米哈游 2024春季招聘 后端试卷 在线考试 - 正式考试阶段

编程题 | 20分 1/3

题目 提交记录

转圈圈

时间限制： 3000MS
内存限制： 589824KB
题目描述：
黑塔是一个喜欢转圈圈的女孩子。
有一天，黑塔遇到了n个怪物，第i个怪物的生命值为h_i。
由于被守护者之影禁用了普通攻击，黑塔现在只能使用E技能攻击敌人。具体地说，黑塔每次使用E技能，会对敌方全体造成E点伤害。
除此之外，每当一个怪物的生命值首次减小到其最大生命值的50%及以下（含50%）时，如果敌方尚有怪物存活（生命值大于零），那么黑塔就会自动释放一次追加攻击“转圈圈”，对敌方全体造成R点伤害。如果一次攻击同时使多个敌人满足以上条件，那么黑塔也会连续释放多次“转圈圈”，直到“转圈圈”次数耗尽或者敌人全部倒下为止。在“转圈圈”结束之前，黑塔无法再次使用E技能。
作为天才俱乐部#83的天才，黑塔只用了0.0114514秒就算出了自己需要使用多少次E技能才能击败这些怪物，以及在这个过程中她会释放多少次“转圈圈”，觉得这个问题太简单了，于是将其留给了你作为课后习题。

输入描述

第一行一个正整数T，表示有T组数据。
对于每一组数据，第一行一个正整数n，表示怪物的数量；
第二行n个正整数$h_1,h_2,...,h_n$,表示每个怪物的生命值；
第三行两个正整数E,R，分别表示黑塔E技能的伤害和“转圈圈”的伤害。
$1<=n<=10^5$，$1<=T<=10$，$1<=h_i,E,R<=10^9$。

输出描述

对于每一组数据，输出一行两个正整数$cntE, cntR$, 分别表示黑塔使用E技能的次数和“转圈圈”的次数。

样例输入

```
3
5
100 50 60 80 70
25 10
5
100 50 60 80 70
20 20
5
100 200 300 4000 5000
50 1000
```

样例输出

```
2 5
2 3
1 5
```

提示

对于第一组数据：
初始怪物生命值为[100,50,60,80,70]；
黑塔使用E技能，怪物生命值变为[75,25,35,55,45]；
怪物2生命值小于等于50%，触发一次转圈圈，怪物生命值变为[65,15,25,45,35]；
怪物3,5生命值小于等于50%，触发两次转圈圈，生命值变为[45,0,5,25,15]；
怪物1,4触发两次转圈圈，生命值变为[25,0,0,5,0]；
使用E技能，生命值变为[0,0,0,0,0]，战斗结束，**一共使用2次E，5次转圈圈。**

对于第二组数据：
初始怪物生命值为[100,50,60,80,70]；
黑塔使用E技能，怪物生命值变为[80,30,40,60,50]；
再次使用E,生命值变为[60,10,20,40,30]；
怪物2,3,4,5触发四次转圈圈，但是只转3次所有怪物就全部被击杀，因此**一共使用2次E，3次转圈圈。**

对于第三组数据：
初始生命值为[100,200,300,4000,5000]
使用E技能，[50,150,250,3950,4950]
怪物1触发一次转圈圈，[0,0,0,2950,3950]
怪物2，3触发两次转圈圈，[0,0,0,950,1950]
怪物4，5触发两次转圈圈，[0,0,0,0,0]，战斗结束，**一共使用1次E，5次转圈圈。**

Code:

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstdio>
#include<numeric>
#include<cstring>
#define __DEBUG__x
using namespace std;
int h[100010],cur_h[100010];
int vis[100010];//当前怪物是否已经转过圈
int get_minh(int n,int &index){
    int min_h=1e8;
```

```
    for(int i=0;i<n;i++){
        if(!vis[i]){
            if(cur_h[i]<min_h){
               min_h=cur_h[i];
               index=i;
            }
        }
    }
    return min_h;
}
void show_h(int n){
    for(int i=0;i<n;i++){
        cout<<cur_h[i]<<" ";
    }
    cout<<endl;
}
int main(){
    int t=0;
    freopen("data1.in","r",stdin);
    cin>>t;
    while(t--){
        int cycle=0,cntE=0,cntR=0,min_h=1e8,n=0,e=0,r=0;
        cin>>n;
        for(int i=0;i<n;i++){
            cin>>h[i];
            cur_h[i]=h[i];
        }
        cin>>e>>r;
        # ifdef __DEBUG__
            show_h(n);
        # endif
        memset(vis,0,sizeof(vis));
        //1.每个怪物只能让黑塔转一次圈 故 cntR<=n
        //2.每次只能执行一种
        //3. 转圈次数用完前不能执行 E 操作
        int index=0;
        while((min_h=get_minh(n,index))!=0){//step1.计算当前(可执行转圈操作)的怪兽血量的最小值
            //关键公式推导 当前血量到预警血量的最少次数 h'-k*e<=h/2=>k>=(2*h'-h)/(2*e)
            //cout<<"index: "<<index<<endl;
            int  k=ceil(((2*min_h-h[index])*1.0)/(2.0*e));//step2.计算血量最小的怪兽让黑塔转圈所需要的最小次数
            cntE+=k;
```

