

A**B****C**

线段树

AC代码

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int maxn=1e6+50;
ll n,m,k;
ll lazy[maxn];
ll minn[maxn];
struct operation{
    ll t,x;
    int g;
    operation(ll a,ll b,int c){
        t=a;
        x=b;
        g=c;
    }
};
vector<operation>op[maxn];
void pushdown(int rt){
    if(lazy[rt]){
        lazy[rt<<1]+=lazy[rt];
        lazy[rt<<1|1]+=lazy[rt];
        minn[rt<<1]+=lazy[rt];
        minn[rt<<1|1]+=lazy[rt];
        lazy[rt]=0;
    }
}
void pushup(int rt){
    minn[rt]=min(minn[rt<<1],minn[rt<<1|1]);
}
void update(int l,int r,int L,int R,ll v,int rt){
    if(r<=R&&l>=L){
        minn[rt]+=v;
        lazy[rt]+=v;
        return;
    }
    pushdown(rt);
    int mid=(l+r)>>1;
    if(L<=mid)update(l,mid,L,R,v,rt<<1);
    if(R>mid)update(mid+1,r,L,R,v,rt<<1|1);
}
```

```
    pushup(rt);
}
ll query(int l,int r,int L,int R,int rt){
    if(r<=R&&l>=L)return minn[rt];
    pushdown(rt);
    int mid=(l+r)>>1;
    ll res=1e9;
    if(L<=mid)res=min(res,query(l,mid,L,R,rt<<1));
    else if(R>mid)res=min(res,query(mid+1,r,L,R,rt<<1|1));
    return res;
}
int main(){
    cin>>n>>m>>k;
    int o,a,b;
    ll c;
    for(int i=1;i<=m;i++){
        cin>>o;
        if(o==1){
            cin>>a>>b>>c;
            op[a].push_back(operation(i,c,0));
            op[b+1].push_back(operation(i,-c,0));
        }else{
            cin>>a>>b;
            op[a].push_back(operation(i,-k,1));
            op[b+1].push_back(operation(i,k,-1));
        }
    }
    ll ans=0,tmp=0,num=0;
    for(int i=1;i<=n;i++){
        int sz=op[i].size();
        for(int j=0;j<sz;j++){
            update(1,m,op[i][j].t,m,op[i][j].x,1);
            num+=op[i][j].g;
        }
        if(sz>0){
            tmp=num-ceil(-1.0*min(minn[1],0ll)/k);
        }
        ans+=tmp;
    }
    cout<<ans<<endl;
    return 0;
}
```

D

E

F

G

H

题目大意

给定两个长度为 n 的数组 $\mathbf{a}, \mathbf{b}$ ，记 $d = \sum(\text{abs}(a[i] - b[i]))$ 在至多交换数组 $\mathbf{b}$ 中的两个元素一次的情况下求 d 的最小值。

算法思路

设不交换时的答案为 d 若交换 $b[i]$ 和 $b[j]$ 则最终答案变为 $d - 2 * (\min(\max(a[i], b[i]), \max(a[j], b[j])) - \max(\min(a[i], b[i]), \min(a[j], b[j])))$

目标是找到 i, j 使上式被减数最大。

考虑尺取法，将 $\max(a[i], b[i])$ 记为 $\text{maxx}[i]$ 并且将 $a[i] > b[i]$ 和 $a[i] \leq b[i]$ 的 $a[i]$ 和 $b[i]$ 分别存在两个数组里，记为 $\mathbf{p}[0]$ 和 $\mathbf{p}[1]$

$\mathbf{p}[0], \mathbf{p}[1]$ 按照较大数从大到小排序 maxx 按照从大到小排列。

从大到小遍历 maxx 数组，找到当 $\mathbf{p}[0], \mathbf{p}[1]$ 的较大数 $\geq \text{maxx}[i]$ 时，较小数的最小值 $\text{min0}, \text{min1}$ 并且在之前遍历时所计算得到的最小值依然满足要求，可以直接使用，最后计算 $\text{maxx}[i] - \max(\text{min0}, \text{min1})$ 取最大值

遍历结束后会求出一个最大的 $\text{maxx}[i] - \max(\text{min0}, \text{min1})$ ，为所求，输出 $\text{ans} - 2 * (\text{maxx}[i] - \max(\text{min0}, \text{min1}))$;

写的思路略乱，看代码比较清晰

AC代码

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int maxn=1e6+5;
ll a[maxn],b[maxn],n;
ll ans;
pair<ll,ll>pa[2][maxn];
ll maxx[maxn];
bool cmp(pair<ll,ll>a, pair<ll,ll>b){
    return a.first<b.first;
}
int main(){
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>a[i];
    }
    int cnt[2]={0,0};
    for(int i=0;i<n;i++){
        cin>>b[i];
```

```
    ans+=abs(b[i]-a[i]);
    if(a[i]>b[i]){
        pa[0][cnt[0]++]=make_pair(a[i],b[i]);
    }else{
        pa[1][cnt[1]++]=make_pair(b[i],a[i]);
    }
    maxx[i]=max(a[i],b[i]);
}
sort(pa[0],pa[0]+cnt[0],cmp);
sort(pa[1],pa[1]+cnt[1],cmp);
sort(maxx,maxx+n);
int p[2]={cnt[0]-1,cnt[1]-1};
ll d=0;
ll mina=2e9;
ll minb=2e9;
for(int i=n-1;i>=0;i--){
    while(p[0]>=0&&pa[0][p[0]].first>=maxx[i]){
        mina=min(mina,pa[0][p[0]].second);
        p[0]--;
    }
    while(p[1]>=0&&pa[1][p[1]].first>=maxx[i]){
        minb=min(minb,pa[1][p[1]].second);
        p[1]--;
    }
    d=max(d,maxx[i]-max(mina,minb));
}
ans=ans-2*d;
cout<<ans<<endl;
return 0;
}
```

I

J

题目大意

赌博，初始赌1块钱，如果输了，下次赌两倍(1,2,4···)，如果赢了，获得当前次两倍的赌资(赢2,4,8···)，如果现在的钱不够赌，则失败。

现在有n块钱，想赢到n+m块，求成功的概率。

算法思路

钱数为x时失败的概率为 $1/(2^k)$, $k=\lceil \log_2(x+1) \rceil$

设从n赢到x块钱时失败的概率为ans,则从n赢到x+1块钱时失败的概率为 $ans+(1-ans)*1/(2^k)$, $k=\lceil \log_2(x+2) \rceil$

对于一段区间内的x其k是相同的，这是一个线性递推数列，很容易得到通项公

式 $a_n=(ans-1)*(1-1/(2^k))^n+1$, $a_0=ans$;

即 $ans=(ans-1)*(1-1/(2^k))^n+1$

遍历每一个 k 找到对应的项数 n 更新 ans 可求出最终失败的概率

输出 $1-ans$ 即为答案

AC代码

```
#include<bits/stdc++.h>
using namespace std;
long long n,m;
const int mod=998244353;
long long mul(long long a,long long b){
    long long res=1;
    while(b){
        if(b&1){
            res=res*a%mod;
        }
        a=a*a%mod;
        b>>=1;
    }
    return res;
}
int main(){
    cin>>n>>m;
    long long tmp=n;
    long long ans=0;
    long long p=pow(2,int(log2(n+1))+1)-1;
    p=min(p,tmp+m);
    int a0=log2(n+1);
    int a1=log2(n+m);
    for(int i=a0;i<=a1;i++){
        long long Ln=mul(pow(2,i),mod-2);
        long long C=(ans-1+mod)%mod;
        ans=(C*mul((1-Ln+mod)%mod,p-n)%mod+1)%mod;
        n=p;
        p=min(tmp+m,(p+1)*2-1);
    }
    cout<<(1-ans+mod)%mod<<endl;
    return 0;
}
```

K

题目大意

给定一个简单图和常数 k 每条边长度均为1, 可以在图上的任意一条边上加一个点, 操作次数不限。求最终图中与1号节点距离不大于 k 的点最多有多少个。

算法思路

bfs固定一棵生成树，同时处理每个点到1号点的距离。可以在所有非生成树的边加满点，所有生成树边不动（非生成树的边好像叫桥来着）

遍历每一条非生成树边，可以加 $2*k-dis[u]-dis[v]$ 数量的点 u,v 为该边两端的节点。注意处理 u 或 v 本身距离大于 k 的情况。

最后检查生成树的叶节点，若叶节点的距离小于 k 可以在叶子结点处加点 $k-dis[leaf]$

答案为上述所添加的节点数 + 最初距离小于等于 k 的节点数

AC代码

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int maxn=2e5+50;
ll n,m,k,ans;
int to[maxn<<1];
int nxt[maxn<<1];
int head[maxn],cnt=-1;
int a,b,d[maxn];
int dis[maxn];
bool vis[maxn];
bool f[maxn<<1];
bool flag[maxn];
void add_edge(int u,int v){
    to[++cnt]=v;
    nxt[cnt]=head[u];
    head[u]=cnt;
}
vector<int>ve[maxn];
void bfs(int st){
    queue<pair<int,int> >qu;
    qu.push(make_pair(st,0));
    vis[st]=1;
    while(!qu.empty()){
        int fr=qu.front().first;
        int d=qu.front().second;
        qu.pop();
        for(int i=head[fr];i!=-1;i=nxt[i]){
            if(vis[to[i]]){
                continue;
            }
            vis[to[i]]=1;
            qu.push(make_pair(to[i],d+1));
            ve[fr].push_back(to[i]);
            f[i]=1;
            dis[to[i]]=d+1;
        }
    }
}
```

```

    }
}
bool dfs(int p,int fa){
    int sz=ve[p].size();
    ans++;
    for(int i=0;i<sz;i++){
        if(dis[ve[p][i]]>k)continue;
        if(!dfs(ve[p][i],p)){
            ans+=k-dis[ve[p][i]];
        }
        flag[p]=1;
    }
    for(int i=head[p];i!=-1;i=nxt[i]){
        if(f[i]||to[i]==fa||dis[to[i]]>k)continue;
        ans+=2*k-dis[p]-dis[to[i]];
        f[i]=1;
        f[i^1]=1;
        flag[p]=1;
        flag[to[i]]=1;
    }
    return flag[p];
}
int main(){
    cin>>n>>m>>k;
    memset(head,-1,sizeof(head));
    for(int i=0;i<m;i++){
        cin>>a>>b;
        add_edge(a,b);
        add_edge(b,a);
    }
    bfs(1);
    dfs(1,0);
    cout<<ans<<endl;
    return 0;
}

```

L

题目大意

给定三个长度为 n 的排列 a, b, c 和 x, y, z 经过一次操作 x, y, z 变为 $a[y], b[z], c[x]$ （注意下标顺序），初始 $x=y=z=1$ 。Q次询问，每次询问 x', y', z' 求最少的操作次数，使 $(x, y, z) = (1, 1, 1)$ 变为 (x', y', z') 。如果不能则输出-1。

算法思路

只考虑 x ，每3次操作 x 会变成 $a[b[c[x]]]$ ，维护 $px[i]$ 为 i 经过3次操作变成了 $px[i]$ ，则 $px[i]$ 也是 n 的一个排列。 x 按照 $x=px[x]$ 的规则变换，该变换会在 px 中形成若干个环，我们预处理出每个 x 在 px 中参与的环长度 $cir[x]$ 和在环中的位置 $dis[x]$ ，同时记录每个 x 所属环的编号。那么最终 x 到达 x' 的操作次数为 $dis[x'] - dis[x] + k * cir[x]$ 。对 a, b, c 三个排列都按如上方式处理。

查询时，只需求是否存在一组 k_1, k_2, k_3 使得 $\text{dis}[x'] - \text{dis}[1] + k_1 * \text{cir}[x'] = \text{dis}[y'] - \text{dis}[1] + k_2 * \text{cir}[y'] = \text{dis}[z'] - \text{dis}[1] + k_3 * \text{cir}[z']$.

即求同余方程组：（用扩展中国剩余定理）

$$T = (\text{dis}[x'] - \text{dis}[1]) \% \text{cir}[x']$$

$$T = (\text{dis}[y'] - \text{dis}[1]) \% \text{cir}[y']$$

$$T = (\text{dis}[z'] - \text{dis}[1]) \% \text{cir}[z']$$

由于该预处理方法是每三个一跳，遍历三组初始

值 $x=1, y=1, z=1, x=a[1], y=b[1], z=c[1], x=a[b[1]], y=b[c[1]], z=c[a[1]]$ 每组求出一个 T 最终每组的答案分别为 $3 * T + i (i=0, 1, 2)$ 取最小值。

无解情况 x' 与 x 不在同一个环上，或方程组无解。

AC代码

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int maxn=1e5+50;
ll n,q;
ll a[3][maxn];
bool f[3][maxn];
ll p[3][maxn];
ll dis[3][maxn];
ll cir[maxn];
int ma[3][maxn];
int cnt;
int getstart(int u,int v,int id){
    if(ma[id][u]!=ma[id][v])return -1;
    int tmp=dis[id][v]-dis[id][u];
    return tmp<0?tmp+cir[ma[id][u]]:tmp;
}
ll mul(ll a,ll b,ll mod){
    ll res=0;
    while(b){
        if(b&1){
            res=(res+a)%mod;
        }
        a=(a+a)%mod;
        b>>=1;
    }
    return res;
}
ll ex_gcd(ll A,ll B,ll &x,ll &y){
    if(B==0){
        x=1;
        y=0;
        return A;
    }
```



```

    }
    ll d=ex_gcd(B,A%B,y,x);
    y-=A/B*x;
    return d;
}
int main(){
    cin>>n;
    memset(dis,-1,sizeof(dis));
    for(int i=0;i<3;i++){
        for(int j=1;j<=n;j++){
            cin>>a[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        p[0][i]=a[0][a[1][a[2][i]]];
        p[1][i]=a[1][a[2][a[0][i]]];
        p[2][i]=a[2][a[0][a[1][i]]];
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<3;j++){
            if(ma[j][i])continue;
            ma[j][i]=++cnt;
            dis[j][i]=0;
            int k=i,len=0;
            do{
                ma[j][p[j][k]]=cnt;
                dis[j][p[j][k]]=dis[j][k]+1;
                k=p[j][k];
                len++;
            }while(i!=k);
            cir[cnt]=len;
        }
    }
    cin>>q;
    ll x,y,z;
    while(q--){
        cin>>x>>y>>z;
        ll ans=-1;
        for(int i=0;i<3;i++){
            int x1=1,y1=1,z1=1;
            if(i==1){
                x1=a[0][1];
                y1=a[1][1];
                z1=a[2][1];
            }
            if(i==2){
                x1=a[0][a[1][1]];
                y1=a[1][a[2][1]];
                z1=a[2][a[0][1]];
            }
            int st[3];

```

}

知识点总结

<https://wiki.cvbbacm.com/>

解同余方程组

$$x \equiv r[i] \pmod{m[i]}, i=0, 1, \dots, n-1$$

注意模数 m 可能不两两互质

遍历 $i=1, 2, 3, \dots, n-1$, 每次解方程 $x^* + t^*M \equiv r[i] \pmod{m[i]}$, 即 $t^*M + k^*m[i] = r[i] - x^*$, 其中 x^* 为之前 $i-1$ 个方程的一个解, M 为前 $i-1$ 个 m 的最小公倍数

最后解为 $x^* + k^* \text{lcm}(m[i])$

模板 (缩略版)

```
ll m[n], r[n];
ll ex_CRT(){
    ll M=m[0];
    ll tmp=r[0];
    for(int i=1; i<n; i++){
        ll X,Y;
        ll d=ex_gcd(M, m[i], X, Y);
        if((tmp-r[i])%d!=0){
            tmp=-1;
            //no solution
            break;
        }
        //mul() is a fast multiply which prevents overflow
        //just like fast power
        X=mul(X, ((r[i]-tmp%m[i]+m[i])%m[i])/d, m[i]/d);
        //X = X*(r[i]-tmp)%(c[i]/d)
        tmp=tmp+X*M;
        M=M*c[i]/d; //update M=lca(m[i])
        tmp=(tmp%M+M)%M; //make the solution positive
    }
    return tmp;
}
```

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2023-2024:teams:ikun_is_coding:2023_%E7%89%9B%E5%AE%A2%E6%9A%91%E6%9C%9F%E5%A4%9A%E6%A0%A1%E8%AE%AD%E7%BB%83%E8%90%A5_1&rev=1689861267

Last update: 2023/07/20 21:54