上下车的?
        if (m == get_last_unboard_people(a, t, n))
            break;
    }
    printf("%lld\n", people_aboard[x]);
}
```

```
36
```

./1084.Fj&haozi.c

```c
// 8:09
// 8:50

#include<stdio.h>
#include<string.h>
#define MAX_N 101 // 原来写的21导致出错

long long previous_num[MAX_N] = {0};
long long current_num[MAX_N] = {0};

void expand_num(int n, int m) {
    for (int i = 0; i < m; i++) {
        for (int j = 2; j < n; j++) {
            current_num[j] = previous_num[j - 1] + previous_num[j + 1];
        }
        current_num[1] = previous_num[2];
        current_num[n] = previous_num[n - 1];
        memcpy(previous_num, current_num, MAX_N * sizeof(long long));
        memset(current_num, 0, MAX_N * sizeof(long long));
    }
}

int main() {
    int n, m, p, t;
    int cas;
    scanf("%d", &cas);
    while(cas--) {
        scanf("%d %d %d %d", &n, &p, &m, &t);
        memset(previous_num, 0, MAX_N * sizeof(long long));
        memset(current_num, 0, MAX_N * sizeof(long long));
        previous_num[p] = 1;
        expand_num(n, m);
        printf("%lld\n", previous_num[t]);
    }
}


```

./1085.泰波那契数列的前74项.c

```c
// 8:14
// 9:23

#include<stdio.h>

long long tribonacci[74] ={0LL, 1LL, 1LL};

int main(){
```

```
    for(int i = 3; i < 74; i++){
        tribonacci[i] = tribonacci[i - 1] + tribonacci[i - 2] + tribonacci[i - 3];
    }
    int t;
    scanf("%d",&t);
    for (int i = 0; i < t; i++) {
        int n;
        scanf("%d", &n);
        printf("case #%d:\n%lld\n", i, tribonacci[n]);
    }
    return 0;
}

```

./1087.统计字符串个数.c

```c
// 9:41
// 10:01

#include <stdio.h>
#include <string.h>
#define substring_width 3

int end_with_0[substring_width];
int end_with_1[substring_width];

void init() {
    memset(end_with_0, 0, substring_width * sizeof(end_with_0[0]));
    memset(end_with_1, 0, substring_width * sizeof(end_with_1[0]));
}

void expand(int n) {
    end_with_0[0] = 0;
    end_with_1[0] = 0;
    end_with_0[1] = 1;
    end_with_1[1] = 1;
    for (int i = 2; i < n + 1; i++) {
        end_with_0[2] = end_with_0[1] + end_with_1[1];
        end_with_1[2] = end_with_0[2] - end_with_1[0];
        memmove(end_with_0, end_with_0 + 1, substring_width * sizeof(end_with_0[0]));
        memmove(end_with_1, end_with_1 + 1, substring_width * sizeof(end_with_1[0]));
    }
}

int main() {
    int n = 0;
    while (1) {
        scanf("%d", &n);
        if (n < 0)
            break;
        init();
```

```
        expand(n);
        printf("%d\n", end_with_0[1] + end_with_1[1]);
    }
    return 0;
}

```

./1090.最短路径.c

```c
// 9:19
// 9:39

#include<stdio.h>
#define MAX_N 200

int matrix[MAX_N][MAX_N];
int loss[MAX_N][MAX_N];
char route[MAX_N][MAX_N];

void expand_route(int m, int n) {
    loss[0][0] = matrix[0][0];
    for (int i = 1; i < m; i++) {
        loss[i][0] = loss[i - 1][0] + matrix[i][0];
        route[i][0] = 'D';
    }
    for (int i = 0; i < n; i++) {
        loss[0][i] = loss[0][i - 1] + matrix[0][i];
        route[0][i] = 'R';
    }
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            if (loss[i][j - 1] < loss[i - 1][j]) {
                loss[i][j] = loss[i][j - 1] + matrix[i][j];
                route[i][j] = 'R';
            }
            else {
                loss[i][j] = loss[i - 1][j] + matrix[i][j];
                route[i][j] = 'D';
            }
        }
    }
}

void output_route(int i, int j) {
    if (i == 0 & j == 0) {
        return;
    }
    if (route[i][j] == 'R') {
        output_route(i, j - 1);
        putchar(route[i][j]);
    } else {
        output_route(i - 1, j);
        putchar(route[i][j]);
    }
```

```c
}

int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    expand_route(m, n);
    printf("%d\n", loss[m - 1][n - 1]);
    output_route(m - 1, n - 1);
}

```

./1093.波浪图.c

```c
// 15:00
// 15:23

#include<stdio.h>
#include<string.h>
#define N 80

typedef char Picture[2 * N][2 * N];

void get_pic(Picture pic, char *s) {
    memset(pic, ' ', sizeof(Picture));
    int row = N, col = 0;
    pic[row][col] = *s;
    for (s++; *s; s++) {
        col++;
        if (*s > *(s - 1)) row--;
        if (*s < *(s - 1)) row++;
        pic[row][col] = *s;
    }
}


void output(Picture pic, int len) {
    char blank[2 * N];
    memset(blank, ' ', sizeof(blank));
    int start_row = 0, end_row = 2 * N - 1;
    for (; memcmp(pic[start_row], blank, sizeof(blank)) == 0; start_row++);
    for (; memcmp(pic[end_row], blank, sizeof(blank)) == 0; end_row--);
    for (int i = start_row; i <= end_row; i++) {
        for (int j = 0; j < len; j++) {
            putchar(pic[i][j]);
        }
        putchar('\n');
    }
}

```

```c
int main() {
    char s[N + 1];
    Picture pic;
    while (scanf("%s", s) != EOF) {
        get_pic(pic, s);
        output(pic, strlen(s));
    }
}


```

./1094.坏掉的彩灯.c

```c
// 15:07
// 16:19

// 18:38
// 18:51

// 19:18
// 19:19

#include<stdio.h>
#include<string.h>
#define N 100

char Colour[] = "RBYG";
char Index[128];

void init() {
    Index['R'] = 0;
    Index['B'] = 1;
    Index['Y'] = 2;
    Index['G'] = 3;
}

int global_valid = 0;

int get_possible_solution(char s[], char *result, int len) {
//    int len = strlen(s);
    int i = 0;
    for (; i < len; i++) {
        if (s[i] == '!') break;
    }
    // printf("%s\n%d %d\n", result, i, len);
    if (i == len) {
        global_valid = 1;
        memcpy(result, s, (N + 1) * sizeof(char));
        return 1;
    }
    int possible[] = {1, 1, 1, 1}; // RBYG
    for (int j = i - 3; j <= i + 3; j++) {
        if (j >= 0 && j < len && s[j] != '!') {
            possible[Index[s[j]]] = 0;
```

