2023.8.5 进阶数据结构 题单

1. FT robot

【问题描述】

题意描述： 现有一个机器人在坐标轴原点处，朝向x轴正方向。给定一个命令串s，s内含有两种命令：F,T。F表示向当前朝向前进1步，T表示向左**或**向右旋转90度。请你回答，按照命令串s执行，会不会到达(x,y)处。（x与y由题目输入）

x和y，以及字符串长度不超过8000

【链接】https://www.luogu.com.cn/problem/AT3726

【输入格式】

第一行输入一些F和T

第二行输入x和y

【输出格式】

输出Yes或No

【样例输入】

FTFFTFFF
4 2

【样例输出】

Yes

题解

核心思路是对于横坐标和纵坐标分别讨论，因为横纵坐标是独立的

以横坐标为例，我们把一段段的 F 的长度记录下来。假设已经遍历完了的最近一段 F 长度为 w。我们用一个 bool 值 dp[i][j] 代表这次操作前能否达到第 j 位，那么这一次操作可以是减少横坐标，也可以是增加横坐标。所以我们就有 dp[i][j] = dp[i-1][j+w] or dp[i-1][j-w]，其中 or 是逻辑或运算。然后这个东西就是一堆逻辑或运算，所以我们可以用滚动数组优化和 bitset 优化，直接将一次转移写成 x = x << w | x >> w。

最后就是判断所有操作后能否同时达到最终的横纵坐标。

时间复杂度 O(n^2/32)，可以通过此题。

代码

```cpp
#include <iostream>
#include <cstdio>
#include <bitset>
#define M 8005
using namespace std;

int x, y, z, c;
string s;
bitset<M * 2> bs[2];
int main() {
        int i;
        cin >> s >> x >> y;
        for (i = 0; s[i] == 'F'; i++);
        bs[0][i + M] = bs[1][M] = 1;
        for (; i <= s.size(); i++) {
                if (s[i] == 'F') c++;
                else bs[z] = (bs[z] << c) | (bs[z] >> c), z = !z, c = 0;
        }
        cout << (bs[0][x + M] & bs[1][y + M] ? "Yes" : "No");
        return 0;
}
```

2. Median of Medians

【问题描述】

定义一个长度为 M 的序列的中位数为这个序列中第 ⌊M/2⌋+1 小的数。

现在有一个长度为 N的序列 A，将 A 的所有子段的中位数取出来作为一个序列 S，问序列 S 的中位数是多少。

N不超过1e5，数字范围1e9

【链接】https://www.luogu.com.cn/problem/AT4351

【输入格式】

先给出一个数字N

接下来输入N个数字

【输出格式】

输出答案

【样例输入】

3
10 30 20

【样例输出】

30

题解

二分答案一个数，然后想办法求出中位数小于等于这个数字的区间有多少个。

考虑将所有小于mid的都改成-1，所有大于的改成1，那么区间和小于0的区间就是中位数小于mid的区间。

区间和小于0的可以用树状数组快速维护。

代码

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 2e5 + 5;
inline int read(int x = 0) { return scanf("%d", &x), x; }
int n, a[N], c[N], b[N], sum[N], d[N];
ll cnt, value;
inline int lowbit(int x) { return x & -x; }
inline void add(int p) {
        while(p <= 2*n + 1) { c[p]++, p += lowbit(p); }
}
inline ll get(int p, ll tmp = 0) {
        while(p) { tmp += c[p], p -= lowbit(p); }
        return tmp;
}
bool pan(int mid) {
        cnt = 0;
        for(register int i = 1; i <= n; ++i)
                b[i] = a[i] > mid ? -1 : 1, sum[i] = sum[i-1] + b[i];
        memset(c, 0, sizeof(c));
        add(n+1);
        for(register int i = 1; i <= n; ++i) {
                cnt += get(sum[i] + n), add(sum[i] + n + 1);
        }
        return cnt >= value;
}
int main() {
        n = read(), value = 1ll*n*(n+1)/4+1;
        for(int i = 1; i <= n; ++i) d[i] = a[i] = read();
        sort(d + 1, d + n + 1);
        int l = 1, r = n, mid = 0;
        while(l <= r) {
                mid = (l + r) / 2;
                if(pan(d[mid])) r = mid - 1;
```

