目录

10.3 模拟赛

1. Credit Card

【问题描述】

Recenlty Luba 有一张信用卡可用，一开始金额为 0，每天早上可以去充任意数量的钱。到了晚上，银行会对信用卡进行一次操作，操作有三种操作。 1.如果 a[i]>0，银行会给卡充入 a[i]元。 2.如果 a[i]<0 银行从卡中扣除 a[i]元。 3.如果 a[i]=0 银行会查询卡里的金额。 有两条规则，如果违背信用卡就会被冻结。 1.信用卡里的金额不能大于 d。 2.当银行查询卡里的金额时，金额不能为负。 Recenlty Luba 想知道最少去充多少次钱，可以使她在接下来的 n 天里信用卡不被冻结。

【输入格式】

第一行输入两个整数 n，d，n(1≤n≤10^5,1≤d≤10^9) 表示天数和限额；

接下来输入一行 n 个整数 $a1,a2,\cdots,an$(-10^4≤ai≤10^4)，表示第 i 天的操作。

【输出格式】

若无解则输出-1，否则输出最少充钱次数。

【样例输入】

```
5 10
-1 5 0 -5 3
```

【样例输出】

```
0
```

题目链接：https://www.luogu.com.cn/problem/CF893D

【题目分析】

根据题意，由于存在上界和下界，所以每次充钱后的资金实际上是一个区间，我们既不能让钱太多也不能让钱太少，而维护区间很明显是不可行的，但区间的上下界是可以维护的。我们可以考虑下界和上界分别表示的两种情况，一种是尽量少的充钱，用于判断数值会不会超，用 sum1 表示，另一种是尽可能多的充钱，用于判断能否满足金额不为负，同时记录最少充值次数，用 sum2 表示。利用上下界进行答案的维护。

在修改金额时，若此时 sum1 已经超过 d 了，则无论如何都不可能有解，若 sum2 超过 d，则修改 sum2 的金额为 d。

在查询金额时，由于 sum1 和 sum2 都在为负时才需要充钱，那么对 sum1 来说，要充的尽量少，即 sum1=0，而对于 sum2 来说，此时必须充钱，尽量多的充，sum2=d。

用 sum1 判断无解情况，用 sum2 记录充值次数即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
int a[100010];
int main()
{
    ios::sync_with_stdio(false);
    int n,d;
    cin >> n >> d;
    for(int i=1;i<=n;i++)
        cin >> a[i];
    int sum1,sum2,cnt;
    sum1=sum2=cnt=0;
    for(int i=1;i<=n;i++)
    {
        if(!a[i])
        {
            if(sum1 <0)
                sum1 = 0;//不存钱变负数,那么存到剩 0
            if(sum2 <0)
                sum2 = d,cnt++;//存满钱
        }
```

```
        else
        {
            sum1 += a[i];
            sum2 += a[i];//余额的更改
            if(sum1 >d)
            {
                cout << -1;
                return 0;
            }
            if(sum2 > d)
                sum2 = d;
        }
    }
    cout << cnt;
}
```

2. Covered Points Count

【问题描述】
给你 n 个区间，求被这些区间覆盖层数 k(k<=n)的点的个数
【输入格式】
第一行一个整数，n<=2*10^5
以下 n 行，每行有两个整数，即这个区间的左右端点 l,r(0<=l<=r<=10^18)
【输出格式】

n 个整数，即每个被覆盖层数对应的点的个数

【样例输入】

```
3
0 3
1 3
3 8
```

【样例输出】

```
6 2 1
```

题目链接：https://www.luogu.com.cn/problem/CF1000C

【题目分析】

观察题目数据范围，发现区间范围很大，区间数量很少，所以无法用差分数组直接进行维护，需要离散化后或利用结构体排序进行维护。端点个数最多只有 4*1e5 个，所以实际受影响的点仍然可以用差分维护。

离散化或结构体排序可以将受影响的点进行排序，此时可以应用差分数组进行答案统计，每次遇到区间端点，用当前端点减去上一个端点，获得这段区间内的答案，遇到左端点+1，右端点-1。利用此方法可以拓展到大区间范围小区间数量的维护逻辑。

【参考代码】

```
#include<bits/stdc++.h>
#define int long long//开 long long
using namespace std;
const int N = 4e5 + 10;
const int INF = 0x3f3f3f3f;
int n, len, ans[N];//len 是点的个数
```

```
struct node
{
    int num;
    bool flag;
```

```
}p[N];
bool cmp(node a, node b)
{
    return a.num < b.num;//从小到大
}
signed main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        int x, y;
        cin >> x >> y;
        p[++len].num = x;//左
        p[len].flag = 1;
        p[++len].num = y + 1;//右，差分
        p[len].flag = 0;
    }
    sort(p + 1, p + len + 1, cmp);
    int id = p[1].num, now = p[1].flag;
    for(int i = 2; i <= 2 * n; i++)
    {
        ans[now] += p[i].num - id;//算个数
        id = p[i].num;
        if(p[i].flag == 1)//判断是否右端点
        {
            now++;
        }
        else
        {
            now--;
        }
    }
    for(int i = 1; i <= n; i++)
    {
        cout << ans[i] << " ";
    }
    return 0;
}
```

3. Maze 2D

【问题描述】

有一个 2×n 的迷宫，有一些格子不能走。能走的格子用 . 表示，不能走的用 X 表示。
第一行的格子编号为 1~n，第二行的格子编号为 n+1~2n。
有 m 组询问，每组询问给出两个数 u,v，您需要回答：编号为 u 的格子和编号为 v 的格子之间的最短路。保证 u 号格子和 v 号格子是能走的。如果 u,v 两个格子间不可达，输出 -1。1≤n,m≤2×10^5，1≤u,v≤2n。

【样例输入】

```
4 7
.X..
```

```
...X
5 1
1 3
7 7
1 4
6 1
4 7
5 7
```

【样例输出】

```
1
4
0
5
2
2
2
```

题目链接：https://www.luogu.com.cn/problem/CF413E

【题目分析】

搜索思路可以得部分分，当作迷宫问题来看即可。

观察题目数据范围，只有两行的迷宫，但却有多列，此时将问题转化为维护多列中上下两行元素关系的问题，将问题转化为区间值的维护，考虑应用线段树。

考虑用线段树维护四个值，d1、d2、d3、d4，分别表示在某段区间内，从左上角到右上角、从左上角到右下角，从左下角到右上角、从左下角到右下角的最短路，这样在合并时只需要考虑两条最短路的拼接情况，每种情况都由左右儿子的两种拼接可能完成。

最终查询输出时，找到 u，v 对应的端点，分类讨论输出答案即可。

【参考代码】

```
#include<bits/stdc++.h>
#define int long long
#define ls (k*2)
#define rs (k*2+1)
using namespace std;
const int N = 2e5+5,inf = 0x3f3f3f3f;
struct node{
    int d1,d2,d3,d4;
    void print()
    {
        cout<<d1<<' '<<d2<<' '<<d3<<' '<<d4<<'\n';
    }
    friend node operator + (node x,node y)
    {
        node res;
        res.d1 = min(inf,min(x.d1+y.d1,x.d2+y.d3)+1);
        res.d2 = min(inf,min(x.d1+y.d2,x.d2+y.d4)+1);
        res.d3 = min(inf,min(x.d4+y.d3,x.d3+y.d1)+1);
        res.d4 = min(inf,min(x.d4+y.d4,x.d3+y.d2)+1);
        return res;
    }
}s[N*4];
int n,m;
char a[2][N];
```

```
void pushup(int k)
{
    s[k] = s[ls]+s[rs];
}
void build(int k,int l,int r)
{
    if(l==r)
    {
        s[k] = (node){inf,inf,inf,inf};
        if(a[0][l]=='.') s[k].d1 = 0;
        if(a[1][l]=='.') s[k].d4 = 0;
        if(a[0][l]=='.'&&a[1][l]=='.') s[k].d2 = s[k].d3 = 1;
        return;
    }
    int mid = (l+r)/2;
    build(ls,l,mid),build(rs,mid+1,r);
    pushup(k);
}
node ask(int k,int l,int r,int x,int y)
{
    if(l>=x&&r<=y) return s[k];
    int mid = (l+r)/2;
    if(mid<x) return ask(rs,mid+1,r,x,y);
    if(mid>=y) return ask(ls,l,mid,x,y);
    return ask(ls,l,mid,x,y)+ask(rs,mid+1,r,x,y);
}
signed main()
{
//  freopen("family.in","r",stdin);
//  freopen("family.out","w",stdout);
    ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
    cin>>n>>m>>(a[0]+1)>>(a[1]+1);
    build(1,1,n);
    while(m--)
    {
        int u,v;
        cin>>u>>v;
        int x = u,y = v;
        if(x>n) x-=n;
        if(y>n) y-=n;
        if(x>y) swap(x,y),swap(u,v);
        auto now = ask(1,1,n,x,y);
        int ans;
        if(u<=n&&v<=n) ans = now.d1;
        if(u<=n&&v>n) ans = now.d2;
        if(u>n&&v<=n) ans = now.d3;
        if(u>n&&v>n) ans = now.d4;
        if(ans==inf) ans = -1;
        cout<<ans<<'\n';
    }
    return 0;
```

```
}
```

4. Civilization

【问题描述】

给出一个由 n 个点 m 条边组成的森林，有 q 组询问

给出点 x，输出点 x 所在的树的直径

给出点 x,y，（如果 x,y 在同一棵树中则忽略此操作）选择任意两点 u,v，使得 u 跟 x 在同一棵树中且 v 跟 y 在同一棵树中。将 u,v 之间连一条边，使得连边后的到的新树的直径最小

【输入格式】

第一行三个整数 n,m,q，分别表示 点的个数，边的个数和询问个数

接下来 m 行，每行两个整数 x,y，表示有一条链接点 x,y 的边

接下来 q 行，每行表示一条操作

操作 1：1 x

操作 2：2 x y

【输出格式】

输出行数为操作 1 的个数

每行一个整数表示对应的操作一的答案

【样例输入】

```
6 0 6
2 1 2
2 3 4
2 5 6
2 3 2
2 5 3
1 1
```

【样例输出】

```
4
```

题目链接：https://www.luogu.com.cn/problem/CF455C

【题目分析】

首先，可以用树形 dp 或者 bfs 求出每棵树的直径，并用并查集维护每个结点的联通情况。维护数组 c，对于每棵树的根结点 x，c[x]表示该棵树的直径长度。

接下来考虑操作 2，怎样使新树的直径最小，对于两棵树来讲，肯定是把直径上中间位置的点相连，新生成的直径最小，所以，最终的答案与树的形态无关，仅与树的直径有关，而且，合并后新的直径只跟两棵树的直径长度有关，记 ci 为以 i 为根的树的直径，要合并以 x，y 为根的两棵树，则新树的最小直径为$\lceil cx/2 \rceil + \lceil cy/2 \rceil + 1$，cx，cy 的最大值，其中新的树至少有 cx 和 cy 的长度。然后每次生成新的直径继续用并查集维护即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int Maxn=300000+10,inf=0x3f3f3f3f;
int d[Maxn],g[Maxn];
int f[Maxn],c[Maxn];
bool vis[Maxn];
vector <int> e[Maxn];
int n,m,q,len;
inline int read()
{
    int s=0,w=1;
```

```
    char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')w=-1;ch=getchar();}
    while(ch>='0' && ch<='9')s=(s<<3)+(s<<1)+(ch^48),ch=getchar();
    return s*w;
}
int find(int x)
{
    if(f[x]==x)return x;
    return f[x]=find(f[x]);
}
void dfs(int x,int fa) // 树形 DP 求树的直径
{
    int m1=-1,m2=-1;
    for(int i=0;i<e[x].size();++i)
    {
        int y=e[x][i];
        if(y==fa)continue;
        dfs(y,x);
        int tmp=d[y]+1;
        d[x]=max(d[x],tmp);
        if(tmp>m1)m2=m1,m1=tmp;
        else if(tmp>m2)m2=tmp;
    }
    g[x]=max(max(0,m1+m2),max(m1,m2));
    len=max(len,g[x]);
}
void calc(int x) // 寻找树的直径
{
    len=0;
    dfs(x,0);
    c[x]=len;
}
int main()
{
    n=read(),m=read(),q=read();

    for(int i=1;i<=n;++i)
    f[i]=i;

    for(int i=1;i<=m;++i)
    {
        int x=read(),y=read();
        f[find(x)]=find(y);
        e[x].push_back(y);
        e[y].push_back(x);
    }

    for(int i=1;i<=n;++i)
    {
        if(f[i]!=i || vis[i])continue;
        calc(i);
```

```
        vis[i]=1;
    }

    while(q--)
    {
        int opt=read(),x=read();
        if(opt==1)
        {
            printf("%d\n",c[find(x)]);
            continue;
        }
        int y=read();
        x=find(x),y=find(y);
        if(x==y)continue;
        int tmp=((c[x]+1)>>1)+((c[y]+1)>>1)+1; //向上取整的方法

        tmp=max(tmp,max(c[x],c[y]));

        f[find(x)]=find(y);
        c[find(x)]=tmp;
    }

    return 0;
}
```