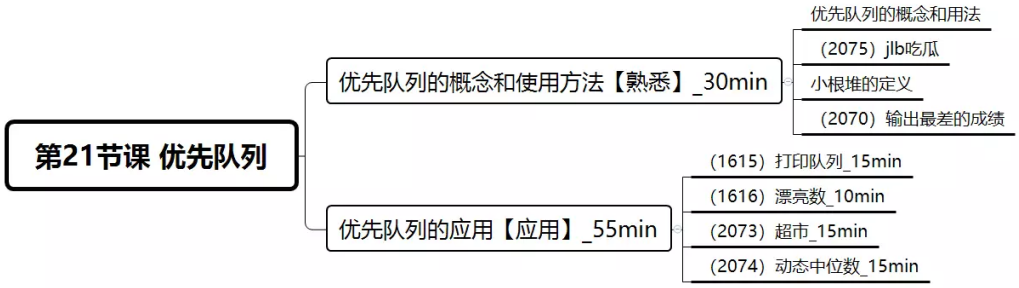
21 优先队列_讲义



优先队列的概念与使用方法

优先队列（priority queue）是STL库提供的一个数据结构，可以用来快速获取一堆元素中的最大元素。

优先队列常见用法：

```
//定义
priority_queue<int> pq;
//把元素加入优先队列，操作复杂度O(log n)
pq.push(10);
//获取优先队列中的最大元素，操作复杂度O(1)
pq.top();
//删除优先队列中的最大元素，操作复杂度O(log n)
pq.pop();
//判空，操作复杂度O(1)
if(pq.empty()){
    cout<<"优先队列为空"<<endl;
}
```

注意：优先队列容器内保存的类型必须有”小于号”操作方便比较大小。所以如果要放入自定义的结构，需要重载小于号。

例1：（2075）jlb吃瓜

【题目描述】
jlb的口袋里有n个西瓜，接下来的n天jlb的妈妈每天都会给她一个新的西瓜。
西瓜的质量可以用一个值a来表示，a越大表示西瓜的质量越好。
由于jlb的食量有限，她每天只能吃一个西瓜。
现在给你n个整数，表示jlb口袋里原有西瓜的质量。
再给你n个整数，表示未来n天jlb的妈妈将会给她的西瓜质量。(这n个数的顺序不能改变)
第一天，jlb会先吃掉一个质量最好的西瓜，然后她的妈妈会给她一个新的西瓜；
第二天，jlb又会吃掉一个质量最好的西瓜，然后她的妈妈会给她一个新的西瓜...

…
我们想知道未来n天jlb吃的西瓜的质量是多少?

【输入】
第一行一个整数n，表示jlb口袋里原有的西瓜数量和天数。
第二行n个元素，表示jlb口袋里原有西瓜的质量，两两间以空格间隔。
第三行n个元素，表示未来n天依次得到的西瓜的质量，两两间以空格间隔。
$(n \leq 10000, a_i \leq 100000)$

【输出】
共n行，输出在每次jlb得到一个新的西瓜前，她吃掉的最好的西瓜的质量是多少。

【输入样例】
3
3 9 6
5 2 10

【输出样例】
9
6
5

分析

可以用一个优先队列模拟这个过程。算法的时间复杂度为 $O(logn)$ 。

答案

```
priority_queue<int> pq;
int n;
int main(){
    cin>>n;
    int a;
    for(int i=1;i<=n;i++){
        cin>>a;
        pq.push(a);
    }
    for(int i=1;i<=n;i++){
        cout<<pq.top()<<endl;
        pq.pop();
        cin>>a;
        pq.push(a);
    }
    return 0;
}
```

可以看到优先队列的优势在于：可以动态的获取实时的最值。
提问：如果每次吃掉是一个质量最差的瓜，应该如何修改程序?

优先队列获取最小元素的方法：

1）基本数据类型，以int类型为例：

```
priority_queue<int,vector<int>,greater<int> > pq;//也称为定义了一个小根堆
//获取优先队列中的最小元素
pq.top();
//删除优先队列中的最小元素
pq.pop();
```

2）自定义数据类型，如零食，包含名字和价格

```
struct snack{
string name;
int price;
    bool operator <(const snack s2) const{
        return price>s2.price;
    }
};
priority_queue<snack> pq;
```