```
        }
    }
    int valid = 0;
    for (int j = 0; j < 4; j++)
        if (possible[j]) valid = 1;
    if (valid == 0) {
//        memcpy(result, s, N); // 还原
        return 0; // 表明这种情况不可能
    }
    for (int j = 0; j < 4; j++) {
        if (possible[j]) {
            char temp[N + 1];
            memcpy(result, s, (N + 1) * sizeof(char));
            result[i] = Colour[j];
            memcpy(temp, result, (N + 1) * sizeof(char));
            global_valid = get_possible_solution(temp, result, len);
            if (global_valid) return 1;
        }
    }
    return 0;
}

int get_broken_nums(char good[], char broken[], int colors[], int len) {
//    int len = strlen(broken);
    for (int i = 0; i < len; i++) {
        if (broken[i] == '!') {
            colors[Index[good[i]]] += 1;
        }
    }
}

int main() {
    init();
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        char s[N + 1], result[N + 1];
        scanf("%s", s);
        int len = strlen(s);
        memcpy(result, s, (N + 1) * sizeof(char));
        global_valid = 0;
        int ret = get_possible_solution(s, result, len);
        int colors[4] = {0};
        get_broken_nums(result, s, colors, len);
        printf("case #%d:\n", i);
        for (int j = 0; j < 3; j++) {
            printf("%d ", colors[j]);
        }
        printf("%d\n", colors[3]);
    }
}
```

./A.进制转换.c

```c
#include<stdio.h>

char str[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main(){
    int t, n, r;
    char result[33];
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        scanf("%d %d", &n, &r);
        if (n < 0) {
            n = - n;
            printf("-");
        }
        int j = 0;
        for (; n; n/=r, j++) result[j] = str[n%r];
        while (--j >= 0) printf("%c", result[j]);
        printf("\n");
    }
}

```

./B.神秘信息.c

```c
#include<stdio.h>

int main(){
    int a[128];
    int t, digit, n;
    long long result;
    char s[61], *p;
    scanf("%d", &t);
    for (int i=0; i<t; i++) {
        printf("case #%d:\n", i);
        for (int i=0; i<128; i++) a[i] = -1;
        scanf("%s", s);
        a[*s] = -1;
        n = 1;
        p = s;
        digit = 0;
        a[*p] = 1;
        while (*++p){
            if (a[*p] != -1) continue;
            a[*p] = digit;
            digit = digit ? digit + 1 : 2;
            n ++;
        }
        if (n < 2) n = 2;
        p = s;
        result = 0;
```

```
        while (*p)
            result = result * n + a[*p++];
        printf("%lld\n", result);
    }
}

```

./C.按数据中1的位数排序.c

```c
#include <stdio.h>
#include <stdlib.h>

# define BIT_NUM 64
# define MAX_N 10000
typedef struct
{
    long long value;
    int number;
} Data;

int cmp(const void *pa, const void *pb){
    Data a = *(Data*)pa;
    Data b = *(Data*)pb;
    if (a.number == b.number) {
        if (a.value == b.value)
            return 0;
        else
            return a.value > b.value ? 1 : -1;
    } else
        return b.number - a.number;
}

int main() {
    int t, n;
    Data nums[MAX_N];
    scanf("%d", &t);
    for (int i=0; i<t; i++) {
        printf("case #%d:\n", i);
        scanf("%d", &n);
        for (int j=0; j<n; j++) {
            scanf("%lld", &nums[j].value);
            nums[j].number = 0;
            long long d = 1;
            for (int k=0; k<BIT_NUM; k++) {
                if (nums[j].value & d) nums[j].number++;
                d <<= 1;
            }
        }
        qsort(nums, n, sizeof(nums[0]), cmp);
        for (int j=0; j<n; j++)
            printf("%lld ", nums[j].value);
        printf("\n");
    }
}
```

```
46
```

./D.内存显示.c

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void solveint(int n){
    for (int i=0; i<sizeof(n); i++){
        printf("%02x ", ((unsigned char*)&n)[i]);
    }
}

void solvedouble(double d){
    for (int i=0; i<sizeof(d); i++){
        printf("%02x ", ((unsigned char*)&d)[i]);
    }
}

int main(){
    char s[31];
    while (scanf("%s", s) != EOF){
        if (strchr(s, '.') == 0)
            solveint(atoi(s));
        else
            solvedouble(atof(s));
        printf("\n");
    }
    return 0;
}

```

./E.平衡三进制.c

```c
#include<stdio.h>

int main(){
    int t;
    char s[30];
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        printf("case #%d:\n", i);
        scanf("%s", s);
        int result = 0;
        for (int j=0; s[j]; j++){
            switch (s[j]){
            case '-':
                result = result * 3 - 1;
                break;
            case '0':
                result = result * 3;
```

```
                break;
            case '1':
                result = result * 3 + 1;
                break;
            }
        }
        printf("%d\n", result);
    }
}

```

./F.非重复二进制串.c

```c
#include<stdio.h>

int main(){
    int t;
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        int n;
        printf("case #%d:\n", i);
        scanf("%d", &n);
        int max_diff_bin_len = 1, diff_bin_len = 1; // 初始值应为1，因为如果大于0的话，一位肯定是不重复的。
        for (; n > 1; n >>= 1){ // 最左的01不能算，因为前置0不计算在内，所以右移到01时就可以停止了，大于1代表可以继续右移。
            switch (n & 0b11){
                case 0b10:
                case 0b01:
                    diff_bin_len ++;
                    if (diff_bin_len > max_diff_bin_len)
                        max_diff_bin_len = diff_bin_len;
                    break;
                case 0b00:
                case 0b11:
                    diff_bin_len = 1;
            }
        }
        printf("%d\n", max_diff_bin_len);
    }
}

```

./G.二进制位不同的个数.c

```c
#include<stdio.h>

int main(){
    int t;
    scanf("%d", &t);
    for (int i=0; i<t; i++){
        int x, y;
```

```
        scanf("%d %d", &x, &y);
        int distance = 0;
        int xor = x ^ y;
        for (; xor > 0; xor >>= 1){
            distance += xor & 1;
        }
        printf("%d\n", distance);
    }
}

```

./H.数据密度.c

```
#include<stdio.h>
#define MAX_LEN 130

int bin_with_1(char *pc){
    unsigned char c = *(unsigned char*)pc; // 中文的每个字节首位可能是1
    int result = 0;
    do{
        result += c & 1;
    }while (c >>= 1);
    return result;
}

/* 只能计算b > a情况时的最大公因数 */
int gcd(int a, int b){
    if (a <= 0) return b;
    int temp = b % a;
    return gcd(temp, a);
}

int main(){
    char s[500], *pchar;
    int n;
    scanf("%d", &n);
    getchar(); // 因为scanf不会读取换行符，所以要手动把第一个换行符读取了
    while (n--){
        fgets(s, MAX_LEN, stdin); // 会把结尾的换行符也读进来
        // getline(&s, MAX_LEN, stdin);
        int sum_with_bin_1 = 0;
        for (pchar = s; (*pchar != '\n') && *pchar; pchar++){
            sum_with_bin_1 += bin_with_1(pchar);
        }
        int sum_of_bin = (pchar - s) * 8;
        int divisor = gcd(sum_with_bin_1, sum_of_bin);
        printf("%d/%d\n", sum_with_bin_1 / divisor, sum_of_bin / divisor);
    }
}

