排序

一、简单选择排序——O(n^2)

对待排序序列进行 **n 趟操作**，每一趟都找到**最小值对应的下标**，并与待排序部分的**第一个元素**进行交换。

```
void SelectSort(){
    for(int i=0;i<n;i++){
        int k=i;
        for(int j=i;j<n;j++){           //选出待排序部分中的最小值
            if(a[j] < a[k]) k=j;
        }
        int temp = a[i];                //交换最小值与待排序部分的第一个值
        a[i] = a[k];
        a[k] = temp;
    }
}
```

二、插入排序——O(n^2)

直接将待排序元素插入到有序部分的对应位置。

```
void InsertSort(){                      //从后往前枚举已有序部分
    for(int i=1;i<n;i++){
        int temp = a[i], j = i;
        while(j>1 && temp<a[j-1]){      //若前一个数字更大，将其后移一位
            a[j] = a[j-1];
            j--;
        }
        a[j] = temp;
    }
}
```

三、计数排序——O(n)

```
void CountSort(int array[]) {
    //1.得到数列的最大值与最小值，并算出差值d
    int max = array[0], min = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] > max) {
```

```
            max = array[i];
        }
        if(array[i] < min) {
            min = array[i];
        }
    }
    int d = max - min;

    //2.创建基于差值长度的统计数组并统计填充对应元素个数
    int countArray[d+1];
    int len = sizeof(array) / sizeof(array[0]);
    for (int i = 0; i < len; i++) {
        countArray[array[i] - min]++;
    }

    //3.统计数组变形，后面的元素等于前面的元素之和
    for (int i = 0; i < d+1; i++) {
        countArray[i] = countArray[i] + countArray[i-1];
    }

    //4.倒序遍历原始数组，从统计数组找到正确位置，输出到结果数组
    int sortedArray[len];
    for (int i = len - 1; i >=0; --i) {
        //index 反映的是：array[i] 在结果数组中的下标
        int index = countArray[array[i] - min] -1;
        sortedArray[index] = array[i];
        //计数数组相应位置减1（已经取出一个数了）
        countArray[array[i] - min]--;
    }
    return sortedArray;
}
```

四、排序题与Sort函数的应用

1、如何使用 sort 函数：

（1）头文件：

```
#include <algorithm>
using namespace std;
```

（2）sort函数使用方式：

`sort(首元素地址(必填)，尾元素的下一个地址(必填)，比较函数(选填))` 若不写比较函数，则默认对前面给出的区间进行**递增排序。**

2、如何实现比较函数：

（1）基本数据类型数组的排序：

① 不填比较函数，默认从小到大排序；

②若想要从大到小排序，构造比较函数如下：

```
bool Compare(int a, int b){
    return a>b;          //可以理解为当 a>b 时，将 a 放在 b 的前面
}
// 也可以先从小到大排序，然后使用 reverse 函数
```

（2）结构体数组的排序：

定义结构体如下：

```
struct node{
    int x,y;
}ssd[10];
```

①按照 x 从大到小进行排序；

②若 x 相等，则按照 y 从大到小进行排序；

构建比较函数如下：

```
bool Compare(node a, node b){
    if(a.x != b.x) return a.x > b.x;
    else return a.y > b.y;
}
```

（3）容器的排序：

在 STL 标准容器中，只有 vector、string、deque 是可以使用 sort 的。

这是因为 set、map 这些容器是用红黑树实现的，元素本身有序，故不允许使用 sort 排序。

3、如何在排序题中使用 sort 函数：

（1）定义相关结构体；

（2）定义比较函数：实际上是确定排序的规则；

（3）排名的实现：在完成排序的基础上，实现有关的需求；

五、甲类习题：

1、The Best Rank

To evaluate the performance of our first year CS majored students, we consider their grades of three courses only: `C` - C Programming Language, `M` - Mathematics (Calculus or Linear Algrbra), and `E` - English.

At the mean time, we encourage students by emphasizing on their best ranks -- that is, among the four ranks with respect to the three courses and the average grade, we **print the best rank** for each student.

For example, The grades of `C`, `M`, `E` and `A` - Average of 4 students are given as the following:

```
StudentID  C  M  E  A
310101     98 85 88 90
310102     70 95 88 84
310103     82 87 94 88
310104     91 91 91 91
```

Input Specification:

Each input file contains one test case.

Each case starts with a line containing 2 numbers *N* and *M* (≤2000), which are **the total number of students**, and **the number of students who would be checked** their ranks, respectively.

Then *N* lines follow, each contains a student ID which is a string of 6 digits, followed by the three integer grades (in the range of [0, 100]) of that student in the order of `C`, `M` and `E`.

Then there are *M* lines, each containing a student ID.

Output Specification:

For each of the *M* students, print **the best rank for him/her**, and **the symbol of the corresponding rank**, in one line, separated by a space.

The priorities of the ranking methods are ordered as `A` **>** `C` **>** `M` **>** `E`. Hence if there are two or more courses for a student to obtain the same best rank, output the one with the highest priority.

If a student is not on the grading list, simply output `N/A`.

Sample Input:

```
5 6
310101 98 85 88
310102 70 95 88
310103 82 87 94
310104 91 91 91
310105 85 90 90
310101
310102
310103
310104
310105
999999
```

Sample Output:

```
1 C
1 M
1 E
1 A
3 A
N/A
```

Key Codes:

- [ ] 首先要求出各个学生的平均成绩（向下取整即可）；
- [ ] 其次，按照 `A` > `C` > `M` > `E` 的顺序，依次进行排序；
- [ ] 在每一次排序的过程中，注意：相同分数的同学，具有相同的排名；
- [ ] 使用一个数组 grade 用于记录该生在对应学科的排名；

```
struct Student{
    char id[10];
    int c, m, e, a;
    int grade[4];
};
bool cmpC(Student a, Student b){
    return a.c > b.c;
}
bool cmpM(Student a, Student b){
    return a.m > b.m;
}
bool cmpE(Student a, Student b){
    return a.e > b.e;
}
bool cmpA(Student a, Student b){
    return a.a > b.a;
}
void Output(Student student){
    int min=4;
    for(int i=0;i<4;i++){
```

```
        if(student.grade[i] < min) min = student.grade[i];
    }
    if(student.grade[0] == min) printf("%d A\n", min);
    else if(student.grade[1] == min) printf("%d C\n", min);
    else if(student.grade[2] == min) printf("%d M\n", min);
    else if(student.grade[3] == min) printf("%d E\n", min);
}
int main(){
    int N, M;
    scanf("%d %d", &N, &M);

    Student student[N], query[M];
    for(int i=0;i<N;i++){
        scanf("%s ", student[i].id);
        scanf("%d%d%d", &student[i].c, &student[i].m, &student[i].e);
        student[i].a = (student[i].c + student[i].m + student[i].e)/3;
    }
    sort(student, student+N, cmpA);
    student[0].grade[0] = 1;
    for(int i=1;i<N;i++){
        if(student[i].a == student[i-1].a){
            student[i].grade[0] = student[i-1].grade[0];
        }
        else student[i].grade[0] = i+1;
    }
    sort(student, student+N, cmpC);
    student[0].grade[1] = 1;
    for(int i=0;i<N;i++){
        if(student[i].c == student[i-1].c){
            student[i].grade[1] = student[i-1].grade[1];
        }
        else student[i].grade[1] = i+1;
    }
    sort(student, student+N, cmpM);
    student[0].grade[2] = 1;
    for(int i=0;i<N;i++){
        if(student[i].m == student[i-1].m){
            student[i].grade[2] = student[i-1].grade[2];
        }
        else student[i].grade[2] = i+1;
    }
    sort(student, student+N, cmpE);
    student[0].grade[3] = 1;
    for(int i=0;i<N;i++){
        if(student[i].e == student[i-1].e){
            student[i].grade[3] = student[i-1].grade[3];
        }
        else student[i].grade[3] = i+1;
    }

    for(int i=0;i<M;i++){
        scanf("%s", query[i].id);
```

```
        bool flag = false;
        for(int j=0;j<N;j++){
            if(strcmp(query[i].id, student[j].id) == 0){
                Output(student[j]); flag = true;
            }
        }
        if(flag == false){printf("N/A\n");}
    }
    return 0;
}
```

2、Phone Bills

A long-distance telephone company charges its customers by the following rules:

- Making a long-distance call costs a certain amount per minute, **depending on the time of day** when the call is made. （按分钟收费）
- When a customer starts connecting a long-distance call, the time will be recorded, and so will be the time when the customer **hangs up** the phone. （开始和结束的时间会被记录）
- Every **calendar month**, a bill is sent to the customer for each minute called (at a rate determined by the time of day). （每个月生成一份账单）
- Your job is to prepare the bills for each month, given a set of phone call records.

Input Specification:

Each input file contains one test case. Each case has two parts: the rate structure, and the phone call records.

- The rate structure consists of a line with **24** non-negative integers denoting the toll **(cents/minute)** from 00:00 - 01:00, the toll from 01:00 - 02:00, and so on for each hour in the day. （收费标准在每个小时的区间内都不一样）
- The next line contains a positive number *N* (≤1000), followed by ***N* lines of records**.
- Each phone call record consists of the **name** of the customer (string of up to 20 characters without space), the **time and date** (`MM:dd:HH:mm`), and the word `on-line` **or** `off-line`. （给出每个记录的人名、日期、时间、通话状态）

For each test case, all dates will be within **a single month**. （所有记录都在同一个月）

Each `on-line` record is **paired with** the **chronologically** next record for the same customer provided it is an `off-line` record. （在按照时间排序后，两条配对 on-line 和 off-line 对应时间内不允许出现其他 on-line 和 off-line 的记录。）Any `on-line` records that are not paired with an `off-line` record are **ignored**, **as are** `off-line` records not paired with an `on-line` record.

It is guaranteed that at least one call is well paired in the input. You may assume that **no two records for the same customer have the same time**. Times are recorded using a 24-hour clock.

Output Specification:

For each test case, you must print a phone bill for each customer.

Bills must be printed in **alphabetical order** of customers' names. For each customer:

- First print in a line the **name** of the customer and the **month** of the bill in the format shown by the sample.
- Then for each time period of a call, print in one line **the beginning and ending time and date** (`dd:HH:mm`), **the lasting time** (in minute) and **the charge of the call.** The calls must be listed in **chronological order**.
- Finally, print the **total charge** for the month in the format shown by the sample.

Sample Input:

```
10 10 10 10 10 10 20 20 20 15 15 15 15 15 15 15 20 30 20 15 15 10 10 10
10
CYLL 01:01:06:01 on-line
CYLL 01:28:16:05 off-line
CYJJ 01:01:07:00 off-line
CYLL 01:01:08:03 off-line
CYJJ 01:01:05:59 on-line
aaa 01:01:01:03 on-line
aaa 01:02:00:01 on-line
CYLL 01:28:15:41 on-line
aaa 01:05:02:24 on-line
aaa 01:04:23:59 off-line
```

Sample Output:

```
CYJJ 01
01:05:59 01:07:00 61 $12.10
Total amount: $12.10
CYLL 01
01:06:01 01:08:03 122 $24.40
28:15:41 28:16:05 24 $3.85
Total amount: $28.25
aaa 01
02:00:01 04:23:59 4318 $638.80
Total amount: $638.80
```

