# 2023.8.2 数据结构上

## 训练

---

## 1. Range Sums

---

【问题描述】
给定一个数组，数组长度为n，以及q次输入。每次输入包含两个整数l和r，表示我们知道数组中[l, r]区间的元素和。现在要求判断是否可以根据这些已知区间的和，最终确定整个数组的元素和。

如果可以确定数组的元素和，则输出"Yes"，否则输出"No"。

【链接】https://www.luogu.com.cn/problem/AT_abc238_e

【输入格式】
第一行包含两个整数n和q，表示数组长度和查询次数。
接下来q行，每行包含两个整数l和r，表示我们知道数组中[l, r]区间的元素和。

【输出格式】
输出q行，每行一个字符串，表示对应查询的结果，如果可以确定数组的元素和则输出"Yes"，否则输出"No"。

【输入样例】
3 3
1 2
2 3
2 2

【输出样例】
Yes

【数据范围】
约束条件：
$1 \leqslant n \leqslant 2 \times 10^5$
$1 \leqslant q \leqslant \min(2 \times 10^5, n(n+1)/2)$
$1 \leqslant l \leqslant r \leqslant n$
$(l, r) \neq (l', r')$ $(l \neq l'$ or $r \neq r')$
输入为整数。

### 题解

$l$ 到 $r$ 的区间和，很容易让人想到前缀和。

而正向求区间和的公式是：

$ans = s_r - s_{l-1}$，其中 $s$ 为前缀和数组。

在往反方向想，

$s_{l-1} = s_r - ans$，所以对于题目给定的 $ans$，用 $s_r$ 可以求出 $s_{l-1}$。

这个特点就很熟悉了，连一条边，然后判断 0 和 $n$ 是否在同一个联通块中，如果在，那么就求得出 $s_n - s_0$，也就是所有数的和。联通块直接用并查集即可。

### 代码

```cpp
#include <bits/stdc++.h>using namespace std;
int n,q,l,r,fa[2000005];
int find(int x){
  return fa[x]==x?x:fa[x]=find(fa[x]); //并查集压缩路径
}
int main(){
  cin>>n>>q;
  for(int i=0;i<=n+3;i++)
    fa[i]=i;
  for(int i=1;i<=q;i++)
  {
    cin>>l>>r;
    fa[find(r)]=find(l-1); //连边
  }
  cout<<(find(0)==find(n)?"Yes":"No")<<'\n'; //判断return 0;
}
```

## 2. Coins

【问题描述】

有x+y+z个人，第i个人有Ai个金币，Bi个银币，Ci个铜币。选出x个人获得其金币，选出y个人获得其银币，选出z个人获得其铜币，在不重复选某个人的情况下，最大化获得的币的总数。

【链接】

https://www.luogu.com.cn/problem/AT_agc018_c

【输入格式】

第一行包含三个整数x、y和z，表示选取金币、银币和铜币的人数。

接下来x+y+z行，每行包含三个整数Ai、Bi和Ci，表示第i个人分别拥有的金币、银币和铜币数量。

【输出格式】

输出一个整数，表示在满足条件的前提下，选取x个人获得金币，选取y个人获得银币，选取z个人获得铜币所能获得的最大币的总数。

【输入样例】

1 2 1

2 4 4

3 2 1

7 6 7

5 2 3

【输出样例】

18

【输入样例】

3 3 2

16 17 1

2 7 5

2 16 12

17 7 7

13 2 10

12 18 3

16 15 19

5 6 2

【输出样例】

110

【说明/提示】

约束条件：

1 ⩽ X ⩽ 10^5

1 ⩽ Y ⩽ 10^5

1 ⩽ Z ⩽ 10^5

X+Y+Z ⩽ 10^5

1 ⩽ Ai ⩽ 10^9

1 ⩽ Bi ⩽ 10^9

1 ⩽ Ci ⩽ 10^9

## 题解

考虑只有两种权值 ⊠,⊠ 的时候我们怎么做，我们会先假设所有人都选 ⊠，然后把所有人按 ⊠⊠−⊠⊠ 从大到小排序，选前 ⊠ 个人从 ⊠ 变成 ⊠。

这个贪心过程的正确性显然，但却不是很好严谨地证明。但是我们考虑这样一种表述方式：

“将所有人按照 ⊠⊠−⊠⊠ 从大到小排序，那么所有选择 ⊠ 的人一定排在所有选择 ⊠ 的人左边。”

