202203 解题报告

张子阳 2021210530

题目通过截图

3115599	张子阳	张子阳	通信系统管理	07-04 19:07	2.382KB	CPP14	正确	100	1.781s	98.10MB
3114469	张子阳	张子阳	博弈论与石子合并	07-04 12:25	869B	CPP14	正确	100	46ms	3.429MB
3114398	张子阳	张子阳	计算资源调度器	07-04 11:37	1.965KB	CPP14	正确	100	78ms	3.140MB
3113652	张子阳	张子阳	出行计划	07-04 09:19	505B	CPP14	正确	100	218ms	4.375MB
3113410	张子阳	张子阳	未初始化警告	07-04 08:50	368B	CPP14	正确	100	31ms	2.769MB

202203-1	未初始化警告	100	查看我的提交	查看试题/答题
202203-2	出行计划	100	查看我的提交	查看试题/答题
202203-3	计算资源调度器	100	查看我的提交	查看试题/答题
202203-4	通信系统管理	100	查看我的提交	查看试题/答题
202203-5	博弈论与石子合并	100	查看我的提交	查看试题/答题

T1 未初始化变量

解题思路:

1 （100pts）对题目描述进行模拟，可以将数组初始化为极大值（a[0]=0），执行语句每次执行赋值时，只需模拟赋值的操作，先判断右值是否合法（不为极大值），再将左值赋值为右值（若右值为极大值则赋值为 0）。统计答案即可，时间复杂度 O（n）

2:（100pts）模拟过程中不难发现，不需要开 int 整型数组，只需将已经初始化的数组元素标记为 1，未初始化的数组元素标记为 0，读入左右值时，先判断右值（统计答案），再把左值标记为 1（顺序不可调换），只需一个布尔数组即可实现。时间复杂度 O（n）。

代码展示（方法二）

```
#include<iostream>
#include<cstdio>
using namespace std;
inline int read(){
	int x=0,f=1;char c=getchar();
	for(;!isdigit(c);c=getchar()) c=='-'?f=-1:1;
	for(;isdigit(c);c=getchar()) x=x*10+c-'0';
	return x*f;
}

bool b[100600];
int main(){
	int k(read()),n(read()),ans=0;
```

```
	b[0]=1;
	while(n--){
		int x(read()),y(read());
		if(!b[y]) ans++;
		b[x]=1;
	}
	cout<<ans;
}
```

T2 出行计划

解题思路

1：(100pts) 在线处理，利用线段树或树状数组等数据结构，把[ti-ci+1,ti]整个区间+1，查询时只需判断[qi+k]处的值，每次区间加法和查询的复杂度为 logn，
总复杂度 O（(m+n）logn）
2：（100pts）离线处理，利用差分的思想，把 a[ti-ci+1]处+1，然后把 a[ti]处-1，进行查询时，只需预处理出前缀和便可以在 O (1) 的时间内查询。每次查询和处理的复杂度都是 O1，时间复杂度 O（m+n）

代码展示（方法二）

```
#include<iostream>
#include<cstdio>
using namespace std;
const int ss=2e5+777;
int sum[ss],c[ss],a[ss];
inline int read(){
	int x=0,f=1;char c=getchar();
	for(;!isdigit(c);c=getchar()) c=='-'?f=-1:1;
	for(;isdigit(c);c=getchar()) x=x*10+c-'0';
	return x*f;
}
int main(){
	int n(read()),m(read()),k(read());
	for(int i=1;i<=n;i++){
		int x(read()),t(read());
		a[max(x-t+1,1)]++;
		a[x+1]--;
	}
	for(int i=1;i<=200000;i++) sum[i]=sum[i-1]+a[i];
	while(m--){
		int t(read());
		printf("%d\n",sum[t+k]);
```

```
	}
}
```

T3 计算资源调度器

解题思路

1:（50pts）可以注意到前 10 个测试点对任务亲和性和反亲和性没有要求，因此值按照题意给每个节点打上标记，按照题意模拟即可，时间复杂度 O（n^2），但无法解决计算任务亲和性和非亲和性的问题。