Key Codes: 19/25

- [ ] 通话要匹配，要计算总时长；
- [ ] 通话费用根据各个小时的不同而产生差异；
- [ ] 输出姓名（ASCII码顺序）、月份、通话开始时间、结束时间、持续时间、费用、总费用；
- [ ] 题目保证至少有一对有效的通话记录，但不保证每个用户都有有效的通话记录，因此不存在有效通话记录的用户不能输出；

```
int feelist[24];
```

```
struct Customer{
    char name[25], time[15];
    int status = -1;
};
struct Bill{
    char name[25], Stime[15], Etime[15];
    int lastime = 0;
    double cost;
};

bool cmp(Customer a, Customer b){
    if(strcmp(a.name, b.name) != 0) return strcmp(a.name, b.name) < 0;
    else return strcmp(a.time, b.time) < 0;
}
//计算一则通话记录的通话时间及其所需要的费用
void CountAndCost(Bill a, int &lastime, double &cost){
    int month, day, hour, min;
    char time[25];
    sscanf(a.Stime, "%02d:%02d:%02d:%02d", &month, &day, &hour, &min);
    strcpy(time, a.Stime);

    while(strcmp(time, a.Etime) < 0){
        if(min < 60){
            min++; lastime++;
            cost += feelist[hour];
        }
        else {min = 0; hour++;}
        if(hour >= 24) {hour = 0; day++;}
        sprintf(time, "%02d:%02d:%02d:%02d", month, day, hour, min);
    }
    cost = cost / 100;
}
void Output(Bill a){    //输出通话时间记录、时长及费用
    for(int i=3;i<11;i++){printf("%c", a.Stime[i]);}
    printf(" ");
    for(int i=3;i<11;i++){printf("%c", a.Etime[i]);}
    printf(" ");

    printf("%d $%.2f\n", a.lastime, a.cost);
}

int main(){
    /*第一部分：测试点的输入*/
    int N;
    for(int i=0;i<24;i++){scanf("%d", &feelist[i]);}
    scanf("%d", &N);
    vector<Customer> cus(N);
    for(int i=0;i<N;i++){
        char s[10];
        scanf("%s %s %s", cus[i].name, cus[i].time, s);
        if(strcmp(s, "on-line") == 0) cus[i].status = 1;
        else cus[i].status = 0;
```

```
    }

    /*第二部分：对记录进行排序，首先按字母升序排序，其次按时间升序排序*/
    sort(cus.begin(), cus.end(), cmp);

    /*第三部分：遍历记录，找到配对的记录，并存放至 cumlist 中*/
    vector<Bill> cumlist;
    Bill cum;
    for(int i=0;i<cus.size();i++){
        for(int j=i+1 ; j<cus.size() ; j++){
            if(strcmp(cus[i].name, cus[j].name) == 0){
                if(cus[i].status == 1 && cus[j].status ==0){
                    strcpy(cum.name, cus[i].name);
                    strcpy(cum.Stime, cus[i].time);
                    strcpy(cum.Etime, cus[j].time);
                    cumlist.push_back(cum); break;
                }
                else break;
            }
            else break;
        }
    }

    /*第四部分：根据 cumlist 中的记录来计算通话时长和费用*/
    char name[25];
    double total = 0;
    strcpy(name, cumlist[0].name);
    printf("%s %c%c\n", name, cumlist[0].Stime[0], cumlist[0].Stime[1]);
    for(int i=0;i<cumlist.size();){
        if(strcmp(name, cumlist[i].name) == 0){
            //计算一则通话记录的通话时间及其所需要的费用
            CountAndCost(cumlist[i], cumlist[i].lastime, cumlist[i].cost);
            //输出通话时间记录、时长及费用
            Output(cumlist[i]);
            total += cumlist[i].cost;
            i++;
        }
        else{
            printf("Total amount: $%.2f\n", total);
            total = 0;
            strcpy(name, cumlist[i].name);
            printf("%s ",name);
            printf("%c %c\n",cumlist[i].Stime[0],cumlist[i].Stime[1]);
        }
    }
    printf("Total amount: $%.2f\n", total);
    return 0;
}
```

Code Modify： 25/25

通过对代码的复盘，可以发现问题出现在第三部分：记录配对。

```
/*第三部分：遍历记录，找到配对的记录，并存放至 cumlist 中*/
vector<Bill> cumlist;
Bill cum;
for(int i=0;i<cus.size()-1;i++){
    int j = i + 1;
    //只考虑同一用户的记录
    if((j<cus.size()) && strcmp(cus[i].name, cus[j].name) == 0){
        //on、off 是相邻的两条记录
        if(cus[i].status == 1 && cus[j].status ==0){
            strcpy(cum.name, cus[i].name);
            strcpy(cum.Stime, cus[i].time); strcpy(cum.Etime, cus[j].time);
            cumlist.push_back(cum);
        }
    }
}
```

上面的代码是将记录配对与费用、时长计算分开来执行的，我们可以考虑整合这两部分代码：

```
/*第三部分：对通话记录进行配对，计算通话时长和费用，并进行输出*/
int on=0, off, next;
while(on < N){
    int needPrint = 0;
    next = on;
    while((next<N) && strcmp(cus[next].name, cus[on].name) == 0){
        //找到 on-line，置 needPrint 为 1
        if(needPrint == 0 && cus[next].status == 1) {needPrint = 1;}
        //找到 off-line，置 needPrint 为 2
        else if(needPrint == 1 && cus[next].status == 0) {needPrint = 2;}
        next++;                  //next 自增，直至找到下一个用户；
    }
    if(needPrint < 2) {on = next; continue;}
    double total = 0;            //总共花费的钱
    printf("%s %c%c\n", cus[on].name, cus[on].time[0], cus[on].time[1]);
    while(on < next){            //找到该用户的所有配对记录
        while(on < next - 1){    //next 指向的是下一个用户的第一条记录
            if(!(cus[on].status == 1 && cus[on+1].status == 0)) { on++; }
            else break;
        }
        off = on + 1;
        //已经输出完该用户所有配对记录
        if(off == next) { on = next; break; }
        OutputTime(cus[on].time, cus[off].time);
        int time = 0;            //单次通话的时长、费用
        double cost = 0;
        //计算本次通话的时长及费用；
        TimeAndCost(cus[on], cus[off], time, cost);
        total += cost;
```

```
            printf("%d $%.2f\n", time, cost);
            on = off + 1;
        }
        printf("Total amount: $%.2f\n", total);
    }
```

Code Optimize: 学完map后可进一步优化代码

3、PAT Ranking

Programming Ability Test (PAT) is organized by the College of Computer Science and Technology of Zhejiang University. Each test is supposed to run **simultaneously** in several places, and the **ranklists** will be **merged** immediately after the test. Now it is your job to write a program to correctly merge all the ranklists and generate the final rank.