这个的证明非常简单明了：如果有左 ⊠ 右 ⊠ 的对，把他们交换，显然很优！

而这个证明的简单明了，意味着它的做法可以放到这道题上面。

具体地，我们仍然把所有人按 ⊠⊠−⊠⊠ 从大到小排序，那么所有选择 ⊠ 的人依然一定排在所有选择 ⊠ 的人左边，尽管不是所有人都选了 ⊠ 或 ⊠。

于是一定存在一个分界点 ⊠，在 ⊠ 左边的所有人选的都是 ⊠ 或 ⊠，在 ⊠ 右边的所有人选的都是 c 或 ⊠。

两边分别拿个对顶堆维护即可。时间复杂度 ⊠(⊠log⊠)。

## 代码

```
#include <cstdio>#include <algorithm>#include <queue>#define debug(...) fprintf(

struct {inline operator int () { int x; return scanf("%d", &x), x; }
        template<class T> inline void operator () (T &x) { x = *this; }
        template<class T, class ...A> inline void operator () (T &x, A &...a){ x
} read;

const int maxn = 100005;
struct obj {int a, b;
};
obj ob[maxn];
ll lget[maxn], rget[maxn];

int main() {
        int x = read, y = read, z = read;

        int n = x + y + z;
        ll ans = 0;
        for(int i = 1; i <= n; i ++) {
```

```
            int v = read;
            ans += v;
            ob[i].a = read - v;
            ob[i].b = read - v;
        }

        std::sort(ob + 1, ob + n + 1, [](obj x, obj y) {
                        return x.a - x.b > y.a - y.b;
                    });

        std::priority_queue<int, std::vector<int>, std::greater<int> > biggest;

        for(int i = 1; i <= n; i ++) {
            lget[i] = lget[i - 1] + ob[i].a;
            biggest.push(ob[i].a);
            if(int(biggest.size()) > y) {
                lget[i] -= biggest.top();
                biggest.pop();
            }
        }

        while(!biggest.empty()) biggest.pop();
        for(int i = n; i; i --) {
            rget[i] = rget[i + 1] + ob[i].b;
            biggest.push(ob[i].b);
            if(int(biggest.size()) > z) {
                rget[i] -= biggest.top();
                biggest.pop();
            }
        }

        ll max = - 1000000000000000000;
        for(int k = y; k <= n - z; k ++)
            max = std::max(max, lget[k] + rget[k + 1]);

        printf("%lld\n", ans + max);
}
```

## 3. Ball Collector

【问题描述】

有一棵N个点的树，每个顶点i上有两个球，一个写着Ai，一个写着Bi。树共有N-1条边，对于每条边i连接点Ui和Vi。接着，给定N-1次互相独立的询问：当v=2,3，⋯，N时，求点1到点v的最短路径，这条路

径（包含1和v）所经过的点i，必须选择Ai和Bi两个小球中的一个。求问每次操作最多能选几个标数不同的小球。

【输入格式】

第一行包含一个整数N，表示树的节点个数。

接下来N-1行，每行包含两个整数Ui和Vi，表示树的边连接情况。

接下来N-1行，每行包含两个整数Ai和Bi，表示每个顶点i上写着的两个球的标号。

【输出格式】

输出一行，包含N-1个整数，表示对于每次询问（v=2,3，⋯，N），最多能选几个标数不同的小球。

【输入样例】

4

1 2

2 3

3 1

1 2

1 2

2 3

3 4

【输出样例】

2 3 3

【说明/提示】

约束条件：

$2 \leqslant N \leqslant 2 \times 10^5$

$1 \leqslant Ai, Bi \leqslant N$

给定的图是一棵树。

输入为整数。

## 题解

考虑建图，对每个 $(a_i, b_i)$ 建一条无向边，那么问题就变成了：对于每条边都要选择它以及其连接的一个点，最大化选出的点数。

很明显可以对每个连通块分开考虑。

记当前连通块的点数为 V，边数为 E。那么有结论：该连通块对答案的贡献为 min(V, E)。

考虑证明。由于是连通块，所以 E 最小为 V-1（树）。接下来我们根据 V 和 E 的关系分类讨论：

1. V = E = V-1。此时连通块为一棵树。随便钦定一个点为根，然后每条边选儿子，这样就可以选掉除了根节点外的所有点，答案为 V-1 = E = min(V, E)。很明显选出来的点数不可能比选的边数还多，所以这是对的。
2. V ⩾ E ⩾ V。这个时候必然能给每个点都选出一条边，答案为 min(V, E) = V。