```
            # ifdef __DEBUG__
                cout<<min_h<<" "<<k<<endl;
            #endif
            int max_h=-1e8;
            cycle=0;
            for(int i=0;i<n;i++){
                cur_h[i]=max(0,cur_h[i]-k*e);//step3.对每一个怪物执行
cntE 次操作 E
                if(!vis[i]&&cur_h[i]<=h[i]/2){//step4. 判断并统计之前没
有转圈而现在需要转圈的怪物数量和 id
                    cycle++;//记录转圈数
                    vis[i]=1;//记录第 i 个怪物让对手转圈
                }
                max_h=max(max_h,cur_h[i]);//记录当前最大怪物血量
            }
            # ifdef __DEBUG__
              show_h(n);
            # endif
            //step5.计算当前血量最多的怪物血量清 0 最少需要的次数
            int min_k=ceil((max_h*1.0)/r);
            if(min_k<cycle){//step6.特判:当前血量最多的怪物血量清 0 所需要
转圈操作的数量 min_k
                //都比当前需要的转圈数还小-->再转圈次数用完之前怪物被团灭。
                cntR=min_k;//end1: 次数执行的转圈数就为消灭血量最多的怪物所
需要的转圈操作数
                break;
            }
            //step6.执行 cycle 次转圈操作
            cntR+=cycle;//cycle 个操作转圈操作一定会被执行 故这里记录其值
            max_h=-1e8;
            int max_index=0;
            while(1){
                int cnt=0;
                for(int i=0;i<n;i++){// step7.对每一个怪物执行 cycle 次转
圈操作
                    cur_h[i]=max(0,cur_h[i]-cycle*r);
                    if(!vis[i]&&cur_h[i]<h[i]/2){//step8.判断是否有新的
转圈操作产生
                        vis[i]=1;
                        cnt++;//记录新的转圈操作
                        if(cur_h[i]>max_h){
                            max_h=cur_h[i];
                            max_index=i;
                        }
```

```
                }
            }
            if(cnt==0){//如果新转圈次数为 0
                break;
            }
            cycle=cnt;//更新当前需要新转的圈的数量
            int k=ceil((max_h*1.0)/(1.0*r));
            if(k<=cnt){
             //cnt 次数执行完之前 怪物都死完了
             cntR+=k;
             break;
            }
            cntR+=cnt;//记录转圈操作次数
        }
        #ifdef __DEBUG__
          show_h(n);
        #endif
        if(accumulate(cur_h,cur_h+n,0)==0){
            //所有怪物都被消灭了
            break;
        }
        if(cntR==n){//step10.转圈次数用完转圈操作不能进行->只能全部执
行 E 操作
            //最多执行的 e
            //cout<<cntE<<" dg "<<max_h<<endl;
            // h'-ek=0->k=h'/e 其中 h'为当前最大值
            cntE+=(*max_element(cur_h,cur_h+n))/e;
            break;
        }
    }
    cout<<cntE<<" "<<cntR<<endl;
  }
}
```

运行结果:

笔试题 2:

整数数组 nums 按升序排列，数组中的值 互不相同 。在传递给函数之前，nums 在预先未知的某个下标 k（0 <= k < nums.length）上进行了 旋转，使数组变为 [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]（下标 从 0 开始 计数）。例如， [0,1,2,4,5,6,7] 在下标 3 处经旋转后可能变为 [4,5,6,7,0,1,2] 。给你 旋转后 的数组 nums 和一个整数 target ，如果 nums 中存在这个目标值 target ，则返回它的下标，否则返回 -1 。你必须设计一个时间复杂度为 O(log n) 的算法解决此问题。

提示：nums 中的值独一无二

样例输入：

```
8
1
1
0
8
123 1234 12345 77 89 100 111 120
111
6
5 6 1 2 3 4
4
7
4 5 6 7 0 1 2
0
7
4 5 6 7 0 1 2
5
9
7 8 9 1 2 3 4 5 6
5
9
7 8 9 1 2 3 4 5 6
7
4 5 6 7 0 1 2
```