Input Specification:

Each input file contains one test case. For each case：

- The first line contains a positive number *N* (≤100), **the number of test locations**. （N--考场数量）
- Then *N* ranklists follow, each starts with a line containing **a positive integer *K*** (≤300), the number of testees, and then ***K* lines** containing the registration number (a 13-digit number) and the total score of each testee. （K--考场人数）
- All the numbers in a line are separated by a space.

Output Specification:

- For each test case, first print in one line the **total number of testees**. （首先输出考试总人数）
- Then print the final ranklist in the following format：（然后输出排名）
  - 格式：`registration_number final_rank location_number local_rank`
  - The locations are numbered from 1 to *N*.
  - The output must be sorted in **nondecreasing order** of the final ranks. （总名次数值的升序序排序）
  - The testees with the **same score must have the same rank**, and the output must be sorted in **nondecreasing order of their registration numbers**.（同分同名，按考号升序排序）

Sample Input:

```
2
5
1234567890001 95
1234567890005 100
1234567890003 95
1234567890002 77
1234567890004 85
4
1234567890013 65
1234567890011 25
1234567890014 100
1234567890012 85
```

Sample Output:

```
9
1234567890005 1 1 1
1234567890014 1 2 1
1234567890001 3 1 2
1234567890003 3 1 2
1234567890004 5 1 4
1234567890012 5 2 2
1234567890002 7 1 5
1234567890013 8 2 3
1234567890011 9 2 4
```

Key Codes: 16/25

```
struct Student{
    char id[15];
    int score, location_number;
    int final_rank, local_rank;
};

bool cmp(Student a, Student b){
    if(a.score != b.score) return a.score > b.score;
    else return strcmp(a.id, b.id) < 0;
}

int main(){
    /*第一部分：输入*/
    int N, k=0;
    scanf("%d", &N);

    vector<Student> testees;
    for(int i=0;i<N;i++){              //输入每个考场的信息
        int K;
        scanf("%d", &K);
        vector<Student> testee(K);
        for(int j=0;j<K;j++){
```

```
            scanf("%s %d", testee[j].id, &testee[j].score); //输入
            testee[j].location_number = i+1;
        }
        sort(testee.begin(), testee.end(), cmp);            //排序
        for(int j=0;j<K; ){                                 //排名
            if(testee[j].score == testee[j+1].score){
                testee[j].local_rank = testee[j+1].local_rank = j+1;
                j = j+2;
            }else{
                testee[j].local_rank = j+1;
                j++;
            }
        }

        //插入总的数组中，并修改计数下标 k 的值
        testees.insert(testees.begin()+k, testee.begin(), testee.end());
        k += K;
    }
    sort(testees.begin(), testees.end(), cmp);
    for(int j=0;j<testees.size(); ){
        if(testees[j].score == testees[j+1].score){
            testees[j].final_rank = testees[j+1].final_rank = j+1;
            j = j+2;
        }else{
            testees[j].final_rank = j+1;
            j++;
        }
    }

    printf("%d\n", testees.size());
    for(int j=0;j<testees.size();j++){
        printf("%s %d %d %d\n", testees[j].id, testees[j].final_rank,
testees[j].location_number,testees[j].local_rank);
    }
    return 0;
}
```

Code Modify：25/25

通过对代码的复盘分析，我们发现原代码中的排名部分代码有一定的漏洞问题，因此修改如下：

```
testees[0].final_rank = 1;
printf("%s %d %d %d\n", testees[0].id, testees[0].final_rank,
testees[0].location_number, testees[0].local_rank);
for(int j=1;j<k;j++){        //针对排序结果进行排名
    if(testees[j].score != testees[j-1].score) testees[j].final_rank=j+1;
    else testees[j].final_rank = testees[j-1].final_rank;
    printf("%s %d %d %d\n", testees[j].id, testees[j].final_rank,
testees[j].location_number, testees[j].local_rank);
}
```

4、List Sorting

Excel can sort records according to any column. Now you are supposed to imitate this function.

Input Specification:

Each input file contains one test case. For each case:

- The first line contains two integers *N* (≤105) and *C*, where ***N* is the number of records** and ***C* is the column that you are supposed to sort the records with**.
- Then *N* lines follow, each contains a record of a student. A student's record consists of his or her distinct **ID** (a 6-digit number), **name** (a string with no more than 8 characters without space), and **grade** (an integer between 0 and 100, inclusive).

Output Specification:

For each test case, output the sorting result in *N* lines. That is:

- If *C* = 1 then the records must be sorted in **increasing order** according to ID's;
- If *C* = 2 then the records must be sorted in **non-decreasing order** according to names;
- If *C* = 3 then the records must be sorted in **non-decreasing order** according to grades.
- If there are several students who have the same name or grade, they must be sorted according to their ID's in **increasing order.**

Sample Input 1:

```
3 1
000007 James 85
000010 Amy 90
000001 Zoe 60
```

Sample Output 1:

```
000001 Zoe 60
000007 James 85
000010 Amy 90
```

Sample Input 2:

```
4 2
000007 James 85
000010 Amy 90
000001 Zoe 60
000002 James 98
```

Sample Output 2:

```
000010 Amy 90
000002 James 98
000007 James 85
000001 Zoe 60
```

Sample Input 3:

```
4 3
000007 James 85
000010 Amy 90
000001 Zoe 60
000002 James 9
```

Sample Output 3:

```
000002 James 9
000001 Zoe 60
000007 James 85
000010 Amy 90
```