这样我们就证完了。

具体实现的时候使用并查集，维护连通块内点数、边数，合并时分类讨论即可。

代码实现时，接下来考虑树上的每个点，我们在 DFS 的时候把当前点 u 加入连通块中并计算答案，搜索完 u 时再消除 u 对答案的影响即可。

怎么消除影响？使用可撤销并查集，将每次合并压入栈中，撤销时取出栈中信息并复原即可。

注意此时并查集不能使用路径压缩，因为这样会破坏树的结构。使用启发式合并即可。时间复杂度 O(n log n)，可以通过此题。

## 代码

```cpp
#include<bits/stdc++.h>
//#pragma GCC optimize("Ofast")
#define gt getchar
#define pt putchar
//typedef unsigned int uint;
typedef long long ll;
typedef unsigned long long ull;
//typedef __int128 lll;
//typedef __uint128_t ulll;
const int N=2e5+5;
using namespace std;
inline bool __(char ch){return ch>=48&&ch<=57;}
inline int read(){
        int x=0;bool sgn=0;char ch=gt();
        while(!__(ch)&&ch!=EOF) sgn|=(ch=='-'),ch=gt();
        while(__(ch)) x=(x<<1)+(x<<3)+(ch&15),ch=gt();
        return sgn?-x:x;
}
template<class T> inline void print(T x){
        static char st[70];short top=0;
        if(x<0)pt('-');
         do{st[++top]=x>=0?(x%10+48):(-(x%10)+48),x/=10;}while(x);
    while(top) pt(st[top--]);
}
template<class T> inline void printsp(T x){
        print(x);
        putchar(' ');
}
```

```
template<class T> inline void println(T x){
        print(x);
        putchar('\n');
}
inline void put_str(string s){
        int siz=s.size();
        for(int i=0;i<siz;++i) pt(s[i]);
        printf("\n");
}
struct edge{
        int to,nxt;
}e[N<<1];
int head[N],ecnt,n,a[N],b[N];
inline void add_edge(int f,int t){
        e[++ecnt].to=t;
        e[ecnt].nxt=head[f];
        head[f]=ecnt;
}
inline void add_double(int f,int t){
        add_edge(f,t);
        add_edge(t,f);
}
int anss,fa[N],siz[N],cnt[N],ans[N];
struct Node{
        int opt,x,y;
}stk[N];
int top;
inline int get_ans(int x){
        return min(siz[x],cnt[x]);
}
inline int find(int x){
        while(x!=fa[x]) x=fa[x];
        return fa[x];
}
inline void link(int x,int y){
        int xx=find(x),yy=find(y);
        if(xx==yy){
                anss-=get_ans(xx);
                cnt[xx]++;
                anss+=get_ans(xx);
                stk[++top]={0,xx,0};
        }else{
                anss-=get_ans(xx);
                anss-=get_ans(yy);
                if(siz[yy]>siz[xx]) swap(xx,yy);
                fa[yy]=fa[xx],siz[xx]+=siz[yy],cnt[xx]+=cnt[yy]+1;
                anss+=get_ans(xx);
```

```
			stk[++top]={1,xx,yy};
		}
}
inline void ret(){
	int x=stk[top].x,y=stk[top].y;
	if(stk[top].opt==0){
		anss-=get_ans(x);
		cnt[x]--;
		anss+=get_ans(x);
	}else{
		anss-=get_ans(x);
		fa[y]=y,siz[x]-=siz[y],cnt[x]-=(cnt[y]+1);
		anss+=get_ans(x);
		anss+=get_ans(y);
	}
	top--;
}
void dfs(int u,int fa){
	link(a[u],b[u]);
	ans[u]=anss;
	for(int i=head[u];i;i=e[i].nxt){
		int v=e[i].to;
		if(v!=fa) dfs(v,u);
	}
	ret();
}
signed main(){
	n=read();
	for(int i=1;i<=n;++i){
		a[i]=read(),b[i]=read();
		fa[a[i]]=a[i],fa[b[i]]=b[i];
		siz[a[i]]=siz[b[i]]=1;
		cnt[a[i]]=cnt[b[i]]=0;
	}
	for(int i=1;i<n;++i){
		int u=read(),v=read();
		add_double(u,v);
	}
	dfs(1,0);
	for(int i=2;i<=n;++i) printsp(ans[i]);
	return 0;
}
```

对于全部的测试点，保证 $1 \leqslant M \leqslant 2 \times 10^5$，$1 \leqslant D \leqslant 2 \times 10^9$。

## 题解

我们发现，在需要维护区间最值的时候经常使用单调栈，尤其是这道题没有删除，并且操作只在序列末端。

维护一个单调递减栈，最后一个数最后加入，所以一定在栈内。而如果查询后两个的话，如果末尾最大显然好说，如果倒数第二个最大的话就可以查到这个最大的元素。

**考虑具体开头/末尾 三个数/两个数的情况是一种很好的考虑单调栈和单调队列的做法，可以假设之间元素的大小关系然后确定做法。**

查询使用二分，在单调栈的元素中记录编号，对答案进行二分，由于栈递减，所以查到的第一个编号在后x个的元素就是答案。

## 代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#define ll long long
using namespace std;
struct node
{
 ll id,v;
}st[200010];
ll t,md,m,o,ans,l,r,mid,n,lastans;
int main ()
{
 scanf("%lld %lld",&m,&md);
 while (m--)
 {
  char ch[10];
  scanf("%s",ch);
  if (ch[0]=='A')
  {
   n++;
   scanf("%lld",&o);
   o=(o%md+lastans)%md;
   while (o>st[t].v&&t) t--;
   st[++t].v=o;
   st[t].id=n;
  }
  else
  {
   l=1,r=t,ans=1;
```

```
     scanf("%lld",&o);
     while (l<=r)
     {
      mid=(l+r)>>1;
      if (st[mid].id>=n-o+1)
      {
       r=mid-1,ans=mid;
      }
      else l=mid+1;
     }
     printf("%lld\n",st[ans].v);
     lastans=st[ans].v;
    }
  }
}
```