```

./I.QR Code.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
#define NUM_SIGN "0001"
#define MAX_LEN 500
#define GROUP_LEN 3


void itob_print(int a, int bitlen){
    char *result = (char*)malloc(bitlen + 1);
    result[bitlen] = '\0';
    while (--bitlen >= 0){
        result[bitlen] = (a & 1) + '0';
        a >>= 1;
    }
    printf(result);
    free(result);
}

int main(){
    char s[MAX_LEN + 1], *p_char;
    scanf("%s", s);
    printf(NUM_SIGN);
    itob_print(strlen(s), 10);
    int part_len=0, part_num=0;
    for (p_char = s; *p_char; p_char++){
        part_len ++;
        part_num = part_num * 10 + *p_char - '0';
        if (part_len >= GROUP_LEN){ // 每组的数字已完成
            itob_print(part_num, 10);
            part_len = 0;
            part_num = 0;
        }
    }
    switch (part_len){ // 还有剩余的一些数字没输出
        case 1:
            itob_print(part_num, 4);
            break;
        case 2:
            itob_print(part_num, 7);
            break;
    }

}

```

./J.i-1进制（Easy）.c

```c
// 实部
// sqrt s
// 0 s2 2 2s2 4 4s2 8 8s2 16 16s2
// 虚部
```

```c
// 0 3 6 9 12 15 18 21 24 27 30
// 0 3 6 1  4  7  2  5  0  3  6
// 18:04
#include<stdio.h>
#include<string.h>

typedef struct {
    long long real;
    long long imaginary;
} Complex;

/* 计算-1 + i的幂 */
Complex pow_of_complex(int multiple_times) {
    // if (multiple_times <= 0) return {1LL, 0LL};
    long long modulus = 1LL << (multiple_times / 2); // 偶数时是模长的意思，奇数时是实部或虚部的绝对值（实部和虚部绝对值相等）
    Complex result = {0LL, 0LL};
    switch (multiple_times % 8) {
        case 0: // 0 * 45°
            result.real = modulus;
            break;
        case 1: // 3 * 45°
            result.real = - modulus;
            result.imaginary = modulus;
            break;
        case 2: // 6 * 45°
            result.imaginary = - modulus;
            break;
        case 3: // 1 * 45°
            result.real = modulus;
            result.imaginary = modulus;
            break;
        case 4: // 4 * 45°
            result.real = - modulus;
            break;
        case 5: // 7 * 45°
            result.real = modulus;
            result.imaginary = - modulus;
            break;
        case 6: // 2 * 45°
            result.imaginary = modulus;
            break;
        case 7: // 5 * 45°
            result.real = - modulus;
            result.imaginary = - modulus;
            break;
    }
    return result;
}

/* 两个复数即使在不需要二进制进位的情况下相加，也不能直接使用异或，因为负数的时候异或就不对了 */
Complex add(Complex a, Complex b) {
    Complex result;
    result.real = a.real + b.real;
```

```
    result.imaginary = a.imaginary + b.imaginary;
    return result;
}

int main(){
    char str_hex_num[100];
    Complex result = {0LL, 0LL};
    scanf("0x%s", str_hex_num);
    char *pos_hex_num = str_hex_num;
    for (int weight=strlen(str_hex_num) - 1; weight>=0; weight--, pos_hex_num++){
        int hex_num;
        if (*pos_hex_num <= '9')
            hex_num = *pos_hex_num - '0';
        else
            hex_num = *pos_hex_num - 'A' + 10;
        for (int i=0; hex_num>0; hex_num >>= 1, i++){
            if (hex_num & 1)
                result = add(result, pow_of_complex(4 * weight + i));
        }
    }
    if (result.real > 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '1' || *str_hex_num == 'E'))
        result.real --; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.real != 0 && result.imaginary > 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '3' || *str_hex_num == '7' || *str_hex_num == 'E'))
        result.imaginary --; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.imaginary < 0 && 4 * strlen(str_hex_num) >= 128 && (*str_hex_num == '7' ))
        result.imaginary ++; // 看了测试用例后发现超出范围的都是输出比答案多了1，所以手动判断是否超出范围
    if (result.imaginary == 0) {
        printf("%lld", result.real);
    }else if (result.real == 0 ) {
        if (result.imaginary == 1) printf("i");
        else if (result.imaginary == -1) printf("-i");
        else printf("%lldi", result.imaginary);
    }else{
        if (result.imaginary == 1) printf("%lld+i", result.real);
        else if (result.imaginary == -1) printf("%lld-i", result.real);
        else printf("%lld%+lldi", result.real, result.imaginary);
    }
    return 0;
}
// 20:31

```

./K.-1+i进制.c

```
// 21:13
#include<stdio.h>
```

```c
#include<regex.h>
#include<string.h>
#include<stdlib.h>
#include<malloc.h>
#define MAX_E 50 // 最大位数的整数
#define MAX_LINE 100 // 输入的一行最大字符

typedef struct {
    long long real;
    long long imaginary;
} Complex;

regex_t reg_full, reg_real, reg_imaginary;
regmatch_t matches[10];
int ret = 0;
int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
char input[MAX_LINE] = {'\0'};

int getsign(char *input, regmatch_t *match) {
    // printf("%s %d %d\n", input, match->rm_so, match->rm_eo);
    if (match->rm_so < 0 || match->rm_so == match->rm_eo) // 省略正号
        return 1;
    else if (*(input + match->rm_so) == '+')
        return 1;
    return -1;
}

long long getabs(char *input, regmatch_t *match) {
    // printf("%s %d %d\n", input, match->rm_so, match->rm_eo);
    if (match->rm_so < 0 || match->rm_so == match->rm_eo) // 省略1
        return 1;
    char *abs_str = malloc((match->rm_eo - match->rm_so + 1) *
sizeof(char));
    int i;
    for (i = match->rm_so; i < match->rm_eo; i++) {
        abs_str[i - match->rm_so] = input[i];
    }
    abs_str[i - match->rm_so] = '\0'; // 之前这里少了 - match->rm_so
    // char *abs_str = strndup(input + match->rm_so, match->rm_eo - match-
>rm_so); // 使用这个在本地运行不会报错，在OJ运行会Runtime error: SIGSEGV
    long long result = atoll(abs_str);
    free(abs_str);
    return result;
}

Complex get_complex(char *input, int limit) {
    Complex input_complex = {0LL, 0LL};
    fgets(input, limit, stdin);
    ret = regexec(&reg_full, input, 5, matches, 0);
    if (ret) { // 省略了实部或虚部
        if (strchr(input, 'i') == NULL) { // 只有实部
            ret = regexec(&reg_real, input, 3, matches, 0);
            input_complex.real = getsign(input, matches + 1)
                                 * getabs(input, matches + 2);
        } else { // 只有虚部
```

```
            ret = regexec(&reg_imaginary, input, 3, matches, 0);
            input_complex.imaginary = getsign(input, matches + 1)
                                      * getabs(input, matches + 2);
        }
    } else {
        input_complex.real = getsign(input, matches + 1)
                             * getabs(input, matches + 2);
        input_complex.imaginary = getsign(input, matches + 3)
                                  * getabs(input, matches + 4);
    }
    return input_complex;
}