Key Codes:

```
int n, c;
struct Student{
    char id[10];
    char name[10];
    int grade;
};
bool cmp(Student a, Student b){
    if(c == 1){
        return strcmp(a.id, b.id) < 0;
    }else if(c == 2){
        if(strcmp(a.name, b.name) == 0) return strcmp(a.id, b.id) < 0;
        else return strcmp(a.name, b.name) < 0;
    }else if(c == 3){
        if(a.grade == b.grade) return strcmp(a.id, b.id) < 0;
        else return a.grade < b.grade;
    }
}
int main(){
    scanf("%d %d", &n, &c);
    Student stu[n];
    for(int i=0;i<n;i++){
        scanf("%s %s %d", stu[i].id, stu[i].name, &stu[i].grade);
    }
    sort(stu, stu+n, cmp);
    for(int i=0;i<n;i++){
        printf("%s %s %d\n", stu[i].id, stu[i].name, stu[i].grade);
```

```
    }
    return 0;
}
```

5、The World's Richest

Forbes magazine publishes every year its list of billionaires based on the annual ranking of the world's wealthiest people. Now you are supposed to simulate this job, but concentrate only on the people in a certain range of ages. That is, **given the net worths of *N* people, you must find the *M* richest people in a given range of their ages.**

Input Specification:

Each input file contains one test case. For each case：

- The first line contains 2 positive integers: ***N* (≤105) - the total number of people**, and ***K* (≤103) - the number of queries**.
- Then *N* lines follow, each contains the **name** (string of no more than 8 characters without space), **age** (integer in (0, 200]), and the net **worth** (integer in [−106,106]) of a person.
- Finally there are *K* lines of queries, each contains three positive integers: ***M* (≤100) - the maximum number of outputs**, and [`Amin`, `Amax`] which are **the range of ages**.
- All the numbers in a line are separated by a space.

Output Specification:

For each query, first print in a line `Case #X:` where `X` is the query number starting from 1. Then output the *M* richest people with their ages in the range [`Amin`, `Amax`]. Each person's information occupies a line, in the format

```
Name Age Net_Worth
```

- The outputs must be in **non-increasing order of the net worths**.
- In case there are equal worths, it must be in **non-decreasing order of the ages**.
- If both worths and ages are the same, then the output must be in **non-decreasing alphabetical order of the names**.
- In case no one is found, output `None`.

It is guaranteed that there is no two persons share all the same of the three pieces of information.

Sample Input:

```
12 4
Zoe_Bill 35 2333
Bob_Volk 24 5888
Anny_Cin 95 999999
Williams 30 -22
Cindy 76 76000
Alice 18 88888
```

```
Joe_Mike 32 3222
Michael 5 300000
Rosemary 40 5888
Dobby 24 5888
Billy 24 5888
Nobody 5 0
4 15 45
4 30 35
4 5 95
1 45 50
```

Sample Output:

```
Case #1:
Alice 18 88888
Billy 24 5888
Bob_Volk 24 5888
Dobby 24 5888
Case #2:
Joe_Mike 32 3222
Zoe_Bill 35 2333
Williams 30 -22
Case #3:
Anny_Cin 95 999999
Michael 5 300000
Alice 18 88888
Cindy 76 76000
Case #4:
None
```

Key Codes:

```
struct Richers{
    char name[10];
    int age, worth;
};
bool cmp(Richers a, Richers b){
    if(a.worth != b.worth)
        return a.worth >= b.worth;
    else{
        if(a.age != b.age) return a.age <= b.age;
        else return strcmp(a.name, b.name) <= 0;
    }
}
int main(){
    int n, k;
    scanf("%d %d", &n, &k);
    Richers rich[n];
    for(int i=0;i<n;i++){
        scanf("%s %d %d", rich[i].name, &rich[i].age, &rich[i].worth);
    }
```

```
    sort(rich, rich+n, cmp);
    for(int i=0;i<k;i++){
        int m, agel, ageh, num=0;
        bool flag = false;
        scanf("%d %d %d", &m, &agel, &ageh);
        printf("Case #%d:\n", i+1);
        for(int j=0;j<n;j++){
            if(rich[j].age >= agel && rich[j].age <= ageh){
                printf("%s ", rich[j].age, rich[j].worth);
                printf("%d%d\n", rich[j].age, rich[j].worth);
                num++; flag = true;
            }
            if(num >= m) break;
        }
        if(flag == false) printf("None\n");
    }
    return 0;
}
```

6、PAT Judge

The ranklist of PAT is generated from the status list, which shows the scores of the **submissions**. This time you are supposed to generate the ranklist for PAT.

Input Specification:

Each input file contains one test case. For each case:

- The first line contains 3 positive integers, ***N* (≤10^4), the total number of users**, ***K* (≤5), the total number of problems**, and ***M* (≤10^5), the total number of submissions**. It is then assumed that the **user id**'s are 5-digit numbers from 00001 to *N*, and the **problem id**'s are from 1 to *K*. （N——总用户数，K——总问题数，M——总提交数）
- The next line contains *K* positive integers `p[i]` (`i`=1, ..., *K*), where `p[i]` corresponds to the **full mark of the i-th problem**. （各个问题的满分值）
- Then *M* lines follow, each gives the information of a submission in the following format: `user_id problem_id partial_score_obtained`
    - `partial_score_obtained` is **either** −1 if the submission **cannot even pass the compiler, or** is an integer in the range [0, `p[problem_id]`]. （得分为-1，即未通过编译）

All the numbers in a line are separated by a space.

Output Specification:

For each test case, you are supposed to output the ranklist in the following format:

```
rank user_id total_score s[1] ... s[K]
```

- `s[i]` is the partial score obtained for the `i`-th problem.