2:（100pts）

预处理：设置结构体 e 存储编号和任务数量（用于排序阶段），bool 数组 now 判断当前节点是否能工作，map 映射任务编号对应的节点，Kuai 数组表示每个节点对应的可用区，使用 vector 数组记录每个可用区内的节点编号。

因为 fi 总数不超过 2000，直接进行枚举模拟：

节点亲和性：work1，把该节点对应工作区内的节点全部标记为可用，其余全部标记为不可用

任务亲和性：利用 map 数组 pan 映射任务编号对应的节点，再将没有这些节点的工作区内的所有节点全部标记为不可用

任务反亲和性：必须满足：利用 map 数组映射任务编号对应的节点，将这些节点标记为不可用；尽量满足：重复上述操作后如果不可用，就返回之前的状态。

记录任务节点：若此时没有可以使用的节点，则输出 0，否则按照题意进行排序，并统计任务次数，记录任务编号即可。

时间复杂度分析：由于总节点不超过 n，可用区不超过 m，任务总数不超过 2000（记为 n），执行一次工作的复杂度为 O（n）或 O（nlogn）（map 映射复杂度为 logn），故总时间复杂度为 O（n^2logn）。

代码展示（方法二）

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<map>
using namespace std;

inline int read(){
	int x=0,f=1;char c=getchar();
	for(;!isdigit(c);c=getchar()) c=='-'?f=-1:1;
	for(;isdigit(c);c=getchar()) x=x*10+c-'0';
	return x*f;
}
const int ss=3050;
struct oo{
```

```
	int id,sum;
}e[ss],s[ss];
int kuai[ss];
map<int,int> pan;
vector<int>q[ss],Re[ss];
int n,m;

bool now[ss],c[ss];
inline bool Pan(){
	for(int i=1;i<=n;i++) if(now[i]) return 1;
	return 0;
}
inline bool cmp(oo a,oo b){
	return a.sum==b.sum?a.id<b.id:a.sum<b.sum;
}
inline int quary(){
	int top(0);
	for(int i=1;i<=n;i++) if(now[i]) s[++top]=e[i];
	sort(s+1,s+top+1,cmp);
	return s[1].id;
}
inline void init(){
	for(int i=1;i<=n;i++) now[i]|=1;
}


inline void work1(int k){
	for(int i=1;i<=n;i++) now[i]=0;
	for(int i=0;i<q[k].size();i++) now[q[k][i]]|=1;
}
bool b[ss];
inline void work2(int k){
	int t=pan[k];
//	if(!t) return;

	for(int i=0;i<Re[t].size();i++){
		int K=kuai[Re[t][i]];b[K]=1;
	}
	for(int i=1;i<=m;i++)
	    if(!b[i]) for(int j=0;j<q[i].size();j++) now[q[i][j]]=0;
	for(int i=1;i<=m;i++) b[i]=0;
}
inline void work3(int k){
	int t=pan[k];
```

```
	if(!t) return;
	for(int i=0;i<Re[t].size();i++) now[Re[t][i]]=0;
}
inline void work4(int k){
	for(int i=1;i<=n;i++) c[i]=now[i];
	work3(k);if(!Pan())
	for(int i=1;i<=n;i++) now[i]=c[i];
}


int col=0;
inline int work5(int k){
	if(pan[k]) return pan[k];
	else pan[k]=++col;
	return col;
}//

int main(){
	n=(read()),m=(read());
	for(int i=1;i<=n;i++){
		int x(read());
		e[i]={i,0};
		kuai[i]=x;
		q[x].push_back(i);
	}
	int T(read());
	while(T--){
		int fi(read()),ai(read()),nai(read()),pai(read());
		int paai(read()),paari(read());
		for(int i=1;i<=fi;i++){
			if(nai) work1(nai);
			else init();
			if(pai) work2(pai);
			if(paai){
				if(paari) work3(paai);
				else work4(paai);
			}
			if(!Pan()) printf("0 ");
			else {
				int id=quary();printf("%d ",id);
				e[id].sum++;
				int Hao=work5(ai);
				Re[Hao].push_back(id);
			}
```