Complex divide_once(Complex a) {
    Complex result;
    // >>1是向下取整，/2是向0取整
    result.real = a.imaginary / 2 - a.real / 2;
    result.imaginary = - a.real / 2 - a.imaginary / 2;
    if (a.real & 1) { // 这里已经确保实部虚部奇偶性相同，实部虚部都为奇数时上面两行会偏差1，对负数用%2会出现问题，所以应用&1来判断奇偶
        if (a.real < 0 && a.imaginary < 0)
            result.imaginary ++; // 因为向0取整，所以向远离0的地方偏移1
        if (a.real > 0 && a.imaginary > 0)
            result.imaginary --;
        if (a.real < 0 && a.imaginary > 0)
            result.real ++; // 异号时实部会有偏差
        if (a.real > 0 && a.imaginary < 0)
            result.real --;
    }
    return result;
}

void print_bin_complex(Complex a) {
    if (a.imaginary == 0 && (a.real == 1 || a.real == 0)){
        printf("%lld", a.real);
        return;
    } else if ((a.real & 1) != (a.imaginary & 1)) { // 负奇数对2取余是-1，因此要取绝对值，但用&1来判断就不用取绝对值了
        a.real --;
        print_bin_complex(divide_once(a));
        printf("1");
    } else {
        print_bin_complex(divide_once(a));
        printf("0");
    }
}

// 21:42
// 11:00
// 11:51
// 13:49
// 17:03
// 19:20
// 20:38
int main(){
```

```
    // Complex input_complex = {{"0", POS}, {"0", POS}};
    regcomp(&reg_full, "(-)?([0-9]+)(\\+|-)([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符，不加^来忽略可能不省略的分号
    regcomp(&reg_real, "(-)?([0-9]+)$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    regcomp(&reg_imaginary, "(-)?([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    Complex input_complex = get_complex(input, MAX_LINE - 1);
    // printf("%lld %lld\n", input_complex.real, input_complex.imaginary);
    // Complex temp_complex = get_complex(input, MAX_LINE);
    print_bin_complex(input_complex);
}

```

./K.-1+i进制（长整数加减法未完成）.c

```c
// 21:13
#include<stdio.h>
#include <sys/types.h>
#include <regex.h>
#include<string.h>
#define MAX_E 50 // 最大位数的整数
#define MAX_LINE 100 // 输入的一行最大字符

typedef struct {
    char *abs; // char abs[MAX_E]
    enum {NEG=-1, POS=1} sign;
} Decimal;

typedef struct {
    Decimal real;
    Decimal imaginary;
} Complex;

regex_t reg_full, reg_real, reg_imaginary;

Decimal add(Decimal a, Decimal b) {
    char abs[MAX_E] = "0";
    Decimal result={abs, POS};
    if (a.sign == b.sign) {
        result.sign = a.sign;
        for (int i=MAX_E; i>0; i--) {
            int temp = a.abs[i] + b.abs[i];
            result.abs[i] = temp % 10;
            result.abs[i - 1] = temp / 10;
        }
    } else if (a.abs > b.abs) {
        result.sign = a.sign;
        for (int i=MAX_E; i>0; i--) {
            int temp = a.abs[i] - b.abs[i];
            result.abs[i] += temp % 10;
            result.abs[i - 1] += temp / 10;
        }
    } else if (a.abs < b.abs) {
```

```
        result.sign = b.sign;
        for (int i=MAX_E; i>0; i--) {
            int temp = b.abs[i] - a.abs[i];
            result.abs[i] += temp % 10;
            result.abs[i - 1] += temp / 10;
        }
    }
    return result;
}

int getsign(char *input, regmatch_t *match) {
    if (match->rm_eo == match->rm_so)
        return POS;
    else if (*(input + match->rm_so) == '+')
        return POS;
    return NEG;
}

char *getabs(char *input, regmatch_t *match) {
    if (match->rm_eo == match->rm_so) // 省略1
        return "1";
    return strndup(input + match->rm_so, match->rm_eo - match->rm_so);
}

Complex get_complex(char *input, int limit) {
    int ret;
    regmatch_t matches[5];
    Complex input_complex = {{"0", POS}, {"0", POS}};
    fgets(input, MAX_LINE, stdin);
    ret = regexec(&reg_full, input, 5, matches, 0);
    if (ret) { // 省略了实部或虚部
        if (strchr(input, 'i') == NULL) { // 只有实部
            ret = regexec(&reg_real, input, 3, matches, 0);
            input_complex.real.sign = getsign(input, matches + 1);
            input_complex.real.abs = getabs(input, matches + 2);
        } else { // 只有虚部
            ret = regexec(&reg_imaginary, input, 3, matches, 0);
            input_complex.imaginary.sign = getsign(input, matches + 1);
            input_complex.imaginary.abs = getabs(input, matches + 2);
        }
    } else {
        input_complex.real.sign = getsign(input, matches + 1);
        input_complex.real.abs = getabs(input, matches + 2);
        input_complex.imaginary.sign = getsign(input, matches + 3);
        input_complex.imaginary.abs = getabs(input, matches + 4);
    }
    return input_complex;
}

// 21:42
// 11:00
// 11:51
// 13:49
int main(){
    int ret;
```

```
    int cflags = REG_EXTENDED | REG_ICASE | REG_NEWLINE;
    char input[MAX_LINE];
    // Complex input_complex = {{"0", POS}, {"0", POS}};
    regcomp(&reg_full, "(-)?([0-9]+)(\\+|-)([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符，不加^来忽略可能不省略的分号
    regcomp(&reg_real, "(-)?([0-9]+)$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    regcomp(&reg_imaginary, "(-)?([0-9]*)i$", cflags); // 加入REG_NEWLINE使^$忽略换行符
    Complex input_complex = get_complex(input, MAX_LINE);
    printf("%d %s %d %s\n",
            input_complex.real.sign, input_complex.real.abs,
            input_complex.imaginary.sign, input_complex.imaginary.abs);
    // Complex temp_complex = get_complex(input, MAX_LINE);
    printf("%d %s\n",
            add(input_complex.real, input_complex.imaginary).sign,
            add(input_complex.imaginary, input_complex.real).abs);
}

```

./L.平衡三进制.c

```c
// 9:37
// >>> import math
// >>> math.log(3**30, 10)
// 14.313637641589873

#include<stdio.h>
#include<regex.h>
#include<stdlib.h>
#define MAX_LEN 50
#define BASE 3

/* 计算a < b时a和b的最大公因数 */
long long gcd(long long a, long long b) {
    if (a == 0) return b;
    return gcd(b % a, a);
}

long long tri_to_ll(char *a, int len) {
    long long result = 0;
    for (char *pos_num = a; pos_num < a + len; pos_num++) {
        int num = *pos_num - '0';
        if (num == 2) num = -1;
        result = result * BASE + num;
    }
    return result;
}

int main() {
    char tri[MAX_LEN];
    fgets(tri, MAX_LEN - 1, stdin);
    regex_t reg;
    regmatch_t matches[3];
```