```
35                  else l = mid + 1;
36          }
37          cout << d[l] << '\n';
38          return 0;
39  }
```

3. Meaningful Mean

【问题描述】

给一个长度为N的整数序列{a},对于其一共N*(N+1)/2个的非空区间，求有多少个区间的平均数大于等于K。

1 ⩽ N ⩽ 2×10^5

1 ⩽ K ⩽ 10^9

1 ⩽ a[i] ⩽ 10^9

【链接】https://www.luogu.com.cn/problem/AT2581

【输入格式】

第一行两个数字分别为 N K 接下来N行 每行一个数字表示a[i]

【输出格式】

输出一个整数 代表有多少个区间的平均数大于等于K

【样例输入】

3 6
7
5
7

【样例输出】

5

题解

$$\frac{sum_r - sum_{l-1}}{r-l+1} \geq k$$

$$k \times (r-l+1) \leq sum_r - sum_{l-1}$$

$$sum_r - r \times k \geq sum_{l-1} - (l-1) \times k$$

所以用树状数组统计即可。

代码

```cpp
#include <cstdio>
#include <algorithm>using namespace std;

typedef long long ll;

const int maxn=2e5+50;

int n,k,N,r[maxn];
ll ans,a[maxn],sum[maxn];

inline int read(int x=0,char c=getchar()) {
        while(c<'0'||c>'9') c=getchar();
        while(c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
        return x;
}

ll c[maxn];

inline void add(int x,int v){for(;x<=N;x+=x&-x)c[x]+=v;}
inline int ask(int x){int y=0;for(;x;x-=x&-x)y+=c[x];return y;}

int main() {
        n=read();k=read();
        for(register int i=1;i<=n;++i) a[i]=read()-k;
        for(register int i=1;i<=n;++i) if((sum[i]=(a[i]+=a[i-1]))>=0) ++ans;
        sort(sum+1,sum+1+n); N=unique(sum+1,sum+1+n)-sum-1;
        for(register int i=1;i<=n;++i) r[i]=lower_bound(sum+1,sum+1+N,a[i])-sum;
        for(register int i=1;i<=n;++i) ans+=ask(r[i]),add(r[i],1);
        printf("%lld",ans);
        return 0;
```

```
31 }
```

4. Points

【问题描述】

在一个笛卡尔坐标系中，定义三种操作:

add x y :在坐标系上标记一个点 (x,y)，保证 (x,y) 在添加前不在坐标系上。

remove x y :移除点 (x,y)，保证 (x,y) 在移除前已在坐标系上。

find x y :找到所有已标记并在 (x,y) 右上方的点中，最左边的点，若点不唯一，选择最下面的一个点; 如果没有符合要求的点，给出 -1，否则给出 (x,y)。

现有 n 个操作,对于每个 find 操作，输出结果。

N不超2e5，x和y在1e9之内

【输入】

第一行输入操作的数量

加下来每行输入一个操作

【输出】

对于每个询问，输出答案

【链接】

https://www.luogu.com.cn/problem/CF19D

【输入】

7

add 1 1

add 3 4

find 0 0

remove 1 1

find 0 0

add 1 1

find 0 0

【输出】

1 1

3 4

1 1

方法1

这是一个二维的数据结构问题。问题主要出在第三种操作。

可以对横坐标离散化，然后每个横坐标开一个set记录横坐标为x的点集。

问题转化为我们要找到最靠左边的一个x，使得它的点集中有点在y之上。

对于每x只需要知道最大的一个y就可以确定每个x是否有大于某个的点了。

于是我们用线段树维护每个x中y的最大值，查询的时候，首先看看左子树有没有，如果有直接返回，否则返回右子树的答案。

代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<map>
#include<set>
using namespace std;
#define ll long long
#define ls (rt<<1)
#define rs (rt<<1|1)
#define pb(x) push_back(x)
#define pr(x) cerr<<#x<<"="<<x<<endl
#define pri(x,lo) {cerr<<#x<<"={";for (int ol=0;ol<=lo;ol++)cerr<<x[ol]<<",";cer
#define N 8000100
struct node {int op,o,p;}q[N];
set<int> s[N];
int n,i,ex[N],cnt,size,mx[N];
char ch[100];
inline void init()
{
 sort(ex+1,ex+1+cnt);
        size=unique(ex+1,ex+1+cnt)-ex-1;
 for (i=1;i<=n;i++) q[i].o=lower_bound(ex+1,ex+1+size,q[i].o)-ex;
}
```

```
void modify(int rt,int l,int r,int x,int y,int ty)
{
 if (l==r) {if (ty==1) mx[rt]=max(mx[rt],y);else mx[rt]=y;return ;}
 int mid=(l+r)/2;
 if (x<=mid) modify(ls,l,mid,x,y,ty);
 else modify(rs,mid+1,r,x,y,ty);
        mx[rt]=max(mx[ls],mx[rs]);
}
int solve(int rt,int l,int r,int x,int y)
{
 if (x>r) return -1;
 if (mx[rt]<y) return -1;
 if (l==r) return l;
 int mid=(l+r)/2;
 int tmp=solve(ls,l,mid,x,y);
 if (tmp!=-1) return tmp;
 else return solve(rs,mid+1,r,x,y);
}
int main()
{
freopen("a.in","r",stdin);
 freopen("a.out","w",stdout);
 scanf("%d",&n);
 for (i=1;i<=n;i++)
        {
  scanf("%s",ch);
  if (ch[0]=='a') q[i].op=1;else if (ch[0]=='r') q[i].op=2;else q[i].op=3;
  scanf("%d %d",&q[i].o,&q[i].p);ex[++cnt]=q[i].o;//ex1[++cnt1]=q[i].p;
        }
 init();
 for (i=1;i<=n;i++)
        {
  if (q[i].op==1)
                {
   modify(1,1,size,q[i].o,q[i].p,1);
                        s[q[i].o].insert(q[i].p);
   //pr(s[2].size());
                }
  else if (q[i].op==2)
                {
                        s[q[i].o].erase(q[i].p);
                        set<int>::iterator y=s[q[i].o].end();
   int tmp=0;
   if (y!=s[q[i].o].end()) y--;tmp=*y;
   modify(1,1,size,q[i].o,tmp,2);
                }
  else
```

```
                {
    int x=solve(1,1,size,q[i].o+1,q[i].p+1);
    if (x==-1) {printf("-1\n");continue;}
                        set<int>::iterator y=s[x].upper_bound(q[i].p);
    if (y==s[x].end()) {printf("-1\n");continue;}
    printf("%d %d\n",ex[x],*y);
                }
        }
 return 0;
}
```

