老师想知道从某某同学当中，分数最高的是多少，现在请你编程模拟老师的询问。当然，老师有时候需要更新某位同学的成绩.

先来说说这题的 3 种做法：

最简单的就是暴力了。每次查询直接做。

修改复杂度 O(1)，查询复杂度 O(N)

```
1   #include <stdio.h>
2   #include <algorithm>
3   using namespace std;
4
5   const int MAXN=100000;
6
7   int data[MAXN+5];
8
9   int querymax (  int l ,  int r ) {
10          int ans=data[l];
11          for(int i=l+1;i<=r;i++) ans=max(ans,data[i]);
12          return ans;
13  }
14
15  void update(int idx,int value){
16          data[idx]=value;
17  }
18
19  int main(){
20          int n,m;
```

```
21          while(~scanf("%d%d",&n,&m)){
22                  for(int i=1;i<=n;i++){
23                          scanf("%d",&data[i]);
24                  }
25                  char order;
26                  int a,b;
27                  for(;m--;){
28                          scanf(" %c%d%d",&order,&a,&b);
29                          if(order=='U'){
30                                  update(a,b);
31                          }else if(order=='Q'){
32                                  if(a>b)swap(a,b);
33                                  printf("%d\n",querymax(a,b));
34                          }
35                  }
36          }
37          return 0;
38  }
```

……如果他们手一抖，写成 N<=100000,M<=100000 怎么办？

!!!注意：以下内容涉及高级数据结构!!!

首先，有 ACM 经验的选手，一看这个题，第一反应，线段树好~

线段树这种数据结构，修改 O(logn)，查询 O(logn)，但是要预处理 O(nlogn)

这个，如果没有 ACM 经验，也不打算认真做 ACM 的同学，看看就好。

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

```
2   #include <algorithm>
3   using namespace std;
4
5   const int MAXN=100000;
6
7   int data[MAXN+5];
8   int maxarr[MAXN*4+5];
9
10  void build(int p,int l,int r) {
11          if(l==r){
12                  maxarr[p]=data[l];
13                  return ;
14          }
15          int m = ( l + r ) >> 1 ;
16          int lchild=p<<1,rchild=p<<1|1;
17          build ( lchild , l , m ) ;
18          build ( rchild , m+1 , r ) ;
19          maxarr[p]=max(maxarr[lchild],maxarr[rchild]);
20  }
21
22  int querymax (  int L ,  int R ,  int p ,  int l ,  int r ) {
23          if ( L <= l && r <= R ) {
24                  return maxarr[p];
25          }
26          int m = ( l + r ) >> 1 ;
27          int lans=-1,rans=-1;
```

```
28          if ( L <= m ) lans=querymax ( L , R , p << 1 , l , m ) ;
29          if ( m <   R ) rans=querymax ( L , R , p << 1 | 1 , m + 1 , r ) ;
30          if(lans==-1)return rans;
31          if(rans==-1)return lans;
32          return max(lans,rans);
33  }
34
35  void update(int idx,int value,int p,int l,int r){
36          if(l==r&&l==idx){
37                  maxarr[p]=value;
38                  return;
39          }
40          int m = ( l + r ) >> 1 ;
41          if ( idx <= m ) update( idx, value, p << 1, l, m );
42          if ( m <   idx ) update( idx, value, p << 1|1, m+1, r );
43          maxarr[p]=max(maxarr[p<<1],maxarr[p<<1|1]);
44  }
45
46  int main(){
47          int n,m;
48          while(~scanf("%d%d",&n,&m)){
49                  for(int i=1;i<=n;i++){
50                          scanf("%d",&data[i]);
51                  }
52                  build(1,1,n);
53                  char order;
```

```
54                  int a,b;
55                  for(;m--;){
56                          scanf(" %c%d%d",&order,&a,&b);
57                          if(order=='U'){
58                                  update(a,b,1,1,n);
59                          }else if(order=='Q'){
60                                  if(a>b)swap(a,b);
61                                  printf("%d\n",querymax(a,b,1,1,n));
62                          }
63                  }
64          }
65          return 0;
66  }
```