```c
    int cflags = REG_NEWLINE | REG_EXTENDED | REG_ICASE;
    regcomp(&reg, "([0-9]+)\\.?([0-9]*)", cflags);
    int ret = regexec(&reg, tri, 3, matches, 0);
    // for (int i=1; i<3; i++) {
    //     if (matches[i].rm_so < 0) continue;
    //     for (int j=matches[i].rm_so; j<matches[i].rm_eo; j++){
    //         printf("%c", tri[j]);
    //     }
    //     printf("\n");
    // }
    long long int_part = tri_to_ll(tri + matches[1].rm_so, matches[1].rm_eo - matches[1].rm_so);
    long long numerator = tri_to_ll(tri + matches[2].rm_so, matches[2].rm_eo - matches[2].rm_so);
    long long denominator = 1;
    for (int point_pos = matches[2].rm_eo - matches[2].rm_so; point_pos > 0; point_pos--) {
        denominator *= 3;
    }
    if (int_part > 0 && numerator < 0) { // 借位
        int_part --;
        numerator += denominator;
    }
    if (int_part < 0 && numerator > 0) { // 借位
        int_part ++;
        numerator = denominator - numerator;
    }
    long long factor = gcd(llabs(numerator), llabs(denominator));
    if (int_part || numerator==0)
        printf("%lld ", int_part);
    if (numerator) {
        printf("%lld %lld", numerator / factor, denominator / factor);
    }
}
// 10:52

```

./M.平衡三进制II.c

```c
// 10:53
// >>> math.log(10**9,3)
// 18.86312946860446
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_LEN 50

int get_point_pos(long long denominator) {
    int point_pos = 0;
    while (denominator > 1) {
        denominator /= 3;
        point_pos ++;
    }
    return point_pos;
```

```c
}

/* 十进制转三进制，返回三进制的字符个数 */
int dec_to_tri(long long dec, char *tri) {
    if (dec < 3) {
        *tri = dec + '0';
        return 1;
    }
    int len = dec_to_tri(dec / 3, tri);
    *(tri + len) = dec % 3 + '0';
    return len + 1;
}

/* 注意，这里的target-1必须是一个合法的地址，用来存放进位 */
int add_tri(char *target, char *addtion, int len) {
    int carry = 0; // 存储进位
    for (int i = len - 1; i >= 0; i--) {
        target[i] += addtion[i] - '0' + carry;
        carry = 0;
        if (target[i] >= '3') {
            carry = 1;
            target[i] -= 3;
        }
    }
    if (carry) // 最后一次进位
        target[-1] ++;
}

int sub_ones(char *target, int len) {
    for (char *i = target; i < target + len; i++) {
        (*i) --;
        if (*i < '0') *i = '2';
    }
}

void my_strncpy(char *dest, char *src, int limit) {
    char *p_dest = dest, *p_src = src;
    while (*p_src && limit > 0) {
        *p_dest++ = *p_src++;
        limit --;
    }
    *p_dest = 0;
}

char *simplify_int_part(char *part, int len) {
    while (len > 1 && *part == '0')
        part++, len--;
    return part;
}

int simplify_fraction_part(char *part, int len) {
    while (len > 0 && part[--len] == '0')
        part[len] = 0;
    if (part[len] == '0' || part[len] == 0) return len;
    else return len + 1;
```

```
}

// 11:43
// 12:48
// 14:43

int main() {
    long long numerator, denominator;
    scanf("%lld %lld", &numerator, &denominator);
    char tri[MAX_LEN], ones[MAX_LEN];
    tri[0] = '0';
    int len = dec_to_tri(numerator, tri + 1); // 留出第一位可能的进位空间
    tri[1 + len] = '\0';
    // printf("%s\n", tri);
    for (int i = 0; i < len; i++)
        ones[i] = '1'; // 构造全1序列
    ones[len] = '\0';
    add_tri(tri + 1, ones, len);
    // printf("%s\n", tri);
    sub_ones(tri + 1, len);
    // printf("%s\n", tri);
    int point_pos = get_point_pos(denominator);
    // char *p_endpos = tri + 1 + len;
    // char *output = tri;
    // while (*output == '0') output++; // 去除前面的0
    // if (output >= p_endpos - point_pos) putchar('0'); // 纯小数
    // while (output < p_endpos - point_pos) putchar(*output++); // 输出整数部分
    // while (*(p_endpos - 1) == '0') p_endpos--; // 去除后面的0
    // if (point_pos > 0 && output < p_endpos) { // 判断是否需要输出小数部分
    //     putchar('.');
    //     for (int i = point_pos - (p_endpos - output); i > 1; i--) {
    //         putchar('0'); // 纯小数
    //     }
    //     while (output < p_endpos) putchar(*output++);
    // }
    int full_len = len + 1; // 三进制数字总长度
    char tri_with_start_0[MAX_LEN];
    int i = 0;
    for (; i < point_pos - full_len + 1; i++) {
        tri_with_start_0[i] = '0';
    }
    int j = 0;
    while (tri[j]) tri_with_start_0[i++] = tri[j++];
    tri_with_start_0[i] = 0;
    full_len = strlen(tri_with_start_0);
    char int_part[MAX_LEN], fraction_part[MAX_LEN];
    my_strncpy(int_part, tri_with_start_0, full_len - point_pos);
    my_strncpy(fraction_part, tri_with_start_0 + full_len - point_pos, point_pos);
    char *new_int_part = simplify_int_part(int_part, strlen(int_part));
    int fraction_len = simplify_fraction_part(fraction_part, strlen(fraction_part));
    printf("%s", new_int_part);
    if (fraction_len > 0)
```

```
        printf(".%s",fraction_part);
    return 0;
}


```

./library/大整数计算1.c

```c
// https://blog.csdn.net/u013113678/article/details/113060506

#include<stdint.h>
#include<stdio.h>
#include<string.h>
//计算位数，范围[1,9]
#define NUM_DIGIT 9
//数组类型
#if NUM_DIGIT<=0
    static_assert(1, "NUM_DIGIT must > 0 and <=9");
#elif NUM_DIGIT <=2
#define _NUM_T int8_t
#elif NUM_DIGIT <=4
#define _NUM_T int16_t
#elif NUM_DIGIT <=9
//NUM_DIGIT在[5,9]时用int32就可以装载数据，但是实测发现用int32在32位程序中，对除法性能有影响，用int64反而正常，这是比较奇怪的现象。
#define _NUM_T int64_t
#else
    static_assert(1, "NUM_DIGIT must > 0 and <=9");
#endif
//计算进制
    static const size_t _hex = NUM_DIGIT == 1 ? 10 : NUM_DIGIT == 2 ? 100 : NUM_DIGIT == 3 ? 1000 : NUM_DIGIT == 4 ? 10000 : NUM_DIGIT == 5 ? 100000 : NUM_DIGIT == 6 ? 1000000 : NUM_DIGIT == 7 ? 10000000 : NUM_DIGIT == 8 ? 100000000 : NUM_DIGIT == 9 ? 1000000000 : -1;
#define _MIN_NUM_SIZE 30/NUM_DIGIT

    /// <summary>
    /// 通过字符串初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]字符串首地址</param>
    /// <param name="len">[in]字符串长度</param>
    /// <returns>数据长度</returns>
    static size_t initByStr(_NUM_T* a, const char* value, size_t len)
    {
        size_t  shift = 10;
        size_t i, j, k = 0, n;
        n = len / NUM_DIGIT;
        for (i = 0; i < n; i++)
        {
            a[i] = (value[len - k - 1] - '0');
            for (j = 1; j < NUM_DIGIT; j++)
            {
                a[i] += (value[len - k - j - 1] - '0') * shift;
```