```
		}
		puts("");
	}
}
```

T4 通信系统管理

解题思路

1：（100pts）离线处理，将所有的数据读入后，本质上只需维护两种操作：修改单个数字和维护最大值。使用 set 或者大根堆，维护边，将修改操作看成删除操作和插入操作，维护 set。其余过程模拟即可。
复杂度分析：读入操作复杂度为 O（n），set 每次删边加边的复杂度约为 O（logn），故总时间复杂度为 O（nlogn）。
2：（100pts）在线处理，在读入过程中记录每一天的流量情况，立即进行对当天的处理。修改时先删除原边，后加入新边，根据记录前后是否含有通信对象来判断通信孤岛和通信对的数量，统计完成后，即可立即得出主通信对象以及实时的孤岛和通信对个数。
复杂度分析：读入操作复杂度为 O（n），set 每次删边加边的复杂度约为 O（logn），故总时间复杂度为 O（nlogn）。

代码展示（方法二）

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<set>
#include<map>
#define int long long
using namespace std;
inline int read(){
	int x=0,f=1;char c=getchar();
	for(;!isdigit(c);c=getchar()) c=='-'?f=-1:1;
	for(;isdigit(c);c=getchar()) x=x*10+c-'0';
	return x*f;
}
const int ss=2e5+777;
vector<int>V[ss];
struct oi{
	int fr,to,v,y;
}E[ss];
struct oo{
	int x,y,v;
};
struct ii{
```

```
	int to,v;
	friend bool operator<(ii a,ii b){
		return a.v==b.v?a.to<b.to:a.v>b.v;
	}
};

map<int,vector<oo> >d;
map<int,int>e[ss];


inline void add(int fr,int to,int v,int y,int cnt){
//	e[fr].push_back(to);
//	e[to].push_back(fr);
	E[cnt]={fr,to,v,y};
	d[cnt].push_back({fr,to,v});
	V[fr].push_back(0);
	d[cnt+y].push_back({fr,to,-v});
	V[to].push_back(0);
}
set<ii>edge[ss];
int n(read()),m(read());
int islend=0,pa=0;
inline void delet(int x,int y){
	edge[x].erase({y,e[x][y]});
	edge[y].erase({x,e[y][x]});
}
inline void add(int x,int y,int v){
	e[x][y]+=v;e[y][x]+=v;
	if(e[x][y]){
	edge[x].insert({y,e[x][y]});
	edge[y].insert({x,e[y][x]});
	}
}
inline void pan(int x,int y,int v){
	int px=0,py=0,nx=0,ny=0;
	if(edge[x].size()) px=(*edge[x].begin()).to;
	if(edge[y].size()) py=(*edge[y].begin()).to;
	nx=px?((*edge[px].begin()).to==x):0;
	ny=py?((*edge[py].begin()).to==y):0;
	if(e[x][y]) delet(x,y);
	add(x,y,v);
	if(px&&!edge[x].size()) islend++;
	if(py&&!edge[y].size()) islend++;
	if(!px&&edge[x].size()) islend--;
```

```
	if(!py&&edge[y].size()) islend--;//统计孤岛


	int nowx=0,nowy=0,n2x=0,n2y=0;
	if(edge[x].size()) nowx=(*edge[x].begin()).to;
	if(edge[y].size()) nowy=(*edge[y].begin()).to;
	n2x=nowx?((*edge[nowx].begin()).to==x):0;
	n2y=nowy?((*edge[nowy].begin()).to==y):0;
	if(!nx&&n2x) pa++;
	if(nx&&!n2x) pa--;
	if(!ny&&n2y&&nowy!=x) pa++;
	if(ny&&!n2y&&py!=x) pa--;
	if(ny&&n2y&&py==x&&nowy!=x) pa++;
	if(ny&&n2y&&py!=x&&nowy==x) pa--;//统计点对
}
inline void work(int day,int k){
	for(int i=0;i<d[day].size();i++){
		int x(d[day][i].x),y(d[day][i].y),v(d[day][i].v);
		pan(x,y,v);
	}
}
main(){
	islend=n;
	for(int i=1;i<=m;i++){
		int k=read();
		for(int j=1;j<=k;j++){
			int fr(read()),to(read()),v(read()),y(read());
			add(fr,to,v,y,i);
		}
		work(i,k);
		int l=read();
		for(int j=1;j<=l;j++){
			int x(read());
			if(!edge[x].size()) puts("0");
			else printf("%lld\n",(*edge[x].begin()).to);
		}
		int p(read());
		if(p) printf("%lld\n",islend);
		int q(read());
		if(q) printf("%lld\n",pa);
	}
}
```