- If a user has **never submitted a solution** for a problem, then **"-" must be printed** at the corresponding position. （未提交的问题，输出“-”）
- If a user has **submitted several solutions** to solve one problem, then the **highest score** will be counted.（提交多次，取最高分）
- `rank` is calculated according to the `total_score`, and all the users with the same `total_score` obtain the same `rank`; The ranklist must be printed in **non-decreasing order** of the ranks. （成绩降序，同分同名）
- For those who have the same rank, users must be sorted in **non-increasing order** according to **the number of perfectly solved problems.** (同样的名次，按照完全解决的问题数量的降序输出)
- If there is still a **tie**, then they must be printed in **increasing order** of their id's. （如果解决问题的数量相同，则按照id的升序输出）
- For those who has **never submitted any solution** that can pass the compiler, or has never submitted any solution, they must **NOT be shown** on the ranklist. （没有任何提交通过的、没有任何提交的，不允许被输出）

It is guaranteed that **at least one user can be shown** on the ranklist.

Sample Input:

```
7 4 20
20 25 25 30
00002 2 12
00007 4 17
00005 1 19
00007 2 25
00005 1 20
00002 2 2
00005 1 15
00001 1 18
00004 3 25
00002 2 25
00005 3 22
00006 4 -1
00001 2 18
00002 1 20
00004 1 15
00002 4 18
00001 3 4
00001 4 2
00005 2 -1
00004 2 0
```

Sample Output:

```
1 00002 63 20 25 - 18
2 00005 42 20 0 22 -
2 00007 42 - 25 - 17
2 00001 42 18 18 4 2
5 00004 40 15 0 25 -
```

Key Codes： 19/25

```
struct Submission {
    int id;
    int total = 0, solved_num = 0;
    int score[6]={-2, -2, -2, -2, -2, -2};
    bool flag = false;
};

bool cmp(Submission a, Submission b) {
    if (a.total != b.total) return a.total > b.total;
    else{
        if(a.solved_num != b.solved_num) return a.solved_num >
b.solved_num;
        else return a.id < b.id;
    }
}
int main() {
    /*第一部分：输入及初始化*/
    int n, k, m;
    scanf("%d %d %d", &n, &k, &m);

    int full[k+1];                    // 输入每道题的总分
    for (int i = 1; i <= k; i++) {scanf("%d", &full[i]);}

    vector<Submission> sub(n+1);    //初始化结构
    for(int i = 1; i <= n;i++) {sub[i].id = i;}

    int id, proid, score;
    for (int i = 1; i <= m; i++){
        scanf("%d %d %d", &id, &proid, &score);
        //第一次编译未通过，记为0分
        if(score == -1 && sub[id].score[proid] == -2)
            sub[id].score[proid] = 0;
        //第一次取得满分，完全解决的问题数+1
        if(score == full[proid] && sub[id].score[proid] < full[proid])
            sub[id].solved_num++;
        //保留较大的分数值
        if(score >= sub[id].score[proid])
            sub[id].score[proid] = score;
    }

    /*第二部分：计算总分并排序*/
    for(int i=1;i<=n;i++){        //计算总分
        for(int j=1;j<=k;j++){
            if(sub[i].score[j] != -2){
                sub[i].total += sub[i].score[j];
            }
        }
    }
    sort(sub.begin()+1, sub.end(), cmp);    //排序
```

```
    /*第三部分：排名并输出*/
    int rank = 1;
    for(int i=1;i<=n;i++){
        if(sub[i].total == 0) continue;
        if(sub[i].total != sub[i-1].total) rank = i;
        printf("%d %05d %d", rank, sub[i].id, sub[i].total);
        for(int j=1;j<=k;j++){
            if(sub[i].score[j] != -2) printf(" %d", sub[i].score[j]);
            else printf(" -");
        }
        printf("\n");
    }
    return 0;
}
```

Code Modify： 25/25

题目规定：没有任何提交的，不允许被输出；全部提交都没有编译通过的，不允许被输出；

在程序中，虽然我们规定编译未通过的分数为0，但是我们必须要区分：编译通过的0分、与编译未通过的0分。这二者是不一致的，前者即使总分为0也允许输出，而后者即使总分为0也不允许输出。因此修改如下：

```
struct Submission {
    int id;
    int total = 0, solved_num = 0;
    int score[6]={-2, -2, -2, -2, -2, -2};
    bool flag = false;
};

bool cmp(Submission a, Submission b) {
    if (a.total != b.total) return a.total > b.total;
    else{
        if(a.solved_num != b.solved_num)
            return a.solved_num > b.solved_num;
        else return a.id < b.id;
    }
}
int main() {
    /*第一部分：输入及初始化*/
    int n, k, m;
    scanf("%d %d %d", &n, &k, &m);

    int full[k+1];          // 输入每道题的总分
    for (int i = 1; i <= k; i++) {scanf("%d", &full[i]);}

    vector<Submission> sub(n+1);    //初始化结构
    for(int i = 1; i <= n;i++) {sub[i].id = i;}

    int id, proid, score;
    for (int i = 1; i <= m; i++){
```

```
        scanf("%d %d %d", &id, &proid, &score);
        //修改：若编译通过，则该考生的信息必须被输出
        if(score >= 0) sub[id].flag = true;
        if(score == -1 && sub[id].score[proid] == -2)
            sub[id].score[proid] = 0;
        if(score == full[proid] && sub[id].score[proid] < full[proid])
            sub[id].solved_num++;
        if(score >= sub[id].score[proid])
            sub[id].score[proid] = score;
    }

    /*第二部分：计算总分并排序*/
    for(int i=1;i<=n;i++){          //计算总分
        for(int j=1;j<=k;j++){
            if(sub[i].score[j] != -2){
                sub[i].total += sub[i].score[j];
            }
        }
    }
    sort(sub.begin()+1, sub.end(), cmp);    //排序

    /*第三部分：排名并输出*/
    int rank = 1;
    for(int i=1;i<=n;i++){
        //修改：若有编译通过的提交，则需要输出
        if(sub[i].flag == false) continue;
        if(sub[i].total != sub[i-1].total) rank = i;
        printf("%d %05d %d", rank, sub[i].id, sub[i].total);
        for(int j=1;j<=k;j++){
            if(sub[i].score[j] != -2) printf(" %d", sub[i].score[j]);
            else printf(" -");
        }
        printf("\n");
    }
    return 0;
}
```

7、List Grades

Given a list of *N* student records with **name, ID and grade**.