```
                shift *= 10;
            }
            shift = 10;
            k += NUM_DIGIT;
        }
        if (n = len - n * NUM_DIGIT)
        {
            a[i] = (value[len - k - 1] - '0');
            for (j = 1; j < n; j++)
            {
                a[i] += (value[len - k - j - 1] - '0') * shift;
                shift *= 10;
            }
            i++;
        }
        return i;
    }
    /// <summary>
    /// 通过无符号32整型初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]整型值</param>
    /// <returns>数据长度</returns>
    static size_t initByInt32(_NUM_T* a, uint32_t value)
    {
        size_t i;
        for (i = 0; i < 8096; i++)
        {
            a[i] = value % _hex;
            value /= _hex;
            if (!value)
            {
                i++;
                break;
            }
        }
        return i;
    }
    /// <summary>
    /// 通过无符号64整型初始化
    /// </summary>
    /// <param name="a">[in]高精度数组</param>
    /// <param name="value">[in]整型值</param>
    /// <returns>数据长度</returns>
    static size_t initByInt64(_NUM_T* a, uint64_t value)
    {
        size_t i;
        for (i = 0; i < 8096; i++)
        {
            a[i] = value % _hex;
            value /= _hex;
            if (!value)
            {
                i++;
                break;
```

```
            }
        }
        return i;
    }
    /// <summary>
    /// 比较两个高精度数的大小
    /// </summary>
    /// <param name="a">[in]第一个数</param>
    /// <param name="aLen">[in]第一个数的长度</param>
    /// <param name="b">[in]第二个数</param>
    /// <param name="bLen">[in]第二个数的长度</param>
    /// <returns>1是a>b,0是a==b,-1是a<b</returns>
    static int compare(const _NUM_T* a, size_t aLen, const  _NUM_T* b, size_t bLen)
    {
        if (aLen > bLen)
        {
            return 1;
        }
        else if (aLen < bLen)
        {
            return -1;
        }
        else {
            for (int i = aLen - 1; i > -1; i--)
            {
                if (a[i] > b[i])
                {
                    return 1;
                }
                else if (a[i] < b[i])
                {
                    return -1;
                }
            }
        }
        return 0;
    }
    /// <summary>
    /// 打印输出结果
    /// </summary>
    static void print(const _NUM_T* a, size_t aLen) {
        int i = aLen - 1, j = 0, n = _hex;
        char format[32];
        sprintf(format, "%%0%dd", NUM_DIGIT);
        printf("%d", a[i--]);
        for (; i > -1; i--)
            printf(format, a[i]);
    }

    /// <summary>
    /// 加法(累加)
    ///结果会保存在a中
    /// </summary>
    /// <param name="a">[in]被加数</param>
```

```
152     /// <param name="aLen">[in]被加数长度</param>
153     /// <param name="b">[in]加数</param>
154     /// <param name="bLen">[in]加数长度</param>
155     /// <returns>结果长度</returns>
156     static  size_t acc(_NUM_T* a, size_t aLen, const _NUM_T* b, size_t
bLen)
157     {
158         size_t i, n = bLen;
159         const size_t max = _hex - 1;
160         if (aLen < bLen)
161         {
162             memset(a + aLen, 0, (bLen - aLen + 1) * sizeof(_NUM_T));
163         }
164         else {
165             a[aLen] = 0;
166         }
167         for (i = 0; i < bLen; i++) {
168             uint64_t temp = (uint64_t)a[i] + (uint64_t)b[i];
169             a[i] = temp % _hex;
170             a[i + 1] += temp / _hex;
171         }
172         for (; i < aLen; i++) {
173             uint64_t temp = a[i];
174             if (temp <= max)
175                 break;
176             a[i] = temp % _hex;
177             a[i + 1] += temp / _hex;
178         }
179         n = aLen > bLen ? aLen : bLen;
180         if (a[n])
181             return n + 1;
182         return n;
183     }
184
185 /// <summary>
186     /// 减法(累减)
187     ///结果会保存在a中
188     /// </summary>
189     /// <param name="a">[in]被减数，被减数必须大于等于减数</param>
190     /// <param name="aLen">[in]被减数长度</param>
191     /// <param name="b">[in]减数</param>
192     /// <param name="bLen">[in]减数长度</param>
193     /// <returns>结果长度</returns>
194     static  size_t subc(_NUM_T* a, size_t aLen, const _NUM_T* b, size_t
bLen) {
195         size_t  m, n, i;
196         if (aLen < bLen)
197         {
198             m = bLen;
199             n = aLen;
200         }
201         else {
202             n = bLen;
203             m = aLen;
204         }
```

```
        for (i = 0; i < n; i++)
        {
            int64_t temp = (int64_t)a[i] - (int64_t)b[i];
            a[i] = temp;
            if (temp < 0)
            {
                a[i + 1] -= 1;
                a[i] += _hex;
            }
        }
        for (; i < m; i++)
        {
            _NUM_T temp = a[i];
            if (temp < 0)
            {
                a[i + 1] -= 1;
                a[i] += _hex;
            }
            else
            {
                break;
            }
        }
        for (size_t i = aLen - 1; i != SIZE_MAX; i--)
            if (a[i])
                return i + 1;
        return 1;
    }

/// <summary>
    /// 乘法
    /// </summary>
    /// <param name="a">[in]被乘数</param>
    /// <param name="aLen">[in]被乘数长度</param>
    /// <param name="b">[in]乘数</param>
    /// <param name="bLen">[in]乘数长度</param>
    /// <param name="c">[out]结果，数组长度必须大于等于aLen+bLen+1，且全部元素为0</param>
    /// <returns>结果长度</returns>
    static  size_t multi(const _NUM_T* a, size_t aLen, const  _NUM_T* b, size_t bLen, _NUM_T c[]) {
        _NUM_T  d;
        size_t j;
        c[aLen + bLen] = 0;
        for (size_t i = 0; i < aLen; i++)
        {
            d = 0;
            for (j = 0; j < bLen; j++)
            {
                uint64_t temp = (uint64_t)a[i] * (uint64_t)b[j] + c[j + i] + d;
                d = temp / _hex;
                c[j + i] = temp % _hex;
            }
            if (d)
```

```
            {
                c[j + i] = d;
            }
        }
        for (size_t k = aLen + bLen; k != SIZE_MAX; k--)
            if (c[k])
                return k + 1;
        return 1;
    }

/// <summary>
    /// 除法
    /// </summary>
    /// <param name="a">[in]被除数,被除数必须大于除数。小于商为0、余数为除数，等于商为1、余数为0，不需要在函数内实现，在外部通过compare就可以做到。</param>
    /// <param name="aLen">[in]被除数长度</param>
    /// <param name="b">[in]除数</param>
    /// <param name="bLen">[in]除数长度</param>
    /// <param name="c">[out]商，数组长度大于等于aLen，且全部元素为0</param>
    /// <param name="mod">[out]余数，数组长度大于等于aLen</param>
    /// <param name="modLen">[out]余数长度</param>
    /// <param name="temp">[in]临时缓冲区，由外部提供以提高性能，数组长度大于等于aLen+bLen+1</param>
    /// <returns>商长度</returns>
        static  size_t divide(const _NUM_T* a, size_t aLen, const  _NUM_T* b, size_t bLen, _NUM_T* c, _NUM_T* mod, size_t* modLen, _NUM_T* temp) {
        intptr_t  n, l;
        int equal;
        memcpy(mod, a, (aLen) * sizeof(_NUM_T));
        n = aLen - bLen;
        l = aLen;
        while (n > -1)
        {
            equal = compare(mod + n, l - n, b, bLen);
            if (equal == -1)
            {
                n--;
                if (!mod[l - 1])
                    l--;
                continue;
            }
            if (equal == 0)
            {
                c[n]++;
                n -= bLen;
                l -= bLen;
                continue;
            }
            uint64_t x;
            if ((l - n) > bLen)
            {
                x = ((mod[l - 1]) * _hex) / ((uint64_t)b[bLen - 1] + 1);
                if (x == 0)
                {
                    x = (mod[l - 2]) / ((uint64_t)b[bLen - 1] + 1);
```