方法2：

有一种多一个log的做法。

从纵坐标的角度考虑，我们想要得到的是所有的那些纵坐标大于y的点中，横坐标大于x的点。

那么如果对y轴离散化，就可以用树状数组来保存当前一段范围纵坐标的点集了。

于是可以对y轴建立树状数组，每次找出那些点集中大于x的最小的x，这个点就是答案的横坐标。但是由于我们并不是直接查询的纵坐标，所以并不知道是哪个y。但是知道了x之后可以在横坐标为x的点集中二分。

复杂度$O(n\log^2 n)$

代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<map>
#include<set>
using namespace std;
#define ll long long
#define pb(x) push_back(x)
#define pr(x) cerr<<#x<<"="<<x<<endl
#define pri(x,lo) {cerr<<#x<<"={";for (int ol=0;ol<=lo;ol++)cerr<<x[ol]<<",";cer
#define N 10001000
struct node {int op,o,p;}q[N];
int n,i,cnt1,cnt2,ex1[N],ex2[N],size1,size2;
set<int> c[N],s[N];
```

```
char ch[N];
void init()
{
 sort(ex1+1,ex1+1+cnt1);
        size1=unique(ex1+1,ex1+1+cnt1)-ex1-1;
 for (i=1;i<=n;i++) q[i].o=lower_bound(ex1+1,ex1+1+size1,q[i].o)-ex1;
 sort(ex2+1,ex2+1+cnt2);
        size2=unique(ex2+1,ex2+1+cnt2)-ex2-1;
 for (i=1;i<=n;i++) q[i].p=lower_bound(ex2+1,ex2+1+size2,q[i].p)-ex2;
}
int main()
{
freopen("a.in","r",stdin);
 freopen("a.out","w",stdout);
 scanf("%d",&n);
 for (i=1;i<=n;i++)
        {
  scanf("%s",ch);
  if (ch[0]=='a') q[i].op=1;else if (ch[0]=='r') q[i].op=2;else q[i].op=3;
  scanf("%d %d",&q[i].o,&q[i].p);ex1[++cnt1]=q[i].o;ex2[++cnt2]=q[i].p;
        }
 init();
 for (i=1;i<=n;i++)
        {
  if (q[i].op!=3)
                {
   if (q[i].op==1) s[q[i].o].insert(q[i].p);
   else s[q[i].o].erase(q[i].p);
   int x=q[i].p;
   while(x>0)
                        {
    if (q[i].op==1)c[x].insert(q[i].o);
    else c[x].erase(q[i].o);
                                x-=(x&-x);
                        }
                }
  else
                {
   int x=q[i].p+1;
   int minn=10000000;
   while (x<=size2)
                        {
                                set<int>::iterator tmp=c[x].upper_bound(q[i].o);
    if (tmp!=c[x].end()) minn=min(minn,*tmp);
                                x+=(x&-x);
                        }
   if (minn==10000000) {printf("-1\n");continue;}
```