You are supposed to sort the records with **respect to the grade in non-increasing order,** and output those student records of which the grades are in a given interval.

Input Specification:

Each input file contains one test case. Each case is given in the following format:

```
N
name[1] ID[1] grade[1]
name[2] ID[2] grade[2]
... ...
name[N] ID[N] grade[N]
grade1 grade2
```

where `name[i]` and `ID[i]` are strings of **no more than 10 characters** with no space, `grade[i]` is an integer in [0, 100], `grade1` and `grade2` are **the boundaries of the grade's interval**. It is guaranteed that all the grades are **distinct**.

Output Specification:

For each test case you should output the student records of which the grades are in the given interval [`grade1`, `grade2`] and are in **non-increasing order**. Each student record occupies a line with the student's **name and ID**, separated by **one space**. If there is no student's grade in that interval, output `NONE` instead.

Sample Input 1:

```
4
Tom CS000001 59
Joe Math990112 89
Mike CS991301 100
Mary EE990830 95
60 100
```

Sample Output 1:

```
Mike CS991301
Mary EE990830
Joe Math990112
```

Sample Input 2:

```
2
Jean AA980920 60
Ann CS01 80
90 95
```

Sample Output 2:

```
NONE
```

Key Codes:

```
struct Student{
    char name[15], id[15];
    int grade;
};
bool cmp(Student a, Student b){
    return a.grade > b.grade;
}
int main(){
    int n, grade1, grade2;
    scanf("%d", &n);
    Student stu[n];
    for(int i=0;i<n; i++){
        scanf("%s %s %d", stu[i].name, stu[i].id, &stu[i].grade);
    }
    scanf("%d %d", &grade1, &grade2);

    sort(stu, stu+n, cmp);
    bool flag = false;
    for(int i=0;i<n;i++){
        if(stu[i].grade >= grade1 && stu[i].grade <=grade2){
            printf("%s %s\n", stu[i].name, stu[i].id);
            flag = true;
        }
    }
    if(flag == false) printf("NONE");
    return 0;
}
```

8、Graduate Admission

It is said that in 2011, there are about 100 graduate schools ready to proceed over 40,000 applications in Zhejiang Province. It would help a lot if you could write a program to **automate the admission procedure**.（自动化录取程序）

Each applicant will have to provide two grades: **the national entrance exam grade** G_E, and **the interview grade** G_I. The **final grade** of an applicant is $(G_E + G_I)/2$. The admission rules are:

- The applicants are ranked according to their **final grades**, and will be admitted one by one from the top of the rank list.（按照总成绩排名，由高到低录取）
- If there is a tied final grade, the applicants will be ranked according to their **national entrance exam grade** G_E. If still tied, their ranks must be the **same**.（总成绩相同，考试成绩更高的排名更高；考试成绩相同，排名相同）
- If there is a tied rank, and if the corresponding applicants are applying to the same school, then **that school must admit all the applicants with the same rank, even if its quota will be exceeded.**（相同排名的考试申请了同一学校，则该校必须同时录取这两个考生）

- Each applicant may have ***K* choices** and the admission will be done according to his/her choices: （每个考生有K个志愿，并根据志愿来决定录取）
    - if according to the rank list, it is one's turn to be admitted（根据排名表，轮到一个人被录取）
        - if the ***quota*** of one's most preferred shcool is not exceeded, then one will be admitted to this school, or one's other choices will be considered one by one in order.（未超过配额，则被录取；否则，依次考察其他志愿）
        - If one gets **rejected by all of preferred schools**, then this unfortunate applicant will be rejected.（如果被所有志愿拒绝，则不被录取）

Input Specification:

Each input file contains one test case.

- Each case starts with a line containing three positive integers:
    - *N* (≤40,000), the total number of applicants;
    - *M* (≤100), the total number of graduate schools;
    - *K* (≤5), the number of choices an applicant may have.
- In the **next line**, separated by a space, there are ***M* positive integers**. The *i*-th integer is the quota of the *i*-th graduate school respectively.
- Then ***N* lines** follow, each contains **2+*K* integers** separated by a space.
    - The **first 2 integers** are the applicant's G_E and G_I, respectively.
    - The **next *K* integers** represent the preferred schools.

For the sake of simplicity, we assume that the schools are numbered from 0 to *M*−1, and the applicants are numbered from 0 to *N*−1.

Output Specification:

For each test case you should output the **admission results** for all the graduate schools.（输出所有学校的录取情况）

The results of each school must occupy a line, which contains **the applicants' numbers that school admits**.（每个学校占一行）

The numbers must be in **increasing order** and be separated by a space.（每行按考生序号升序排序）

There must be no extra space at the end of each line.（行末不允许有多余的空格）

If no applicant is admitted by a school, you must output an empty line correspondingly.（如果一个学校没有任何人被录取，则输出一个空行）

Sample Input:

```
11 6 3
2 1 2 2 2 3
100 100 0 1 2
60 60 2 3 5
100 90 0 3 4
90 100 1 2 0
90 90 5 1 3
80 90 1 0 2
80 80 0 1 2
80 80 0 1 2
80 70 1 3 2
70 80 1 2 3
100 100 0 2 4
```