```
                }
            }
            else
            {
                x = (mod[l - 1]) / ((uint64_t)b[bLen - 1] + 1);
            }
            if (x == 0)
                x = 1;
            c[n] += x;
            if (x == 1)
            {
                l = n + subc(mod + n, l - n, b, bLen);
            }
            else
            {
                _NUM_T num[_MIN_NUM_SIZE];
                size_t len = initByInt32(num, x);
                memset(temp, 0, (aLen + bLen + 1) * sizeof(_NUM_T));
                len = multi(b, bLen, num, len, temp);
                l = n + subc(mod + n, l - n, temp, len);
            }
        }
        intptr_t i = l - 1;
        for (; i > -1; i--)
            if (mod[i])
                break;
        if (i == -1)
        {
            mod[0] = 0;
            *modLen = 1;
        }
        else
        {
            *modLen = i + 1;
        }
        if (c[aLen - bLen] != 0)
            return aLen - bLen + 1;
        else
            return aLen - bLen;
    }

int sum(){
    int64_t n;
    size_t len, len2;
    _NUM_T num[1024];
    _NUM_T num2[1024];
    scanf("%ld", &n);
    len = initByInt32(num, 0);
    for (int64_t i = 1; i <= n; i++)
    {
        len2 = initByInt64(num2, i);
        len = acc(num, len, num2, len2);
    }
    print(num, len);
    return 0;
```

```
}

int minus()
{
    _NUM_T a1[8096], a2[8096];
    char s1[8096], s2[8096];
    scanf("%s%s", s1, s2);
    size_t len1,len2;
    len1 =initByStr(a1, s1, strlen(s1));
    len2 = initByStr(a2, s2, strlen(s2));
    len1 =subc(a1, len1, a2, len2);
    print(a1,len1);
    return 0;
}

int factorial(){
    int64_t n;
    size_t len, len2;
    _NUM_T num[8096];
    _NUM_T num2[8096];
    _NUM_T num3[8096];
    _NUM_T* p1 = num;
    _NUM_T* p2 = num3;
    scanf("%ld", &n);
    len = initByInt32(num,1);
    for (int64_t i = 1; i <= n; i++)
    {
        len2 = initByInt64(num2, i);
        memset(p2, 0, 8096* sizeof(_NUM_T));
        len = multi(p1, len, num2, len2, p2);
        _NUM_T* temp = p1;
        p1 = p2;
        p2 = temp;
    }
    print(p1, len);
    return 0;
}

int division()
{
    _NUM_T a1[8096], a2[8096],c[8096],mod[8096], temp[8096];
    char s1[8096], s2[8096];
    scanf("%s%s", s1, s2);
    size_t len1,len2,ml;
    len1 =initByStr(a1, s1, strlen(s1));
    len2 = initByStr(a2, s2, strlen(s2));
    memset(c, 0, 8096 * sizeof(_NUM_T));
    len1 =divide(a1, len1, a2, len2,c, mod,&ml, temp);
    print(c,len1);
    printf("\n");
    print(mod, ml);
    return 0;
}

int main() {
```

```
    printf("求 n 累加和(ja)。用高精度方法，求 s=1+2+3+……+n 的精确值(n 以一般整数输入)。\n");
    sum();
    printf("\n两个任意十一位数的减法；（小于二十位）\n");
    minus();
    printf("\n求 N! 的值(ni)。用高精度方法，求 N! 的精确值(N 以一般整数输入)。\n");
    factorial();
    printf("\n给定两个非负整数A，B，请你计算 A / B的商和余数。\n");
    division();
}

```

./library/大整数计算2.c

```c
// https://www.cnblogs.com/HappyXie/articles/1927557.html

#include <stdio.h>
#include <stdlib.h>
#define MAX 200

typedef struct
{
    int len;
    int s[MAX+1];
} hp;

void input(hp *a, int x) //读入数字
{
    int i;

    a->len = 0;

    while (x > 0)
    {
        a->s[1 + a->len++] = x % 10;
        x /= 10;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void input1(hp *a, char *str) //读入字符串
{
    int i, len;

    a->len = 0;

    if (str == NULL)
        return;

    len = strlen(str);

    while (len > 0)
```

```
    {
        a->s[1 + a->len++] = *(str + len - 1) - '0';
        len--;
    }

    for (i = a->len + 1; i <= MAX; i++)
        a->s[i] = 0;
}

void print(hp *y) //打印数字
{
    int i;
    for (i = y->len; i >= 1; i--)
        printf("%d", y->s[i]);
    printf("\n");
}

void add(hp *a, hp *b, hp *c) //高精度加法c = a + b
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] + b->s[i];
        if (c->s[i] >= 10)
        {
            c->s[i] -= 10;
            c->s[i+1]++;
        }
    }

    if (c->s[len+1] > 0) len++;
    c->len = len;
}

void subtract(hp *a, hp *b, hp *c) //高精度减法c = a - b
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] - b->s[i];
        if (c->s[i] < 0)
        {
            c->s[i] += 10;
```

```
            c->s[i+1]--;
        }
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

int compare(hp *a, hp *b) //高精度比较
{
    int len;

    if (a->len > b->len) len = a->len;
    else len = b->len;

    while (len > 0 && a->s[len] == b->s[len]) len--;

    if (len == 0) return 0;
    else return a->s[len] - b->s[len];
}

void multiply(hp *a, int b, hp *c) //高精度 * 单精度
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;
    len = a->len;

    for (i = 1; i <= len; i++)
    {
        c->s[i] += a->s[i] * b;
        c->s[i+1] += c->s[i] / 10;
        c->s[i] %= 10;
    }

    len++;
    while (c->s[len] >= 10)
    {
        c->s[len+1] += c->s[len] / 10;
        c->s[len] %= 10;
        len++;
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void multiplyh(hp *a, hp *b, hp *c) //高精度 * 高精度
{
    int i, j, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;

    for (i = 1; i <= a->len; i++)
    {
```

```
        for (j = 1; j <= b->len; j++)
        {
            c->s[i+j-1] += a->s[i] * b->s[j];
            c->s[i+j] += c->s[i+j-1] / 10;
            c->s[i+j-1] %= 10;
        }
    }

    len = a->len + b->len + 1;
    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

void power(hp *a, int b, hp *c) //高精度乘方c = a ^ b
{
    hp e;

    if (b == 0)
    {
        c->len = 1;
        c->s[1] = 1;
    }
    else if (b == 1)
    {
        memcpy(c, a, sizeof(hp));
    }
    else
    {
        power(a, b / 2, &e);
        multiplyh(&e, &e, c);