有没有什么办法，普通人能去想出来，又不需要专门去学过高级数据结构呢？

有一个，叫块状链表（其实块状数组也行吧）。

好吧，其实分块的思想也不怎么常见，谈针对面试的算法的思路，都谈分治、贪心、递推、动态规划，没见人说过分块的样子。但是，分块是比较容易理解的。

很简单，现在我们把整个 N 大小的数组按顺序拆成 sqrt(n)（根号 n）个小数组，每个小数组有 sqrt(n)个元素

比如 1 2 3 4 5 6 7 8 9

现在拆成

1 2 3

4 5 6

7 8 9

然后对每一个小块，我们除了改掉相应位置的值，还要额外记录一下整个小块的最大值。

如果我更新的时候，那个小块的最大值增大，那很简单，最大值也增大了。

如果把最大值改小了呢？为了正确性，只能把整个小块扫一遍，重新算出最大值了。

所以，修改的复杂度是 O(sqrt(n))

现在看查询。我们要充分利用分小块以后的信息。

比如要查询 2 到 9 的最大值。按之前最朴素的暴力的做法，我要访问 2、3、4、5、6、7、8、9

现在有小块的最大值信息了，我只要判断每个小块是否在查询区间内，不在的没用，一部分在的，就暴力查找，如果是完整在查询区间内的，我们就利用之前算好的这个小块内的最大值。

所以，分块的情况下，查询 2 到 9 的最大值，需要看看 2、3，以及 4~6 的最大值，7~9 的最大值。

很容易证明，查询的复杂度是 O(sqrt(n))的（最坏是 sqrt(n)个块全部要用，左右 2 边只盖住 sqrt(n)-1 个数，要暴力遍历过去）

//TODO:在这里补上分块法的代码

3 种流派全部可过，但是明显的，在极限数据情况下，能够轻易区分出普通应聘者，会动脑的应聘者和有 ACM 经验的应聘者了。

开发一个简单错误记录功能小模块，能够记录出错的代码所在的文件名称和行号。
处理:
1.记录最多 8 条错误记录，对相同的错误记录(即文件名称和行号完全匹配)只记录一条，错误计数增加；(文件所在的目录不同，文件名和行号相同也要合并)
2.超过 16 个字符的文件名称，只记录文件的最后有效 16 个字符；(如果文件名不同，而只是文件名的后 16 个字符和行号相同，也不要合并)
3.输入的文件可能带路径，记录文件名称不能带路径

1　这个程序，看似简单，实际上考了很多细节，我花了很多时间来做，不是被算法

```
2   难住，而是在细节上出了很多漏洞。整体要求很快就实现了，而一直报答案错误
3   ，第一个原因，只能是 8 条记录以内，我全部输出；第二个原因，题目要求稳定排
4   序，我排出来了，却颠倒了关键字的顺序；第三个原因，我以为都 OK 了，却忘了
5   文件名 16 个字符的要求。三个原因我找了很久，我以为我是理解题目错误，实际
6   上就是忽略了细节，造成浪费了大量的时间。
7   #include<string>
8   #include<iostream>
9   #include<vector>
10  #include<list>
11  #include<string.h>
12  using namespace std;
13  void Process(vector<string>& m,vector<int>& t)
14  {
15          int size=t.size();
16          for (int i=0;i<size;i++)
17          {
18                  if (1==t[i])
19                          continue;
20                    for (int j=1;j<size-i;j++)
21                  {
22                          if(t[j-1]<t[j])
23                          {
24
25                                  swap(t[j-1],t[j]);
26                                  swap(m[j-1],m[j]);
27                          }
```

```
28                      }
29              }
30      }
31      int main()
32      {
33              vector<string> mscanf;
34              vector<int> times;
35              string str;
36              while (getline(cin,str))
37              {
38                      if (str.length()==0)
39                      {
40                              break;
41                      }
42                      string info=str.substr(str.rfind("\\")+1);
43                      string partinfo=info.substr(0,info.rfind(" "));
44                      if(partinfo.length()>16)
45                      {
46                              partinfo=partinfo.substr(partinfo.length()-16,16);
47                              info=partinfo+str.substr(str.rfind(" "));
48                      }
49                      int n=0;
50                      for (int i=0;i<mscanf.size();i++)
51                      {
52                              if(strcmp(mscanf[i].c_str(),info.c_str())==0)
53                              {
```

```
54                                  times[i]++;n++;
55                          }
56                  }
57                  if(0==n)
58                  {
59                          mscanf.push_back(info);
60                          times.push_back(1);
61                  }
62          }
63          Process(mscanf,times);
64          int nsize=times.size();
65          if(nsize>=8)
66                  nsize=8;
67          for (int i=0;i<nsize;i++)
68          {
69                  cout<<mscanf[i]<<" "<<times[i]<<endl;
70          }
71          vector<string>().swap(mscanf);
72          vector<int>().swap(times);
73
74          return 0;
75  }
```