```
                        set<int>::iterator y=s[minn].upper_bound(q[i].p);
   printf("%d %d\n",ex1[minn],ex2[*y]);
                }
        }
                cerr<<clock()*1.0/CLOCKS_PER_SEC;
}
```

方法3：

这个做法比较好理解，其实是一种暴力

我们对所有的点用map离散化，按照（x，y）排序。

然后对于所有的点建立线段树，每个节点记录区间内的最大横坐标和最大纵坐标。

那么查询的时候如果当前区间的最大横坐标小于需要的横坐标或者最大纵坐标小于需要的纵坐标，那么就返回。找到的第一个合法的叶子节点就是答案。

代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<map>
#include<set>
using namespace std;
#define ll long long
#define pb(x) push_back(x)
#define pr(x) cerr<<#x<<"="<<x<<endl
#define pri(x,lo) {cerr<<#x<<"={";for (int ol=0;ol<=lo;ol++)cerr<<x[ol]<<",";cer
#define N 10000010
int maxx[N],maxy[N],op[N],x[N],y[N],n,i,bas;
typedef pair<int,int> pa;
map<pa,int> mp;
char ch[N];
void update(int x)
{
 for (x/=2;x>=1;x/=2) maxx[x]=max(maxx[x*2],maxx[x*2+1]),maxy[x]=max(maxy[x*2],m
}
int find(int rt,int x,int y)
{
```

```
 if (maxx[rt]<x||maxy[rt]<y) return -1;
 if (rt>bas) return rt;
 int tmp=find(rt*2,x,y);
 if (tmp!=-1) return tmp;
 return find(rt*2+1,x,y);
}
int main()
{
 freopen("a.in","r",stdin);
 freopen("a.out","w",stdout);
 scanf("%d",&n);
 for (i=1;i<=n;i++)
        {
  scanf("%s",ch);
  if (ch[0]=='a') op[i]=1;else if (ch[0]=='r') op[i]=2;else op[i]=3;
  scanf("%d %d",&x[i],&y[i]),mp[make_pair(x[i],y[i])]=0;
        }
 int cnt=0;
 for (map<pa,int>::iterator it=mp.begin();it!=mp.end();it++) it->second=++cnt;
        bas=(1<<21);
 for (i=1;i<=n;i++)
        {
  int id=mp[make_pair(x[i],y[i])];
  //pr(id);
  if (op[i]==1)
                {
                        maxx[bas+id]=x[i];
                        maxy[bas+id]=y[i];
   update(bas+id);
                }
  else if (op[i]==2)
                {
                        maxx[bas+id]=0;
                        maxy[bas+id]=0;
   update(bas+id);
                }
  else
                {
   int tmp=find(1,x[i]+1,y[i]+1);
   if (tmp==-1) printf("-1\n");
   else printf("%d %d\n",maxx[tmp],maxy[tmp]);
                }
        }
        cerr<<clock()*1.0/CLOCKS_PER_SEC;
}
```

5. Water tree

【问题描述】

疯狂科学家 Mike 构建了一棵有根的树，它由 n 个顶点组成。每个顶点都是一个水库，它可以是空的，也可以是充满水的。树的顶点从1到 n 编号，根在顶点1。对于每个顶点，其子节点的水库位于这个顶点的水库之下，这个顶点与每个子节点通过一个管道连接，水可以通过这个管道向下流动。

Mike 想对这个树做以下几种操作:

1．用水填充顶点 v。这样 v 和它所有的子节点都被水填满了。

2.排空顶点 v 。这样 v 和它所有的祖先都被清空了。

3.确定此刻 v 顶点是否被水填满了。

最初树的所有顶点都是空的。 Mike 已经想了一个他想要按顺序执行的操作的完整列表。他决定通过一个程序来运行这个列表。帮助 Mike 确定他做完所有操作后会得到什么结果。

【输入格式】

输入的第一行包含一个整数 n (1≤ n ≤500000)-树中的顶点数。下面的 n-1行每一行包含两个空格分隔的数字 ai，bi (1≤ ai，bi ≤ n，ai ≠ bi)树的边。下一行包含一个数字 q (1≤ q ≤500000)要执行的操作数。以下每条行包含两个空间分隔的数字 ci (1≤ ci ≤3) ，vi (1≤ vi ≤ n) ，其中 ci 是操作类型(根据语句中给出的编号) ，vi 是执行操作的顶点。保证给定的图是树。

【输出格式】

对于每个类型3的操作，如果顶点已满，则在单独的行上打印1，如果顶点为空，则打印0。按照输入中给出查询的顺序打印查询的答案.