        if (b % 2 == 1)
        {
            memcpy(&e, c, sizeof(hp));
            multiplyh(&e, a, c);
        }
    }
}

void divide(hp *a, int b, hp *c, int *d) //高精度 / 单精度 {d为余数}
{
    int i, len;

    for (i = 1; i <= MAX; i++) c->s[i] = 0;
    len = a->len;
    *d = 0;

    for (i = len; i >= 1; i--)
    {
        *d = *d * 10 + a->s[i];
        c->s[i] = *d / b;
        *d %= b;
    }

    while (len > 1 && c->s[len] == 0) len--;
```

```
    c->len = len;
}

void multiply10(hp *a) //高精度 * 10
{
    int i;
    for (i = a->len; i >= 1; i--)
        a->s[i+1] = a->s[i];

    a->s[1] = 0;
    a->len++;
    while (a->len > 1 && a->s[a->len] == 0) a->len--;
}

void divideh(hp *a, hp *b, hp *c, hp *d) //高精度 / 高精度{d为余数}
{
    hp e;
    int i, len;

    for (i = 1; i <= MAX; i++)
    {
        c->s[i] = 0;
        d->s[i] = 0;
    }

    len = a->len;
    d->len = 1;

    for (i = len; i >= 1; i--)
    {
        multiply10(d);
        d->s[1] = a->s[i];

        while (compare(d, b) >= 0)
        {
            subtract(d, b, &e);
            *d = e;
            c->s[i]++;
        }
    }

    while (len > 1 && c->s[len] == 0) len--;
    c->len = len;
}

```

./他人做法/1009.QR Code.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
int main()
{
```

```
    int i,j,c,d,a,b;char s[501],t[33333],u[3];
    scanf("%s",s);
    c=13;
    int len = strlen(s);
    for(i=0;i<len;c=c+10)
    {
        if(i+3>len) break;
        for(j=0;j<3;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+10;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';

    }
    b=strlen(s);
    if((b-i)==2)
    {
        for(j=0;j<2;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+7;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';
        t[c+8]='\0';
    }
    else if((b-i)==1)
    {
        for(j=0;j<1;j++,i++)
        {
            u[j]=s[i];
        }
        u[j]='\0';
        a=atoi(u);
        for(d=c+4;a!=0;d--)
        {
            t[d]=a%2+'0';a/=2;
        }
        for(;d>c;d--) t[d]='0';
        t[c+5]='\0';
    }
    else
    {
        t[c+1]='\0';
    }
```

```
    for(d=13;b!=0;d--)
    {
        t[d]=b%2+'0';b/=2;
    }
    for(;d>3;d--) t[d]='0';
    t[0]='0';t[1]='0';t[2]='0';t[3]='1';
    printf("%s",t);
    return 0;
}

```

./他人做法/1064.A-B.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
void Input(int a[])
{
    char s[502];
    scanf("%s", s);
    int i = 0, j, k;
    j = strlen(s) - 1;
    for (k = 501; j >= 0; j--)
    {
        a[k] = s[j] - '0';
        k--;
    }
}
int main()
{
    while (1)
    {
        int a[502] = {0};
        int b[502] = {0};
        Input(a);
        Input(b);
        int m = 0, n = 0;
        int j = 0;
        while (a[m] == 0)
        {
            m++;
        }
        while (b[n] == 0)
        {
            n++;
        }
        if (n > m)
        {
            for (int i = 501; i >= m; i--)
            {
                if (a[i] - b[i] >= 0)
                {
                    a[i] = a[i] - b[i];
```

```
                }
                else
                {
                    a[i] = a[i] + 10 - b[i];
                    a[i - 1]--;
                }
            }
            while (a[j] == 0)
            {
                j++;
            }
            for (j; j <= 501; j++)
            {
                printf("%d", a[j]);
            }
            printf("\n");
        }

        if (m > n)
        {
            for (int i = 501; i >= n; i--)
            {
                if (b[i] - b[i] >= 0)
                {
                    b[i] = b[i] - a[i];
                }
                else
                {
                    b[i] = b[i] + 10 - a[i];
                    b[i - 1]--;
                }
            }
            while (b[j] == 0)
            {
                j++;
            }
            printf("-");
            for (j; j <= 501; j++)
            {
                printf("%d", b[j]);
            }
            printf("\n");
        }
        else
        {
            if (a[m] >= b[n])
            {
                for (int i = 501; i >= 0; i--)
                {
                    if (a[i] - b[i] >= 0)
                    {
                        a[i] = a[i] - b[i];
                    }
                    else
                    {
```

```
                        a[i] = a[i] + 10 - b[i];
                        a[i - 1]--;
                    }
                }
                while (a[j] == 0)
                {
                    j++;
                }
                for (j; j <= 501; j++)
                {
                    printf("%d", a[j]);
                }
                printf("\n");
            }
            else
            {
                for (int i = 501; i >= 0; i--)
                {
                    if (b[i] - b[i] >= 0)
                    {
                        b[i] = b[i] - a[i];
                    }
                    else
                    {
                        b[i] = b[i] + 10 - a[i];
                        b[i - 1]--;
                    }
                }
                while (b[j] == 0)
                {
                    j++;
                }
                printf("-");
                for (j; j <= 501; j++)
                {
                    printf("%d", b[j]);
                }
                printf("\n");
            }
        }
    }

    return 0;
}


```

./合并文件.py

```
# 21:50
import glob
import os

os.chdir("/config/workspace/学校文件/编程思维与实践")
```

```

with open("合并后.md", "w", encoding='utf-8') as f:
    files = glob.glob("./**", recursive=True)
    files.sort()
    for i in files:
        if not os.path.isfile(i):
            continue
        f.write("# ")
        print(i, file = f)
        print("```C", file=f)
        try:
            text = open(i, encoding='utf-8').read()
        except:
            text = open(i, encoding='gbk').read()
        f.write(text)
        print('\n', file=f)
        print("```", file=f)
        print('\n', file=f)

```

./正则表达式测试.c

```
#define _GNU_SOURCE
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <regex.h>

int main (void)
{

    int i;
    char ebuff[256];
    int ret;
    int cflags;
    regex_t reg;
    regmatch_t rm[5];
    char *part_str = NULL;

    cflags = REG_EXTENDED | REG_ICASE;

    char *test_str = "Hello World";
    char *reg_str = "e(.*)o";

    ret = regcomp(&reg, reg_str, cflags);
    if (ret)
    {
        regerror(ret, &reg, ebuff, 256);
        fprintf(stderr, "%s\n", ebuff);
        goto end;
    }

    ret = regexec(&reg, test_str, 5, rm, 0);
```

```
    if (ret)
    {
        regerror(ret, &reg, ebuff, 256);
        fprintf(stderr, "%s\n", ebuff);
        goto end;
    }

    regerror(ret, &reg, ebuff, 256);
    fprintf(stderr, "result is:\n%s\n\n", ebuff);

    for (i=0; i<5; i++)
    {
        if (rm[i].rm_so > -1)
        {
            part_str = strndup(test_str+rm[i].rm_so, rm[i].rm_eo-
rm[i].rm_so);
            fprintf(stderr, "%s\n", part_str);
            free(part_str);
            part_str = NULL;
        }
    }

end:
    regfree(&reg);

    return 0;
}

```