扑克牌游戏大家应该都比较熟悉了，一副牌由 54 张组成，含 3~A，2 各 4 张，小王 1 张，大王 1 张。牌面从小到大用如下字符和字符串表示（其中，小写 joker 表示小王，大写 JOKER 表示大王）:)

3 4 5 6 7 8 9 10 J Q K A 2 joker JOKER

输入两手牌，两手牌之间用“-”连接，每手牌的每张牌以空格分隔，“-”两边没有空格，如：4 4 4 4-joker JOKER
请比较两手牌大小，输出较大的牌，如果不存在比较关系则输出 ERROR

基本规则：
（1）输入每手牌可能是个子，对子，顺子（连续 5 张），三个，炸弹（四个）和对王中的一种，不存在其他情况，由输入保证两手牌都是合法的，顺子已经从小到大排列；
（2）除了炸弹和对王可以和所有牌比较之外，其他类型的牌只能跟相同类型的存在比较关系（如，对子跟对子比较，三个跟三个比较），不考虑拆牌情况（如：将对子拆分成个子）
（3）大小规则跟大家平时了解的常见规则相同，个子，对子，三个比较牌面大小；顺子比较最小牌大小；炸弹大于前面所有的牌，炸弹之间比较牌面大小；对王是最大的牌；
（4）输入的两手牌不会出现相等的情况。

答案提示：
（1）除了炸弹和对王之外，其他必须同类型比较。
（2）输入已经保证合法性，不用检查输入是否是合法的牌。
（3）输入的顺子已经经过从小到大排序，因此不用再排序了.

```
1  import java.util.ArrayList;
2  import java.util.Scanner;
3
4  public class Main {
5          public static void main(String[] args) {
6                  Scanner sc =  new Scanner(System.in);
7                  String[] left,right;
8                  String[] line;
9                  String nextLine,outString;
```

```
10                  while(sc.hasNext()){
11                          nextLine = sc.nextLine();
12                          //有王炸就王炸最大
13                          if(nextLine.contains("joker JOKER")){
14                                  outString =  "joker JOKER";
15                          }else{
16                                  //拆分 先拆成左右 再拆成单排
17                                  line = nextLine.split("-");
18                                  left = line[0].split(" ");
19                                  right = line[1].split(" ");
20
21                                  //炸弹最大
22                                  if(left.length ==  4 && right.length !=  4){
23                                          outString = line[0];
24                                  }else if(right.length ==  4 && left.length !=  4){
25                                          outString = line[1];
26                                  }
27                                  // 牌数相同的情况下比较最小的牌的大小，compare 方法返回牌所对应的值
28                                  else if(right.length == left.length){
29                                          if(count(left[0])>count(right[0])){
30                                                  outString = line[0];
31                                          }
32                                          else{
33                                                  outString = line[1];
34                                          }
35                                  }else{
```

```
36                                          outString =  "ERROR";
37                                  }
38                          }
39
40                          System.out.println(outString);
41
42                  }
43          }
44
45          //2-JOKER 按大小返回 2-16
46          private static int count(String str) {
47                  return "345678910JQKA2jokerJOKER".indexOf(str);
1           }
2   }
```