```
3
样例输出：
-1
6
5
4
1
7
0
```

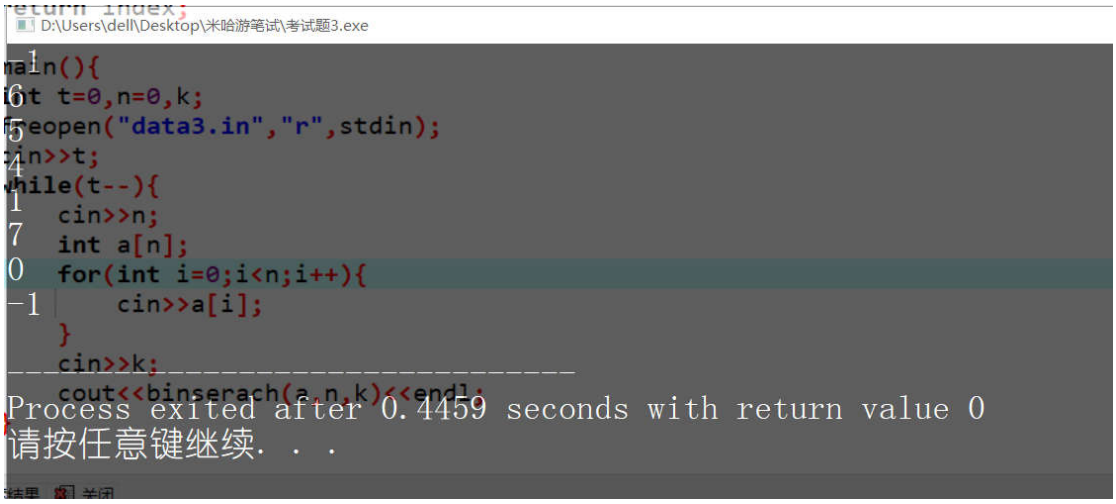
Code:

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstdio>
#include<numeric>
#include<cstring>
#define __DEBUG__x
using namespace std;
int binserach(int a[],int n,int val){
    if(n==0){
        return -1;
    }
    if(n==1){
        return val==a[0]? 0:-1;
    }
    int left=0,right=n,index=-1;
    while(left<right){
        int mid=(left+right)>>1;
        if(a[mid]==val){//查找到结果
            index=mid;
            break;
        }
        //一定成立的关系
        /* 1.a[0]>a[n]
           2.当a[0]<a[mid]时 ->在划分的前半部分
             有a[mid]>a[0]>a[n]-> a[mid]>a[n]
             若a[mid]<val(即查找的值比当前的值还大)
           3.当a[0]>a[mid]时 ->在划分的后半部分

        */
        //step1.判断当前位置在前半部分还是在后半部分
        if(a[0]<=a[mid]){//当前值比第一个元素大 ->[0->mid]是单调的
            //前半部分[4 5 6 7(mid) 0 1 2]
            if(a[mid]>val&&val>=a[0]){//当前数太小了应该右移动指针当前位置比最后一个值大都需要移动
```

```
            //[4 5 6(val) 7(mid) 0 1 2]
            right=mid;
        }else{
            //[4 5 6 7(mid) 0 1(val) 2]
            left=mid+1;
        }
    }else{
        //mid 在后半部分 [7 8 9 1 2(mid) 3 4 5 6]
        if(a[mid]<val&&val<=a[n-1]){//当前的值预测小了 指针左移 知道所知的位置比a[0]大
            //[7 8 9 1 2(mid) 3 4 5(val) 6]
            left=mid+1;
        }else{
            //[7 8(val) 9 1 2(mid) 3 4 5 6]
            right=mid;
        }
    }
  }
  return index;
}
int main(){
  int t=0,n=0,k;
  freopen("data3.in","r",stdin);
  cin>>t;
  while(t--){
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++){
        cin>>a[i];
    }
    cin>>k;
    cout<<binserach(a,n,k)<<endl;
  }
}
```

输出结果



笔试题 3

给定字符串序列 words 和 s，计算 words 中有多少个 s 的子序列。其中，字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。

样例输入：

2

hksjabvjlsvlsbv

9

hksjabvjlsvlsbvx hksjabvjlsvlsbv k kajsv skatpgf ksvjbs vbs abs jjllsv

zxcsedfg

6

xc fgx zxcsxedfg csedgf yzxcsedfgv zxdf

样例输出：

5

2

Code:

```
#include<iostream>
#include<string>
#include<vector>
#include<cstdio>
using namespace std;
const int N=1e9;
// 时间复杂度为 O(N)
bool checkSubStr(string s,string word){
    int quikIndex=0,slowIndex=0;
    for(;quikIndex<s.size();quikIndex++){
        if(s[quikIndex]==word[slowIndex]){
            slowIndex++;
        }
    }
    return slowIndex==word.size();
}
```

```
int main(){
    int n,t,ans=0;
    freopen("data2.in","r",stdin);
    cin>>t;
    while(t--){
        string s;
        cin>>s;
        cin>>n;
        for(int i=0;i<n;i++){
            string word;
            cin>>word;
            if(checkSubStr(s,word)){
                ans++;
            }
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

运行结果: