目录

一、 字符串

1. KMP

```
#include <bits/stdc++.h>
using namespace std;
void solve()
{
    int n,ns;
    string s,p;
    cin>>s>>p;


    n=p.size(),ns=s.size();
    vector<int>next(n);//求 p 的 next 数组
    next[0]=0;
    int pre=0,i=1;
    while(i<n)
    {
        if(p[pre]==p[i]) next[i++]=++pre;
        else
        {
            if(pre==0) next[i++]=0;
            else pre=next[pre-1];
        }
    }

    vector<int>ans;
    int j;
    i=j=0;
    while(i<ns)
    {
        if(s[i]==p[j]) i++,j++;
        else
        {
```

```
            if(j) j=next[j-1];
            else i++;
        }
        if(j==n)
        {
            ans.push_back(i-j+1);
            j=next[j-1];
        }
    }
    for(auto i:ans) cout<<i<<"\n";
    for(auto i:next) cout<<i<<" ";
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
    //cin>>T;
    while(T--) solve();
}
```

2. Manacher

```
#include <bits/stdc++.h>
using namespace std;
string s,t;
void init()
{
    t+="$";
    t+="#";
    for(auto i:s)
    {
        t+=i;
        t+="#";
    }
    t+="^";
}
int p[22000010];
void manacher()
{
    init();
    int n=t.size();
    int mr=0,mid;
```

```
    for(int i=1;i<n;i++)
    {
        if(i<mr) p[i]=min(p[mid*2-i],mr-i);
        else p[i]=1;
        while(t[i-p[i]]==t[i+p[i]]) p[i]++;
        if(i+p[i]>mr) mr=i+p[i],mid=i;
    }
}
void solve()
{
    int res=0;
    cin>>s;
    manacher();
    for(int i=0;i<t.size();i++) res=max(res,p[i]);
    cout<<res-1<<"\n";
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
//  cin>>T;
    while(T--) solve();
}
```

3. Tire 树

```
#include <bits/stdc++.h>
using namespace std;
int n,u,v,w,cnt,ans,s[100010];
vector<pair<int,int>>g[100010];
int nex[30*100010][2];
void insert(int x)//插入 trie 树
{
    int now=0;
    for(int i=30;i>=0;i--)
    {
        int t=x>>i&1;
        if(!nex[now][t]) nex[now][t]=++cnt;
        now=nex[now][t];
    }
}
```

```
void query(int x)//查找与 x 异或最大的
{
	int res=0,now=0;
	for(int i=30;i>=0;i--)
	{
		int t=x>>i&1;
		if(nex[now][!t])//有反向走反向
		{
			res|=1<<i;
			now=nex[now][!t];
		}
		else now=nex[now][t];
	}
	ans=max(ans,res);//找最大
}
void dfs(int u,int fa)
{
	for(auto t:g[u])
	{
		int v=t.first;
		int w=t.second;
		if(v==fa) continue;
		s[v]=s[u]^w;//s[v]为根节点 1 号到 v 的路径异或和
		dfs(v,u);
	}
}
void solve()
{
	cin>>n;
	for(int i=1;i<n;i++)
	{
		cin>>u>>v>>w;
		g[u].push_back({v,w});
		g[v].push_back({u,w});
	}
	dfs(1,0);
	for(int i=1;i<=n;i++) insert(s[i]);
	for(int i=1;i<=n;i++) query(s[i]);
	cout<<ans;
}

signed main()
{
	ios::sync_with_stdio(0);
```

```
    cin.tie(0);cout.tie(0);
    int T=1;
//  cin>>T;
    while(T--) solve();
}
```

4. 字符串哈希

随机数哈希!!!!!!!
返回 unsigned int long long　异或哈希/图论判通路
std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define P 1331
#define mod 998244353
int p[400010],h[400010];
unsigned long long get(int l,int r)
{
    return (h[r]-h[l-1]*p[r-l+1]%mod+mod)%mod;
}
bool ck(int l,int r,int l1,int r1)
{
    return get(l,r)==get(l1,r1);
}
void solve()
{
    string s;
    cin>>s;
    int n=s.size();
    s=" "+s;
    p[0]=h[0]=1;
    for(int i=1;i<=n;i++)
    {
        p[i]=p[i-1]*P%mod;
        h[i]=(h[i-1]*P%mod+s[i]-'a')%mod;
    }
    for(int i=(n+1)/2;i>=2;i--)
    {
        if(ck(i,n,1,n-i+1))
        {
```

```
                cout<<"YES\n";
                cout<<s.substr(1,n-i+1)<<"\n";
                return ;
            }
        }
        cout<<"NO\n";
}
signed main()
{
        ios::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        int T=1;
        //cin>>T;
        while(T--) solve();
}
/*
int a=0,b=0,k=1;
for(int i=1;i<=m;i++)
{
        a=(a*p%P+(t[m+1-i]-'a'+1))%P;//正哈希
        b=(b+k*(t[m+1-i]-'a'+1)%P)%P;//反哈希
        k=k*p%P;
        //m+1-i----m
        if(a==b&&l[m+1-i-1]<r[m]) ans++;//判断是否相等、求回文
}
*/
```

5. Z 函数

```
#include <bits/stdc++.h>
using namespace std;
#define N 200010
string s;
int n,l,r;
vector<int>z;
bool ck(int x)
{
        int cnt=0;
        for(int i=1;i<=n;i++) if(z[i]>=x) i+=x-1,cnt++;
        return cnt>=l;
}
vector<int> zfun(string s)//z 算法，z[i]表示 s[i...n]与 s[1..n]的 lcp
```

```
{
    vector<int>z(s.size(),0);
    z[1]=n;
    for(int i=2,l=1,r=1;i<=n;i++)
    {
        z[i]=i<=r? min(z[i-l+1],r-i+1):0;
        while(i+z[i]<=n&&s[i+z[i]]==s[1+z[i]]) z[i]++;
        if(i+z[i]-1>r) r=i+z[i]-1,l=i;
    }
    return z;
}
vector<int>g[N];
int ans[N];
void solve()
{
    cin>>n>>l>>r>>s;
    for(int i=1;i<=n;i++) g[i].clear(),ans[i]=0;
    s="$"+s;
    z=zfun(s);
    for(int i=1;i<=n;i++) g[z[i]].push_back(i);
    set<int>st;
    for(int i=n;i>=1;i--)//枚举 lcp
    {
        for(auto t:g[i]) st.insert(t);//能跳的下表
        int cnt=0,j=1;
        while(j<=n)
        {
            auto it=st.lower_bound(j);
            if(it==st.end()) break;
            cnt++;//段数
            j=*it+i;
        }
        ans[cnt]=max(ans[cnt],i);//更新
    }
    ans[n+1]=0;
    for(int i=n;i>=1;i--) ans[i]=max(ans[i],ans[i+1]);//i+1 段的 lcp，i 段一定也满足
    for(int i=l;i<=r;i++) cout<<ans[i]<<" ";
    cout<<"\n";
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    int T=1;
```

```
    cin>>T;
    while(T--) solve();
}
```

二、 数学

1. 最大公约数&最小公倍数

```
int gcd(int x,int y)
{
    return y==0? x:gcd(y,x%y);
}
int lcm(int x,int y)
{
    return x*y/gcd(x,y);
}
```

2. SG 函数

```
for(int i=1;i<=n;i++)
{
    cin>>a[i];
    ans^=a[i];
}

if(!ans) cout<<"lose\n";//sg 函数为 0，必败态
else
{
    for(int i=1;i<=n;i++)
    if(a[i]>(a[i]^ans))//找到一个必败态，转移给对手
    {
        cout<<a[i]-(a[i]^ans)<<" "<<i<<"\n";
        a[i]^=ans;
        break;
    }
    for(int i=1;i<=n;i++) cout<<a[i]<<" ";
    cout<<"\n";
}
```

3. 乘法逆元

```
void init(int n)//线性求逆元  inv(x)表示 1/x 对 p 取模
{
	inv[1]=1;
	for(int i=2;i<=n;i++) inv[i]=1ll*(p-p/i)*inv[p%i]%p;
}


int exgcd(int a,int b,int &x,int &y)
{
	if(b==0)
	{
		x=1;
		y=0;
		return a;
	}
	int res=exgcd(b,a%b,y,x);
	y-=a/b*x;
	return res;
}
int getinv(int a)//exgcd 求逆元  getinv(x)返回 1/x 对 P 取模
{
	int x=0,y=0;
	int d=exgcd(a,P,x,y);
	return d==1? (x%P+P)%P:-1;
}


int ksm(int a,int b)
{
	int res=1;
	while(b)
	{
		if(b&1) res=res*a%P;
		a=a*a%P;
		b>>=1;
	}
	return res;
}
int inv(int x)//适用 P 为质数的时候
{
	return ksm(x,P-2);
```

```
}
```

4. 矩阵快速幂

```
#define P 1000000007
typedef vector<vector<int>> M;//矩阵
M mul(M A,M B)//矩阵乘法
{
	int n=A.size();
	M C(n,vector<int>(n,0));//单位矩阵
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			for(int k=0;k<n;k++)
			{
				C[i][j]=(C[i][j]+A[i][k]*B[k][j]%P+P)%P;
			}
		}
	}
	return C;
}
M ksm(M A,int b)//矩阵快速幂
{
	int n=A.size();
	M res(n,vector<int>(n,0));//单位矩阵
	for(int i=0;i<n;i++) res[i][i]=1;//单位矩阵初始化
	while(b)
	{
		if(b&1) res=mul(res,A);
		A=mul(A,A);
		b>>=1;
	}
	return res;
}
```

5. 卢卡斯定理

```
#define int long long
int n,m,P,jc[100010];
int ksm(int a,int b)
{
	int res=1;
```

```
    while(b)
    {
        if(b&1) res=res*a%P;
        a=a*a%P;
        b>>=1;
    }
    return res;
}
int inv(int x)
{
    return ksm(x,P-2);
}
int C(int n,int m)
{
    if(m>n) return 0;
    return jc[n]*inv(jc[m])%P*inv(jc[n-m])%P;
}
int lucas(int n,int m)//卢卡斯定理  C(n,m)%P
{
    if(!m) return 1;
    return C(n%P,m%P)*lucas(n/P,m/P)%P;
}
void init()
{
    jc[0]=1;
    for(int i=1;i<=P;i++) jc[i]=jc[i-1]*i%P;
}
```

6. 欧拉函数

```
//欧拉函数是小于 x 的整数中与 x 互质的数的个数
//p[x]表示 x 的欧拉函数

//O(nlogn)
for(int i=1;i<=n;i++) p[i]=i;
for(int i=2;i<=n;i++)
{
    if(p[i]==i)//i 是质数
    for(int j=i;j<=n;j+=i) p[j]=p[j]*(i-1)/i;//筛 i 的倍数
}
```

```
//O(n)
vector<int>pr;
void euler(int n)
{
	p[1]=1;
	for(int i=2;i<=n;i++)
	{
		if(!f[i])
		{
			pr.push_back(i);
			p[i]=i-1;
		}
		for(int j=0;j<pr.size()&&pr[j]*i<=n;j++)
		{
			f[pr[j]*i]=1;//筛质数
			if(i%pr[j]==0)
			{
				p[i*pr[j]]=p[i]*pr[j];
				break;
			}
			else p[i*pr[j]]=p[i]*pr[j];
		}
	}
 }
```

7. 裴蜀定理

```
//裴蜀定理：ax+by=m，其有解的充要条件为 gcd（a，b）|m
//逆定理：a,b 是不全为 0 的整数，若 d>0 为 a，b 的公因数，且存在整数 x，y 使得 ax+by=d，
则 d=gcd（a，b）
void solve()
{
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		cin>>t;
		if(t<0) t=-t;
		ans=__gcd(ans,t);
	}
	cout<<ans<<"\n";
}
```

8. 斯特林数

```
//组合数     c(n,m)=c(n-1,m-1)+c(n-1,m)
//第一类斯特林数（圆排列）:   s(n,m)=s(n-1,m-1)+(n-1)*s(n-1,m)
//第二类斯特拉数（不排列）:   s(n,m)=求和(k:0-m) (-1^k)*C(m,k)*(m-k)^n*(1/m!)
S[0][0]=1;
for(int i=1;i<=n;i++)
for(int j=1;j<=i;j++)
S[i][j]=(S[i-1][j-1]+S[i-1][j]*(i-1)%P)%P;
```

9. 线性基

```
int p[70],p1[70],cnt;
void insert(int x)//插入一个值
{
    for(int i=63;i>=0;i--)
    {
        if(x>>i&1)
        {
            if(!p[i])
            {
                p[i]=x;
                return ;
            }
            else x^=p[i];
        }
    }
}
bool check(int x)//判断 x 是否可以异或得到
{
    for(int i=63;i>=0;i--)
    {
        if(x>>i&1)
        {
            if(!p[i]) return false;
            else x^=p[i];
        }
    }
    return true;
}
int mx()//返回异或最大值
{
    int res=0;
```

```
    for(int i=63;i>=0;i--) res=max(res,res^p[i]);
    return res;
}
int mi()//返回异或最小值
{
    for(int i=0;i<=63;i++) if(p[i]) return p[i];
}

void exchange()//改造线性基
{
    for(int i=0;i<=63;i++)
    {
        for(int j=i-1;j>=0;j--)
        if(p[i]>>j&1) p[i]^=p[j];//其他位变成 0，保证唯一性

        if(p[i]) p1[cnt++]=p[i];//记录新的线性基
    }

}
int min_k_num(int k)//求第 k 小的异或值，先调用 exchange 函数
{
    int ans=0;
    for(int i=0;i<cnt;i++)
    {
        if(k>>i&1) ans^=p1[i];
    }
    return ans;
}
```

10.线性筛

```
bool f[100000001];
vector<int>p;
void init()//线性筛，时间复杂度 O(n)
{
    for(int i=2;i<=1e8;i++)
    {
        if(!f[i]) p.push_back(i);
        for(int j=0;j<p.size()&&p[j]*i<=1e8;j++)
        {
            f[p[j]*i]=1;
            if(i%p[j]==0) break;
        }
```

```
    }
}
```

11.组合数

```
#define P 1000000007
int jc[100010];
void init()
{
    jc[0]=1;
    for(int i=1;i<100000;i++) jc[i]=jc[i-1]*i%P;
}
int ksm(int a,int b)
{
    int res=1;
    while(b)
    {
        if(b&1) res=res*a%P;
        a=a*a%P;
        b>>=1;
    }
    return res;
}
int inv(int x)
{
    return ksm(x,P-2);
}
int C(int a,int b)//适合 a，b 均大于 1e4
{
    return jc[a]*inv(jc[b]*jc[a-b]%P)%P;
}



int C[2010][2010];
void init()//适合 a*b<=1e7
{
    for(int i=0;i<=2000;i++)
    {
        for(int j=0;j<=i;j++)
        {
            if(!j) C[i][j]=1;
            else C[i][j]=(C[i-1][j-1]+C[i-1][j])%P;
```

```
        }
    }
}
```

12. 三分法

求极小值

```
        int L=1,R=n;
        while(L<R)
        {
            int m1=L+(R-L)/3;
            int m2=R-(R-L)/3;
            if(f(m1,m2)) R=m2-1;//f(m1)<f(m2) 极大值时刚好相反
            else L=m1+1;
        }
        cout<<L<<"\n";//f（L）即为函数极小值
```

三、 图论

1. 2-SAT

```
#include <bits/stdc++.h>
using namespace std;
#define N 2000010
int n,m,a,fa,b,fb;
vector<int>g[N];
int dfn[N],low[N],tot,stk[N],instk[N],top,scc[N],siz[N],cnt;
void tarjan(int x)
{
    //入 x 时，盖戳，入栈
    dfn[x]=low[x]=++tot;
    stk[++top]=x,instk[x]=1;
    for(auto y:g[x])
    {
        if(!dfn[y])//若 y 尚未访问
        {
            tarjan(y);
            low[x]=min(low[x],low[y]);//回 x 时更新 low
        }
        else if(instk[y]) //已访问且在栈中
        low[x]=min(low[x],dfn[y]);//更新 low
```

```
    }
    //离 x 时，记录 scc
    if(dfn[x]==low[x])//若 x 是 scc 的根
    {
        int y;
        cnt++;
        //cout<<cnt<<":";
        do
        {
            y=stk[top--];
        //    cout<<y<<" ";
            instk[y]=0;//出栈
            scc[y]=cnt;//SCC 编号
            siz[cnt]++;//SCC 大小
        }while(y!=x);
        //cout<<"\n";
    }
}

void solve()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>a>>fa>>b>>fb;
        //1 1 -a-->b -b-->a
        //0 1 a-->b -b-->-a
        //1 0 b-->a -a-->-b
        //0 0 a-->-b b-->-a
        g[a+n*(fa&1)].push_back(b+n*(fb^1));
        g[b+n*(fb&1)].push_back(a+n*(fa^1));
    }
    for(int i=1;i<=2*n;i++) if(!dfn[i]) tarjan(i);

    for(int i=1;i<=n;i++)
    {
        if(scc[i]==scc[i+n])
        {
            cout<<"IMPOSSIBLE\n";
            return ;
        }
    }

    cout<<"POSSIBLE\n";
```

```
    for(int i=1;i<=n;i++) cout<<(scc[i]<scc[i+n])<<" ";
    cout<<"\n";
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
    //cin>>T;
    while(T--) solve();
}
```

2. 最小生成树

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define N 5010
int n,m,x,y,z,vt[N],ans,fa[N];
vector<pair<int,int>>g[N];
vector<pair<int,pair<int,int>>>e;
void prim()
{
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>q;
    q.push({0,1});
    while(!q.empty())
    {
        int w=q.top().first;
        int u=q.top().second;
        q.pop();
        if(vt[u]) continue;
        vt[u]=1;
        ans+=w;
        for(auto v:g[u]) q.push(v);
    }
    for(int i=1;i<=n;i++)
    if(!vt[i])
    {
        cout<<"orz\n";
        return ;
    }
    cout<<ans<<"\n";
}
```

```
int find(int x)
{
	if(fa[x]==x) return x;
	return fa[x]=find(fa[x]);
}

void kruskal()
{
	for(int i=1;i<=n;i++) fa[i]=i;
	sort(e.begin(),e.end());
	for(auto i:e)
	{
		int w=i.first;
		int u=i.second.first;
		int v=i.second.second;
		if(find(u)==find(v)) continue;
		ans+=w;
		fa[find(u)]=find(v);
	}
	set<int>st;
	for(int i=1;i<=n;i++) st.insert(find(i));
	if(st.size()==1) cout<<ans<<"\n";
	else cout<<"orz\n";
}
void solve()
{
	cin>>n>>m;
	for(int i=1;i<=m;i++)
	{
		cin>>x>>y>>z;
		e.push_back({z,{x,y}});
		g[x].push_back({z,y});
		g[y].push_back({z,x});
	}
	kruskal();
	//prim();
}

signed main()
{
	ios::sync_with_stdio(0);
	cin.tie(0);cout.tie(0);
	int T=1;
```

```
    //cin>>T;
    while(T--) solve();
}
```

3. Dijkstra

```
priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>q;
for(int i=1;i<=n;i++) dis[i]=(1ll<<31)-1;
dis[s]=0;
q.push({0,s});
while(!q.empty())
{
    int u=q.top().second;
    q.pop();
    if(vt[u]) continue;
    vt[u]=1;
    for(auto v1:g[u])
    {
        int v=v1.first;
        if(dis[v]>dis[u]+v1.second)
        {
            dis[v]=dis[u]+v1.second;
            q.push({dis[v],v});
        }
    }
}
```

4. Floyd

```
memset(dis,0x3f3f3f3f,sizeof(dis));
for(int k=1;k<=n;k++)//中间点最前面（floyd)
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++) dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
```

5. spfa//判负环

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define N 3010
#define inf 0x3f3f3f3f
int n,m,s,u,v,w;
```

```
int d[N],vt[N],cnt[N];
vector<pair<int,int>>g[N];
bool bellmanford(int s)//判负环
{
	d[s]=0;
	vt[s]=1;
	queue<int>q;
	q.push(s);
	while(q.size())
	{
		int u=q.front();
		q.pop();
		vt[u]=0;
		for(auto v1:g[u])
		{
			int v=v1.first;
			int w=v1.second;
			if(d[v]>d[u]+w)
			{
				d[v]=d[u]+w;
				cnt[v]=cnt[u]+1;
				if(cnt[v]>=n) return true;
				if(!vt[v]) q.push(v),vt[v]=1;
			}
		}
	}
	return false;
}
void solve()
{
	cin>>n>>m;
	for(int i=1;i<=n;i++) d[i]=inf,cnt[i]=vt[i]=0,g[i].clear();
	for(int i=1;i<=m;i++)
	{
		cin>>u>>v>>w;
		if(w>=0)
		{
			g[u].push_back({v,w});
			g[v].push_back({u,w});
		}
		else g[u].push_back({v,w});
	}
	cout<<(bellmanford(1)==0? "NO":"YES")<<"\n";
}
```

```
signed main()
{
	ios::sync_with_stdio(0);
	cin.tie(0);cout.tie(0);
	int T=1;
	cin>>T;
	while(T--) solve();
}
```

6. 差分约束

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define N 5010
#define inf 1000000000
int n,m,x,y,z,ans;
int d[N],vt[N],cnt[N],h[N];
vector<pair<int,int>>g[N];
bool spfa(int s)
{
	queue<int>q;
	for(int i=1;i<=n;i++) d[i]=inf;
	d[s]=0;
	vt[s]=1;
	q.push(s);
	while(q.size())
	{
		int u=q.front();
		q.pop();
		vt[u]=0;
		for(auto v1:g[u])
		{
			int v=v1.first;
			int w=v1.second;
			if(d[v]>d[u]+w)
			{
				d[v]=d[u]+w;
				cnt[v]=cnt[u]+1;
				if(cnt[v]>=n+1) return true;
				if(!vt[v]) q.push(v),vt[v]=1;
			}
```

```
            }
        }
        return false;
}
void solve()
{
        cin>>n>>m;
        for(int i=1;i<=m;i++)
        {
                cin>>x>>y>>z;
                g[y].push_back({x,z});
        }
        for(int i=1;i<=n;i++) g[n+1].push_back({i,0});
        if(!spfa(n+1))
        {
                for(int i=1;i<=n;i++) cout<<d[i]<<" ";
                cout<<"\n";
        }
        else cout<<"NO\n";

}

signed main()
{
        ios::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        int T=1;
//      cin>>T;
        while(T--) solve();
}
```

7. 二分图

```
#include    <bits/stdc++.h>
using namespace std;
#define N 1010
int vt[N],mch[N];
vector<int>g[N];
bool dfs(int u,int t)
{
        if(vt[u]==t) return 0;
        vt[u]=t;
        for(auto v:g[u])
```

```
        if(mch[v]==0||dfs(mch[v],t))
        {
            mch[v]=u;
            return 1;
        }
        return 0;
}
void solve()
{
        int n,m,e,u,v;
        cin>>n>>m>>e;
        for(int i=1;i<=e;i++)
        {
            cin>>u>>v;
            g[u].push_back(v);
        }
        int ans=0;
        for(int i=1;i<=n;i++) if(dfs(i,i)) ans++;
        cout<<ans<<"\n";
}
int main()
{
        ios::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        int T=1;
        //cin>>T;
        while(T--) solve();
}
```

8. LCA

```
#include <bits/stdc++.h>
using namespace std;
#define N 500010
int n,m,s,u,v;
vector<int>g[N];
int dep[N],fa[N][22],lg[N];
void dfs(int u,int f)
{
        fa[u][0]=f;//fa[u][i]表示 u 的 2^i 祖先
        dep[u]=dep[f]+1;//深度
        for(int i=1;i<=lg[dep[u]];i++) fa[u][i]=fa[fa[u][i-1]][i-1];//u 的 2^i 祖先等于 2^(i-1)祖先
的 2^(i-1)祖先
```

```
    for(auto i:g[u])
    {
        if(i==f) continue;
        dfs(i,u);
    }
}

int lca(int u,int v)
{
    if(dep[u]<dep[v]) swap(u,v);
    while(dep[u]>dep[v]) u=fa[u][lg[dep[u]-dep[v]]-1];//跳到同一层
    if(u==v) return u;
    for(int k=lg[dep[u]]-1;k>=0;k--) if(fa[u][k]!=fa[v][k]) u=fa[u][k],v=fa[v][k];//跳到 lca 的下
一层
    return fa[u][0];
}
void solve()
{
    cin>>n>>m>>s;
    for(int i=1;i<n;i++)
    {
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;i++) lg[i]=lg[i-1]+(1<<lg[i-1]==i);
    dfs(s,0);
    while(m--)
    {
        cin>>u>>v;
        cout<<lca(u,v)<<"\n";
    }
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
//  cin>>T;
    while(T--) solve();
}
```

9. 树的直径

```
#include <bits/stdc++.h>
using namespace std;
#define N 300010
int n,m,q,op,x,y,fx,fy,fa[N],d[N],ans;
vector<int>g[N];
int find(int x)
{
	if(fa[x]==x) return x;
	return fa[x]=find(fa[x]);
}
int dfs(int u,int fa)//求 u 所在连通块的直径 ans
{
	int d1=0,d2=0;
	for(auto v:g[u])
	{
		if(v==fa) continue;
		int t=dfs(v,u)+1;
		if(t>=d1) d2=d1,d1=t;
		else if(t>=d2) d2=t;
	}
	ans=max(ans,d1+d2);
	return d1;
}
void solve()
{
	cin>>n>>m>>q;
	for(int i=1;i<=n;i++) fa[i]=i;
	for(int i=1;i<=m;i++)
	{
		cin>>x>>y;
		fa[find(x)]=find(y);
		g[x].push_back(y);
		g[y].push_back(x);
	}

	for(int i=1;i<=n;i++)//预处理连通块直径
	{
		if(fa[i]!=i) continue;
		ans=0;
		dfs(i,0);
		d[i]=ans;
```

```
    }

    while(q--)
    {
        cin>>op;
        if(op==1)
        {
            cin>>x;
            fx=find(x);
            cout<<d[fx]<<"\n";
        }
        else
        {
            cin>>x>>y;
            fx=find(x);
            fy=find(y);
            if(fx==fy) continue;//跳过
            int t=(d[fx]+1)/2+(d[fy]+1)/2+1;
            t=max(t,max(d[fx],d[fy]));//计算新的直径
            fa[fx]=fy;//联通
            d[fy]=t;
        }
    }
}

signed main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
    //cin>>T;
    while(T--) solve();
}
```

10.缩点

```
#include   <bits/stdc++.h>
using namespace std;
#define N 10010
vector<int>g[N];
int dfn[N],low[N],tot,stk[N],instk[N],top,scc[N],siz[N],cnt;
void tarjan(int x)
{
```

```
    //入 x 时，盖戳，入栈
    dfn[x]=low[x]=++tot;
    stk[++top]=x,instk[x]=1;
    for(auto y:g[x])
    {
        if(!dfn[y])//若 y 尚未访问
        {
            tarjan(y);
            low[x]=min(low[x],low[y]);//回 x 时更新 low
        }
        else if(instk[y]) //已访问且在栈中
        low[x]=min(low[x],dfn[y]);//更新 low
    }
    //离 x 时，记录 scc
    if(dfn[x]==low[x])//若 x 是 scc 的根
    {
        int y;
        cnt++;
        //cout<<cnt<<":";
        do
        {
            y=stk[top--];
        //    cout<<y<<" ";
            instk[y]=0;//出栈
            scc[y]=cnt;//SCC 编号
            siz[cnt]++;//SCC 大小
        }while(y!=x);
        //cout<<"\n";
    }
}
int main()
{
    int n,m,ans;
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
    }
    for(int i=1;i<=n;i++)
    if(!dfn[i]) tarjan(i);
```

```
    ans=0;
    for(int i=1;i<=cnt;i++)
    if(siz[i]>1) ans++;
    cout<<ans<<"\n";
}
```

11.割点

```
#include   <bits/stdc++.h>
using namespace std;
#define N 20010
vector<int>g[N];
int dfn[N],low[N],tot,cut[N],root;
void tarjan(int x)
{
    //入 x 时，盖戳，入栈
    dfn[x]=low[x]=++tot;
    int child=0;
    for(auto y:g[x])
    {
        if(!dfn[y])//若 y 尚未访问
        {
            tarjan(y);
            low[x]=min(low[x],low[y]);//回 x 时更新 low
            if(low[y]>=dfn[x])//判割点
            {
                child++;
                if(x!=root||child>1) cut[x]=1;//根结点两棵子树
            }
        }
        else   //y 已访问
        low[x]=min(low[x],dfn[y]);//更新 low
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int n,m,x,y,cnt;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
```

```
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i])
        {
            root=i;
            tarjan(i);
        }
    }
    cnt=0;
    for(int i=1;i<=n;i++) if(cut[i]) cnt++;
    cout<<cnt<<"\n";
    for(int i=1;i<=n;i++) if(cut[i]) cout<<i<<" ";
    cout<<"\n";
}
```

12.拓扑排序

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n,x,y;
int w[10010],in[10010],res[10010];
vector<int>g[10010];
void solve()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>x>>y;
        w[x]=y;
        while(1)
        {
            cin>>y;
            if(y==0) break;
            g[y].push_back(x);
            in[x]++;
        }
    }
    queue<int>q;
```

```
    for(int i=1;i<=n;i++)
    if(!in[i])
    {
        q.push(i);
        res[i]=w[i];
    }

    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        for(auto v:g[u])
        {
            res[v]=max(res[v],res[u]+w[v]);
            in[v]--;
            if(!in[v]) q.push(v);
        }
    }
    int ans=0;
    for(int i=1;i<=n;i++) ans=max(res[i],ans);
    cout<<ans<<"\n";
}

signed main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
    //cin>>T;
    while(T--) solve();
}
```

13. 网络最大流

```
#include <bits/stdc++.h>
using namespace std;
#define M 10010
#define N 210
#define int long long
int h[N],d[N],cur[N],s,t;
int idx=1;
struct edge
{
```

```
    int v,c,ne;
}e[M];
void add(int a,int b,int c)
{
    e[++idx]={b,c,h[a]};
    h[a]=idx;
}
bool bfs()//对点分层，找增广路
{
    memset(d,0,sizeof d);
    queue<int>q;
    q.push(s);
    d[s]=1;
    while(q.size())
    {
        int u=q.front();
        q.pop();
        for(int i=h[u];i;i=e[i].ne)
        {
            int v=e[i].v;
            if(d[v]==0&&e[i].c)
            {
                d[v]=d[u]+1;
                q.push(v);
                if(v==t) return true;
            }
        }
    }
    return false;
}
int dfs(int u,int mf)//多路增广
{
    if(u==t) return mf;
    int sum=0;
    for(int i=cur[u];i;i=e[i].ne)
    {
        cur[u]=i;//当前弧优化
        int v=e[i].v;
        if(d[v]==d[u]+1&&e[i].c)
        {
            int f=dfs(v,min(mf,e[i].c));
            e[i].c-=f;
            e[i^1].c+=f;//更新残留网
            sum+=f;//累加 u 的流出流量
```

```
                mf-=f;//减少 u 的剩余流量
                if(!mf) break;//余量优化
            }
        }
        if(sum==0) d[u]=0;//残枝优化
        return sum;
}
int dinic()//累加可行流
{
        int flow=0;
        while(bfs())
        {
            memcpy(cur,h,sizeof h);
            flow+=dfs(s,1e9);
        }
        return flow;
}
void solve()
{
        int n,m,u,v,w;
        cin>>n>>m>>s>>t;
        for(int i=1;i<=m;i++)
        {
            cin>>u>>v>>w;
            add(u,v,w);
            add(v,u,0);
        }
        cout<<dinic()<<"\n";
}

signed main()
{
        ios::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        int T=1;
//      cin>>T;
        while(T--) solve();
}
```

四、 数据结构

1. ST 表

```
//ST 表，快速查询区间最大值/最小值
int f[100010][22];
for(int i=1;i<=n;i++) cin>>f[i][0];//初始化区间[i,i]的最大值
for(int i=1;i<=21;i++) for(int j=1;j+(1<<i)-1<=n;j++)
f[j][i]=max(f[j][i-1],f[j+(1<<(i-1))][i-1]);//动态规划更新

while(m--)
{
	cin>>l>>r;
	k=log2(r-l+1);//求区间长度
	cout<<max(f[l][k],f[r-(1<<k)+1][k])<<"\n";//输出区间最大值
}
```

2. 并查集

```
for(int i=1;i<=n;i++) fa[i]=i;
void merge(int x,int y)
{
	fa[find(x)]=find(y);
	/*
	还可以启发式合并
	*/
}
int find(int x)
{
	if(x==fa[x]) return x;
	return fa[x]=find(fa[x]);
}
```

3. 优先队列

```
priority_queue<int,vector<int>,greater<int>>q; //升序
priority_queue<int,vector<int>,less<int>>q; //降序
```

4. 树状数组

```
int lowbit(int x)
{
	return x&-x;
}
void update(int i,int x)
{
	while(i<=n)
	{
		tr[i]+=x;
		i+=lowbit(i);
	}
}
int query(int i)
{
	int res=0;
	while(i)
	{
		res+=tr[i];
		i-=lowbit(i);
	}
	return res;
}


void update(int x,int y,int k,int cr)//二维点（x，y）加 k，所属类别为 cr
{
	for(int i=x;i<=n;i+=lowbit(i))
	for(int j=y;j<=m;j+=lowbit(j))
	t[i][j][cr]+=k;
}
/*
cin>>x1>>x2>>y11>>y2>>c;
cout<<sum(x2,y2,c)-sum(x1-1,y2,c)-sum(x2,y11-1,c)+sum(x1-1,y11-1,c)<<"\n";
*/
int sum(int x,int y,int cr)//求区间和
{
	int res=0;
	for(int i=x;i;i-=lowbit(i))
	for(int j=y;j;j-=lowbit(j))
	res+=t[i][j][cr];
	return res;
```

```
}
```

5. 线段树

```
#define lc p<<1
#define rc p<<1|1
struct node
{
    int l,r,sum,add;//维护区间和、还可以维护区间 max，min，gcd，等等
}tr[N<<2];

void pushup(int p)//向上更新
{
    tr[p].sum=tr[lc].sum+tr[rc].sum;
}

void pushdown(int p)//向下更新
{
    if(tr[p].add)
    {
        tr[lc].sum+=tr[p].add*(tr[lc].r-tr[lc].l+1);
        tr[rc].sum+=tr[p].add*(tr[rc].r-tr[rc].l+1);
        tr[lc].add+=tr[p].add;
        tr[rc].add+=tr[p].add;
        tr[p].add=0;
    }
}
void build(int p,int l,int r)
{
    tr[p]={l,r,w[l],0};
    if(l==r) return ;
    int m=l+r>>1;
    build(lc,l,m);
    build(rc,m+1,r);
    pushup(p);
}
void update(int p,int x,int k)//单点修改
{//根节点进入，将节点[x,x]更改为 k，并更改其所有祖先
    if(tr[p].l==x&&tr[p].r==x)
    {
        tr[p].sum+=k;//更改
        return ;
    }
```

```
    int m=tr[p].l+tr[p].r>>1;//递归找到叶节点[x,x]
    if(x<=m) update(lc,x,k);
    else update(rc,x,k);

    tr[p].sum=tr[lc].sum+tr[rc].sum;//回溯
}
int query(int p,int x,int y)//区间求和
{
    if(x<=tr[p].l&&tr[p].r<=y) return tr[p].sum;//全部覆盖，直接返回
    int m=tr[p].l+tr[p].r>>1;//不是完全覆盖，裂开
    pushdown(p);
    int sum=0;
    if(x<=m) sum+=query(lc,x,y);//左子树有部分覆盖
    if(y>m) sum+=query(rc,x,y);//右子树有部分覆盖
    return sum;
}

void updatelr(int p,int x,int y,int k)
{
    if(x<=tr[p].l&&tr[p].r<=y)//覆盖则修改
    {
        tr[p].sum+=(tr[p].r-tr[p].l+1)*k;
        tr[p].add+=k;
        return ;
    }
    int m=tr[p].l+tr[p].r>>1;//不覆盖裂开
    pushdown(p);
    if(x<=m) updatelr(lc,x,y,k);
    if(y>m) updatelr(rc,x,y,k);
    pushup(p);
}
```

6.莫队算法

```
#include <bits/stdc++.h>
using namespace std;
int n,m,k,B,c[50010],a[50010];
int ans[50010];
int sum;
struct node
{
    int l,r,id;
```

```
}q[50010];
bool cmp(node q1,node q2)//按 l/B、r 排序
{
    if(q1.l/B!=q2.l/B) return q1.l<q2.l;
    return q1.r<q2.r;
}
void add(int x)//扩展一个数
{
    sum-=c[x]*c[x];
    c[x]++;
    sum+=c[x]*c[x];
}
void del(int x)//删除一个数
{
    sum-=c[x]*c[x];
    c[x]--;
    sum+=c[x]*c[x];
}
void solve()
{

    cin>>n>>m>>k;
    B=sqrt(n);//块的大小

    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=m;i++)
    {
        cin>>q[i].l>>q[i].r;
        q[i].id=i;
    }
    sort(q+1,q+1+m,cmp);
    for(int i=1,l=1,r=0;i<=m;i++)
    {
        while(l>q[i].l) add(a[--l]);//左扩展
        while(r<q[i].r) add(a[++r]);//右扩展
        while(l<q[i].l) del(a[l++]);//左删除
        while(r>q[i].r) del(a[r--]);//右删除
        ans[q[i].id]=sum;
    }
    for(int i=1;i<=m;i++) cout<<ans[i]<<"\n";

}
int main()
{
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
//  cin>>T;
    while(T--) solve();
}
```

7.可持久化线段树，区间第 k 小

```
#include <bits/stdc++.h>
using namespace std;
#define N 200010
#define lc(x) tr[x].l
#define rc(x) tr[x].r
int a[N];
struct node
{
    int l,r,s;
}tr[N*22];
int root[N],idx;
void build(int &x,int l,int r)
{
    x=++idx;//赋值，记得&
    if(l==r) return ;
    int m=l+r>>1;
    build(lc(x),l,m);
    build(rc(x),m+1,r);
}
//x 为前一版本，y 为当前版本
void insert(int x,int &y,int l,int r,int v)
{
    y=++idx;
    tr[y]=tr[x];
    tr[y].s++;
    if(l==r) return ;
    int m=l+r>>1;//双指针同步搜索
    if(v<=m) insert(lc(x),lc(y),l,m,v);
    else insert(rc(x),rc(y),m+1,r,v);
}
//x 为前一版本，y 为当前版本
int query(int x,int y,int l,int r,int k)
```

```
{
    if(l==r) return l;
    int m=l+r>>1;//双指针同步搜索
    int s=tr[lc(y)].s-tr[lc(x)].s;
    if(k<=s) return query(lc(x),lc(y),l,m,k);
    else return query(rc(x),rc(y),m+1,r,k-s);
}
vector<int>v;
int getid(int x)
{
    //从 1 开始
    return lower_bound(v.begin(),v.end(),x)-v.begin()+1;
}
void solve()
{
    int n,m,l,r,k;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        v.push_back(a[i]);
    }
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    int vn=v.size();
    build(root[0],1,vn);
    for(int i=1;i<=n;i++) insert(root[i-1],root[i],1,vn,getid(a[i]));

    for(int i=1;i<=m;i++)
    {
        cin>>l>>r;
        k=(r-l+1+1)/2;
        cout<<v[query(root[l-1],root[r],1,vn,k)-1]<<"\n";
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T=1;
    //cin>>T;
    while(T--) solve();
}
```

8.倍增思想（ST 表、LCM、二进制优化多重背包等）

```
for(int i=1;i<=n;i++)
{
    if(sum[n]-sum[i-1]<=k) f[i][0]=n+1;
    else
    {
        while(sum[j]-sum[i-1]<=k) j++;
        f[i][0]=j;
    }
}

for(int i=0;i<=19;i++) f[n+1][i]=n+1;

for(int j=1;j<=19;j++)
{
    for(int i=1;i<=n;i++)
    {
        f[i][j]=f[f[i][j-1]][j-1];9.
    }
}
```

9.重链剖分

```
#include <bits/stdc++.h>
using namespace std;
#define N 1000010
int n,p[N],fa[N];
vector<int>g[N];
int in[N],len[N],son[N];
vector<int>lp;
void dfs1(int x)//重链剖分
{
    for(auto y:g[x])
    {
        dfs1(y);
        if(len[y]>len[son[x]]) son[x]=y;
    }
    len[x]=len[son[x]]+1;
}
void dfs2(int x,int l)
{
```

```
        if(!son[x]) lp.push_back(l);
        else
        {
            dfs2(son[x],l+1);
            for(auto y:g[x])
            if(y!=son[x]) dfs2(y,1);
        }
}
void solve()
{
        cin>>n;
        lp.clear();
        for(int i=1;i<=n;i++) son[i]=len[i]=0,g[i].clear();
        for(int i=2;i<=n;i++)
        {
            int x;
            cin>>x;
            g[x].push_back(i);
        }

        dfs1(1);
        dfs2(1,1);

        sort(lp.begin(),lp.end(),greater<int>());
        int p=lp.size(),ans=p;
        for(int i=0;i<p;i++) ans=min(ans,lp[i]+i);
        cout<<ans<<"\n";
}
signed main()
{
        ios::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        int T=1;
        cin>>T;
        while(T--) solve();
}
```

10.线段树动态开点

```
#include <bits/stdc++.h>
```

```
using namespace std;

// 定义 long long 类型为默认的整数类型
#define int long long
// 常量 N 表示数组大小，M 为树的最大节点数，inf 为极大值
const int N = 2e5 + 5, M = N * 60, inf = 1e9;

// 全局变量
int a[N], n, m, ls[M], rs[M], sz[M]; // 数组 a 存储输入，ls/rs 为线段树左右子节点，sz 记录节点内元素数量
int sum[M];                          // sum 数组记录线段树节点的区间和值
int nod, rt;                         // nod 为当前可用的新节点编号，rt 为线段树根节点
int msum = 0;                        // 负数部分的总和

// 更新线段树，o 表示当前节点编号，l 和 r 是区间，v 是值，t 是增量（+1 或 -1）
void modify(int &o, int l, int r, const int &v, const int &t) {
    if (!o) o = ++nod;                  // 如果节点不存在，则分配新节点
    if (l == r) {                       // 如果到达叶子节点
        sz[o] += t;                     // 更新叶子节点的元素数量
        sum[o] += t * l;                // 更新叶子节点的区间和值
        return;
    }
    const int mid = (l + r) >> 1;       // 计算中点
    if (v <= mid)                       // 如果值落在左区间
        modify(ls[o], l, mid, v, t); // 递归更新左子树
    else                                // 如果值落在右区间
        modify(rs[o], mid + 1, r, v, t); // 递归更新右子树
    sz[o] = sz[ls[o]] + sz[rs[o]];     // 更新当前节点的元素数量
    sum[o] = sum[ls[o]] + sum[rs[o]]; // 更新当前节点的区间和值
}

// 在当前线段树中查询最多能取出的元素数量，s 为可用的总和
int query(int o, int l, int r, const int &s) {
    if (s >= sum[o]) return sz[o];     // 如果总和 s 足够覆盖整个节点的值，返回所有元素数量
    if (l == r) {                       // 如果到达叶子节点
        if (s > sz[o] * l)              // 如果 s 大于该节点全部元素的值
            return sz[o];               // 返回全部元素数量
        return s / l;                   // 否则返回能取出的数量
    }
    const int mid = (l + r) >> 1;       // 计算中点
    if (s >= sum[ls[o]])                // 如果 s 足够覆盖左子树的总和值
        return sz[ls[o]] + query(rs[o], mid + 1, r, s - sum[ls[o]]); // 继续查询右子树
```

```
    return query(ls[o], l, mid, s);   // 否则只查询左子树
}

// 插入元素到结构中，正数插入线段树，负数加到 msum
void ins(int x) {
    if (x > 0)
        modify(rt, 1, inf, x, 1);      // 正数更新线段树
    else
        msum -= x;                     // 负数更新 msum
}

// 从结构中删除元素，正数从线段树中删除，负数从 msum 减去
void del(int x) {
    if (x > 0)
        modify(rt, 1, inf, x, -1);     // 正数删除线段树
    else
        msum += x;                     // 负数更新 msum
}

// 处理一组测试数据
void solve() {
    int n, q;                          // n 表示数组大小，q 表示查询次数
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {     // 读取数组 a，并初始化结构
        cin >> a[i];
        ins(a[i]);                     // 插入元素到结构中
    }
    while (q--) {                      // 处理 q 次查询
        int x, v;
        cin >> x >> v;                 // x 为修改的索引，v 为新值
        del(a[x]);                     // 删除原来的值
        a[x] = v;                      // 更新数组 a
        ins(a[x]);                     // 插入新的值
        cout << query(rt, 1, inf, msum) + 1 << "\n"; // 输出结果
    }
}

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);           // 优化输入输出
    int T = 1;                         // 测试用例数量，默认为 1
    // cin >> T;
    while (T--) solve();               // 依次处理每个测试用例
}
```

11.线段树优化 bitset

```
#include <bits/stdc++.h> // 包含所有标准库头文件
using namespace std; // 使用标准命名空间

const int N=5e3+10, M=5e5+10; // 定义常量 N 和 M，N 为最大节点数，M 为 bitset 的大小
map<int, vector<int>> mp; // 定义一个 map，用于存储每个数的插入位置

struct SGT // 定义线段树结构体
{
    vector<int> t[N*4]; // 定义线段树数组，每个节点保存一个向量，用于存储区间内的数

    // 插入函数，用于在区间内插入元素 x
    void ins(int p, int l, int r, int ql, int qr, int x)
    {
        if (ql <= l && r <= qr) // 如果当前区间完全在插入区间 [ql, qr] 内
        {
            t[p].push_back(x); // 将元素 x 插入当前节点 p 的向量中
            return; // 结束当前递归
        }
        int m = l + r >> 1; // 计算中点 m
        if (ql <= m) ins(p * 2, l, m, ql, qr, x); // 如果插入区间左端点在左半区间内，递归左子区间
        if (qr > m) ins(p * 2 + 1, m + 1, r, ql, qr, x); // 如果插入区间右端点在右半区间内，递归右子区间
    }

    // 解决函数，用于处理查询区间内的所有元素
    void solve(int p, int l, int r, bitset<M> f)
    {
        bitset<M> g = f; // 复制 f 到局部变量 g，表示当前节点的状态集合
        for (int x : t[p]) g |= g << x; // 遍历当前节点的每个数，将它们加入集合 g 中
        if (l == r) // 如果到达叶子节点
        {
            cout << g.count() - 1 << "\n"; // 输出集合 g 中的 1 的个数减 1，即为不同子集的数量
            return; // 结束当前递归
        }
        else
```

```
        {
            int m = l + r >> 1; // 计算中点 m
            solve(p * 2, l, m, g); // 递归处理左子区间
            solve(p * 2 + 1, m + 1, r, g); // 递归处理右子区间
        }
    }
} sgt; // 定义线段树实例 sgt

void solve() // 主解函数
{
    int n, op, x; // 定义整数 n 表示操作数，op 表示操作类型，x 表示操作数值
    cin >> n; // 输入操作数 n
    for (int i = 1; i <= n; i++) // 遍历每个操作
    {
        cin >> op >> x; // 输入操作类型 op 和数值 x
        if (op == 1) // 如果操作类型为 1，表示插入操作
        {
            mp[x].push_back(i); // 将 x 的插入位置 i 加入到 map 的对应向量中
        }
        else // 如果操作类型为 2，表示删除操作
        {
            int t = mp[x].back(); // 获取 x 的最后插入位置 t
            sgt.ins(1, 1, n, t, i - 1, x); // 将 x 插入区间 [t, i-1]，调用线段树插入函数
            mp[x].pop_back(); // 弹出 x 的最后插入位置
        }
    }

    for (auto x : mp) // 遍历尚未匹配的插入操作
    {
        int cnt=0;
        for (auto y : x.second)
        {
            sgt.ins(1, 1, n, y, n, x.first); // 将尚未匹配的 x 插入区间 [y, n]
        }
    }

    bitset<M> f; // 定义一个 bitset 类型的集合 f
    f[0] = 1; // 初始化 f，表示当前状态包含 0
    sgt.solve(1, 1, n, f); // 调用线段树的 solve 函数，开始处理查询
}

int main() // 主函数
{
    ios::sync_with_stdio(0); // 优化输入输出
```

```
    cin.tie(0); cout.tie(0); // 取消 cin 和 cout 的绑定，提高执行效率
    int T = 1; // 定义测试用例数 T，初始化为 1
    // cin >> T; // 如果有多组测试用例，解除注释此行
    while (T--) solve(); // 逐组测试用例调用 solve 函数
}
```