【样例输入】

5

1 2

5 1

2 3

4 2

12

1 1

2 3

3 1

3 2

3 3

3 4

1 2

2 4

3 1

3 3

3 4

3 5

【样例输出】

0

0

0

1

0

1

0

1

链接

https://www.luogu.com.cn/problem/CF343D

题解

题意：给出一棵树，有如下两种操作和一种询问：

1.将某个点的子树全部染色。

2.将一个点到祖先的路径上的所有点的清空染色。

还要支持查询某个点是否是被染色了。

子树处理，路径处理，可以用树链剖分完成，但是有点大材小用。

有这样一个很有用的性质：

如果一个点子树内有个点被清空了，那么这个点一定被清空了。所以我们可以在处理dfs序之后用线段树维护某个点是否染色，第一种操作可以直接完成，而第二种操作就可以改成单点修改了。

但是这样还有问题。样例中的操作：如果两个点1和2有边，对1染色，对2清空，再对2染色，此时1应该是没有颜色的。

为了处理这种情况，每次染色一个子树之前，先看看子树内有没有被清空过的，如果有，就清空这个子树的根的父亲（单点修改）。

这样只用一个线段树就可以完成了。

代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<map>
#include<set>
using namespace std;
#define ll long long
#define ls (rt<<1)
#define rs (rt<<1|1)
#define pb(x) push_back(x)
#define pr(x) cerr<<#x<<"="<<x<<endl
#define pri(x,lo) {cerr<<#x<<"={";for (int ol=0;ol<=lo;ol++)cerr<<x[ol]<<",";cer
#define N 3001000
int tag[N],cnt,p,tes,n,i,full[N],g[N],size,l[N],r[N],ty,o,fa[N];
struct node {int to,next;}e[N];
void add(int o,int p) {e[++size].to=p,e[size].next=g[o];g[o]=size;}
void dfs(int x)
{
        l[x]=++cnt;
 for (int k=g[x];k;k=e[k].next) if (e[k].to!=fa[x]) fa[e[k].to]=x,dfs(e[k].to);
        r[x]=cnt;
}
void pushdown(int rt)
{
 //printf("pushdown %d value=%d\n",rt,tag[rt]);
 if (tag[rt]==0) return ;
        full[ls]=full[rs]=(tag[rt]==1);
        tag[ls]=tag[rs]=tag[rt];
        tag[rt]=0;
}
void modify(int rt,int l,int r,int x,int y,int d)
{
 //pr(rt);
```

```
 if (x<=l&&r<=y)
        {
                full[rt]=(d==1);
                tag[rt]=d;
  return;
        }
 pushdown(rt);
 int mid=(l+r)/2;
 if (x<=mid) modify(ls,l,mid,x,y,d);
 if (y>=mid+1) modify(rs,mid+1,r,x,y,d);
        full[rt]=full[ls]&full[rs];
}
int query(int rt,int l,int r,int x,int y)
{
 int ret=1;
 if (x<=l&&r<=y) return full[rt];
 pushdown(rt);
 int mid=(l+r)/2;
 if (x<=mid) ret&=query(ls,l,mid,x,y);
 if (y>=mid+1) ret&=query(rs,mid+1,r,x,y);
 return ret;
}
int main()
{
 scanf("%d",&n);
 for (i=1;i<n;i++) scanf("%d %d",&o,&p),add(o,p),add(p,o);
 dfs(1);
 //pri(l,n);
 //pri(r,n);
 scanf("%d",&tes);
 while (tes--)
        {
  //pr(tes);
  scanf("%d %d",&ty,&o);
  if (ty==1)
                {
   int tmp=query(1,1,n,l[o],r[o]);
   //pr(tmp);
   if(tmp==0&&o!=1) modify(1,1,n,l[fa[o]],l[fa[o]],2);
   modify(1,1,n,l[o],r[o],1);
                }
  else if (ty==2) modify(1,1,n,l[o],l[o],2);
  else printf("%d\n",query(1,1,n,l[o],r[o]));
  //for (i=1;i<=9;i++) printf("%d tag=%d full=%d\n",i,tag[i],full[i]);
        }
}
```