T5 博弈论与石子合并

解题思路

1：（100pts）通过分析可得，无论是移去石子还是合并石子，每次操作石子都会减少一堆。所以两人的策略等价为进行最后一次操作时让石子堆最小/最大。
如果小 c 先手且石子有奇数堆，和小 z 先手石子有偶数堆等价（因为小 z 先手一定会合并一堆，这样又回到了小 c 先手石子偶数堆的情况）。故在此只讨论小 c 先手石子奇数堆的情况。
若只有一堆，则游戏结束
若石子有三堆，则小 c 需要移去左右侧最大的一堆。
若石子超过三堆，则小 c 每次会尽量将最左右两侧最大的一堆去掉，小 z 每次则会把最小堆合并，最后剩余的便是相邻的（n+1）/2 堆石子，且这堆石子最小。

如果小 c 先手且石子有偶数堆，和小 z 先手且石子有奇数堆两者等价（与上同理）。
若石子有一堆，游戏结束
若石子有三堆，小 z 会合并左右两侧较小的一堆，此时小 c 只能去掉较大的一堆石子，最后的结果是两堆石子中较小的一堆
当石子超过三堆时，最后的结果是小 z 会尽可能使得最后留下来的石子更多，此时只需找到最大的石子数满足上述条件即可。（可以使用二分）。
复杂度分析：情况 1 寻找答案的复杂度为 O（n），情况 2 寻找答案的复杂度为 O（nlogn）所以总复杂度为 O（nlogn）
2（100pts）对于条件 2，我们可以使用前缀和记录前 n 堆石子数，因为前缀和一定是单调的，我们每次统计寻找下标时，只需要利用多次 lower_bound 二分查找即可。
复杂度分析：每次二分查找的复杂度为 O（clogn），c 为常数，故第二种情况的复杂度为 O（$clog^2n$），理论上小于 O（nlogn）和 O（n），但第一种情况的时间复杂度依然为 O（n）故总时间复杂度为 O（n）

代码展示（方法二）

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<set>
#include<map>

using namespace std;
inline int read(){
	int x=0,f=1;char c=getchar();
	for(;!isdigit(c);c=getchar()) c=='-'?f=-1:1;
	for(;isdigit(c);c=getchar()) x=x*10+c-'0';
	return x*f;
}
const int ss=2e5+888;
int n,k,sum[ss],a[ss];
inline bool pan(int k){
```

```
	int head=0,num=0;
	while(head<=n){
		head=lower_bound(sum+1,sum+n+1,sum[head]+k)-sum;
		if(head<=n) num++;
	}
	return num>n/2;
}

int main(){
	n=read();k=read();
	int ans=1060633226;
	for(int i=1;i<=n;i++) a[i]=read(),sum[i]=sum[i-1]+a[i];
	if(k!=n%2){
		for(int i=1;i<=n;i++){
			int j=i+n/2;
			if(j>n) break;
			ans=min(ans,sum[j]-sum[i-1]);
		}
	}
	else {
		int l=1,r=ans;
		while(l<=r){
			int mid=l+r>>1;
			if(pan(mid)) l=mid+1,ans=mid;
			else r=mid-1;
		}
	}
	cout<<ans;
}
```