例2:（2070）输出最差的成绩

【题目描述】

有n个学生，分别为 S_1 ，S_2 ，$\cdots$ ，S_n 。每个学生都有语数外三门课的成绩。

现在依次输入每个学生的三门成绩，对于每次输入，需要输出当前成绩最差（即总分最低）的学生的名字。

【输入】

第一行一个整数n，表示接下来要输入n个学生的成绩。

第二行~n+1行，每行包含一个字符串代表学生名字，然后三个整数分别表示学生语数外的成绩。

$(n \leq 10000)$

【输出】

共n行，输出在每次输入一个学生成绩时，当前成绩最差的学生的名字。

【输入样例】

3

Evelynn 90 90 90

Liqiu 60 60 60

Natie 100 100 100

【输出样例】

Evelynn

Liqiu

Liqiu

答案

```
struct student{
    string name;
    int m,c,e;
    bool operator<(const student s2) const{
        return m+c+e>s2.m+s2.c+s2.e;
    }
}s;
priority_queue<student> pq;
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>s.name>>s.m>>s.c>>s.e;
        pq.push(s);
        cout<<pq.top().name<<endl;
    }
    return 0;
}
```

优先队列的应用

例3:（1615）打印队列

【题目描述】

jpl只有一台打印机，但是有很多文件需要打印，因此打印任务不可避免需要等待。有些打印任务比较急，有些不那么急，所以每个任务都有一个1~9之间的优先级，优先级越高表示任务越急。

打印机的运作方式如下：首先从打印队列里取出一个任务J，如果队列里有比J更急的任务，则直接把J放到打印机的尾部。否则打印任务J（此时不会把它放回打印队列）。输入打印队列中各个任务的优先级以及所关注的任务在队列中的位置（队首位置为0），输出该任务完成的时刻。每个任务都需要1分钟打印。例如，打印队列为{1,1,9,1,1,1}，目前处于队首的任务最终完成时刻为5。

【输入】

第一行输入n和top，分别表示jpl需要打印的文件的数量和关注的任务的位置。

第二行输入n个1~9之间的数字表示打印任务的队列。

$(n \leq 1000)$

【输出】

输出关注的任务的打印完成时刻。

【输入样例】

5 0

1 1 2 4 3

【输出样例】

4

分析

从打印队列里取出任务和放回打印队列尾部的操作，可以通过队列来实现。
而比较是否有任务比当前任务更紧急，可以通过优先队列来实现。
1.先把所有任务都同时放进队列和优先队列中。
2.每次取出队列的头部任务与优先队列进行比较，
如果不相同，就把该任务放入队列的尾部，
如果相同，打印该任务，然后两边队列的头部任务都出列。

答案

```
#include<bits/stdc++.h>
using namespace std;
struct work{
    int value;
    int id;
    bool operator<(const work &w2)const{
        return value<w2.value;
    }
}a;
int main(){
    int n,top;
    priority_queue<work> pq;
    queue<work> q;
    cin>>n>>top;
    for(int i=0;i<n;i++){
        cin>>a.value;
        a.id=i;
        pq.push(a);
        q.push(a);
    }
    int time=0;
    while(1){
        work qf=q.front();
        work pqf=pq.top();
        if(qf.value==pqf.value){
            time++;
            if(qf.id==top){
                cout<<time<<endl;
                break;
            }
            q.pop();
            pq.pop();
        }
        else{
            q.pop();
            q.push(qf);
```

```
        }
    }
    return 0;
}
```

例4:（1616）漂亮数

【题目描述】
jpl喜欢的数字被称为漂亮数，漂亮数是一些因子只有2，3，5的数。数列1,2,3,4,5,6,8,9,10,12,15,...写出了从小到大的前11个漂亮数，1属于漂亮数。
现在请你编写程序，找出第1500个漂亮数是什么。

【输入】
无

【输出】
第1500个漂亮数是什么。

【输入样例】
无

【输出样例】
不告诉你

分析

将漂亮数列依次写出来：
1 2 3 4 5 6 8 9 10 12 15 16 18 20，
可以发现6=2×3 8=2×4 9=3×3 10=2×5，这几个数字是用1,2,3,4,5×2、3、5得到的。同理，12=2×6 15=3×5 16=2×8 18=2×9 20=2×10，这几个数字是用5,6,8,9,10×2、3、5得到的，即数列靠后面的数字都是它们之前的数字×2、3、5得来的，可以利用优先队列动态的特性来完成。
建立一个小根堆，先把1插入小根堆。然后我们每次取出堆顶的时候，把堆顶的数乘2，3，5的数插入堆中。这样取出的第1500个数，就是第1500个漂亮数。
但是这样会产生重复，所以在取出来的时候要注意做去重操作。

答案

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    priority_queue<long long,vector<long long>,greater<long long> > pq;
    pq.push(1);
    long long x;
    for(int i=1;i<=1500;i++){
```

```
        x=pq.top();
        pq.pop();
        pq.push(x*2);
        pq.push(x*3);
        pq.push(x*5);
        while(pq.top()==x) pq.pop();//注意去重，如果后面堆顶的数和这个数一样大
                                    //就直接删除
    }
    cout<<x<<endl;
    return 0;
}
```

例5:（2073）超市

【题目描述】
超市里有 N 件商品，每件商品都有利润 pi 和过期时间 di，每天只能卖一件商品，过期商品不能再卖。求合理安排每天卖的商品的情况下，可以得到的最大收益是多少。

【输入】
第一行一个整数N表示商品的数量。
第二行~第n+1行每行2个整数分别表示pi 和 di。（0≤N≤10000,1≤pi,di≤10000）

【输出】
一个整数表示的最大收益值。

【输入样例】
7
2 1
20 1
10 3
8 2
100 2
5 20
50 10

【输出样例】
185

分析

先按照过期时间对商品排序，然后建立一个小根堆，顶部元素为队列中利润最小的商品。依次遍历每一个商品。
当过期时间t大于队列中商品个数时，直接入队列。
当过期时间t等于队列中商品个数时，表示已经对前t天安排了t个商品进行售卖，如果此时扫描到的商品比队列顶部的商品利润高的话，将它替换掉即可。

答案

```
struct node{
    int p,d;//p是利润，d是过期时间
    bool operator<(const node &n2)const{
        return p>n2.p;
    }
}a[10005];

bool cmptime(node n1,node n2){
    return n1.d<n2.d;
}
int n;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].p>>a[i].d;
    }
    sort(a+1,a+n+1,cmptime);
    priority_queue<node> pq;
    int num=0;
    int ans=0;
    for(int i=1;i<=n;i++){
        if(num<a[i].d){
            pq.push(a[i]);
            ans+=a[i].p;
            num++;
        }
        else if(a[i].p>pq.top().p){
            ans-=pq.top().p;
            ans+=a[i].p;
            pq.pop();
            pq.push(a[i]);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

例6:（2074）中位数

【题目描述】
依次读入一个整数序列，每当已经读入的整数个数为奇数时，输出已读入的整数构成的序列的中位数。

【输入】
第一行输入n，代表接下来要输入n个整数，其中n一定为奇数。
第二行输入n个整数，都在int的范围内。 （$n \leq 10^5$ ）

【输出】
输出两行，第一行包含一个整数m，代表要输出的中位数的个数。
第二行输出m个中位数。

【输入样例】
9
7 8 9 1 2 3 6 5 4

【输出样例】
5
7 8 7 6 5

分析

用对顶堆的方法去解决。也就是建立两个堆，一个小根堆，一个大根堆。在依次读入这个整数序列的过程中，设当前序列的长度为M，我们始终保持：
序列中从小到大排名为1~M/2的整数存储在大根堆中；
序列中从小到大排名为M/2+1~M的整数存储在小根堆中。
任何时候，如果某一个堆中的元素个数过多，打破了这个性质，就取出该堆的堆顶插入另一个堆。这样一来，序列的中位数就是小根堆的堆顶。
每次新读入一个数值X后，若X比中位数小，则插入大根堆，否则插入小根堆，在插入之后每次检查并维护上述性质即可。

答案

```
int n;
int main(){
    cin>>n;
    cout<<(1+n)/2<<endl;
    priority_queue<int> bpq;//大根堆
    priority_queue<int,vector<int>,greater<int> > spq;//小根堆
    int bnum=0,snum=0,outnum=0;//保存两个堆内的元素个数，以及输出的元素的个数
    int a;
    cin>>a;
    cout<<a<<" ";
    spq.push(a);//第一个数先放入小根堆
    snum++;
    outnum++;
    for(int i=2;i<=n;i++){
        cin>>a;
        if(a<spq.top()){//比中位数小则插入大根堆
            bpq.push(a);
            bnum++;
        }
        else{//否则插入小根堆
            spq.push(a);
            snum++;
        }
```

```
//如果某个堆的元素个数多了，则出堆插入另一个堆
        if(bnum>i/2){
            int x=bpq.top();
            bpq.pop();
            spq.push(x);
            bnum--;
            snum++;
        }
        if(snum>(i+1)/2){
            int x=spq.top();
            spq.pop();
            bpq.push(x);
            snum--;
            bnum++;
        }
        if(i%2==1)  cout<<spq.top()<<" ";
    }
    return 0;
}
```

作业

A

（1617）jpl做医生

（2076）合并果子