Sample Output:

```
0 10
3
5 6 7
2 8

1 4
```

Key Codes: 24/30

```
struct Student{
    int id;
    int gradeE, gradeI, grade;
    int rank = 0;
    int choice[5] = {-1, -1, -1, -1 ,-1};     //考生志愿初始化为-1
    bool flag = false;                        //用于标识是否被录取
};

bool cmpGrade(Student a, Student b){
    if(a.grade != b.grade) return a.grade > b.grade;
    else return a.gradeE > b.gradeE;
}
bool cmpID(int a, int b) {return a <b;}

int main(){
    /*第一部分：输入及初始化*/
    int N, M, K;            //N--申请者数量，M--学校数量，K--可填报的志愿数量
    scanf("%d %d %d", &N, &M, &K);

    int quota[M];                                          //学校的编号从0开始
    for(int i=0;i<M;i++) {scanf("%d", &quota[i]);}         //输入每个学校的配额

    vector<Student> stu(N);
    for(int i=0;i<N;i++){
        stu[i].id = i;                                     //考生编号(从0开始)
```

```
        scanf("%d %d", &stu[i].gradeE, &stu[i].gradeI); //考生成绩
        stu[i].grade = stu[i].gradeE + stu[i].gradeI;
        for(int j=0;j<K;j++){                           //考生志愿
            scanf("%d", &stu[i].choice[j]);
        }
    }

    /*第二部分：排序排名*/
    sort(stu.begin(), stu.end(), cmpGrade);    //排序
    for(int i=0;i<N;){                         //排名
        if(stu[i].grade == stu[i+1].grade) {
            if(stu[i].gradeE > stu[i+1].gradeE){
                stu[i].rank = i+1;
                stu[i+1].rank = i+2;
            }
            else if(stu[i].gradeE == stu[i+1].gradeE)
                stu[i].rank = stu[i+1].rank = i+1;
            i += 2;
        }
        else {stu[i].rank = i+1; i++;}
    }

    /*第三部分：根据排名表确认录取情况，并完成输出*/
    vector<int> admit_num(M, 0);    //记录各学校的录取人数
    vector<int> stu_rank(M, 0);     //记录各学校录取的最后一名的排名
    vector<vector<int>> school(M);  //记录各学校录取的学生id
    for(int i=0;i<N;i++){           //记录每个学校的录取情况
        for(int j=0;j<K;j++){
            int sch_id = stu[i].choice[j];
            if(admit_num[sch_id] < quota[sch_id]){    //该校未录满
                admit_num[sch_id]++;
                if(stu_rank[sch_id] <= stu[i].rank)
                    stu_rank[sch_id] = stu[i].rank;
                school[sch_id].push_back(stu[i].id);
                break;
            }else{                                    //该校已录满
                if(stu[i].rank == stu_rank[sch_id]){  //考生排名同最后一名
                    admit_num[sch_id]++;
                    school[sch_id].push_back(stu[i].id);
                    break;
                }
            }
        }
    }

    for(int i=0;i<M;i++){
        int len = school[i].size();
        if(len == 0) printf("\n");
        else sort(school[i].begin(), school[i].end(), cmpID); //id升序排序

        for(int j=0;j<len;j++){
            if(j < len-1) printf("%d ", school[i][j]);
```

```
            else printf("%d\n", school[i][j]);
        }
    }
    return 0;
}
```

Code Modify： 30/30

根据分析，上述代码中，排名部分的代码存在问题，因此修改如下：

```
stu[0].rank = 1;                                //排名
for(int i=1;i<N;i++){
    if(stu[i].grade == stu[i-1].grade && stu[i].gradeE == stu[i-1].gradeE)
{stu[i].rank = stu[i-1].rank;}
    else if((stu[i].grade == stu[i-1].grade && stu[i].gradeE < stu[i-
1].gradeE) || (stu[i].grade < stu[i-1].grade)){
        stu[i].rank = i+1;}

}
```

9、Cars on Campus——学完Unit6中的map之后做

Zhejiang University has 8 **campuses** and a lot of gates. From each gate we can collect **the in/out times** and **the plate（车牌） numbers of the cars** crossing the gate. Now with all the information available, you are supposed to tell：

- at any specific time point, the number of cars parking on campus（任一具体的时刻，停放在校园内的车辆数）
- at the end of the day find the cars that have parked for the longest time period.（每一天停放时间最长的车）

Input Specification:

Each input file contains one test case.

- Each case starts with two positive integers：
  - N ($\leq 10^4$), the number of records（记录的数量）
  - K ($\leq 8\times10^4$) the number of queries. （要查询的数量）
- Then N lines follow, each gives a record in the format: `plate_number hh:mm:ss status``
  - `` `plate_number `` is a string of 7 English capital letters or 1-digit numbers;
  - `hh:mm:ss` represents the time point in a day by hour:minute:second, with the earliest time being `00:00:00` and the latest `23:59:59`;
  - `status` is either `in` or `out`. Note that **all times** will be within a single day.
  - Each `in` record is **paired with** the **chronologically（按时间顺序地）** next record for the same car provided it is an `out` record.
  - Any `in` records that are not paired with an `out` record are **ignored**, as are `out` records not paired with an `in` record.

    - It is guaranteed that at least one car is well paired in the input, and no car is both `in` and `out` at the same moment.
    - Times are recorded using a 24-hour clock.
- Then *K* lines of queries follow, each gives a time point in the format `hh:mm:ss`. Note: the queries are given in **ascending** order of the times.（查询时间按升序给出）

Output Specification:

For each query, output in a line **the total number of cars** parking on campus.

The last line of output is supposed to give **the plate number** of the car that has parked for the longest time period, a**nd the corresponding time length**.

If such a car is not unique, then output **all of their plate numbers in a line in alphabetical order**, separated by a space.

Sample Input:

```
16 7
JH007BD 18:00:01 in
ZD00001 11:30:08 out
DB8888A 13:00:00 out
ZA3Q625 23:59:50 out
ZA133CH 10:23:00 in
ZD00001 04:09:59 in
JH007BD 05:09:59 in
ZA3Q625 11:42:01 out
JH007BD 05:10:33 in
ZA3Q625 06:30:50 in
JH007BD 12:23:42 out
ZA3Q625 23:55:00 in
JH007BD 12:24:23 out
ZA133CH 17:11:22 out
JH007BD 18:07:01 out
DB8888A 06:30:50 in
05:10:00
06:30:50
11:00:00
12:23:42
14:00:00
18:00:00
23:59:00
```

Sample Output:

```
1
4
5
2
1
0
1
JH007BD ZD00001 07:20:09
```