图论

树重心

一棵无根树的重心是子树大小最大值最小的节点。

时间复杂度 $\mathcal{O}(n)$。

```
int sz[N],centroid[2];
void get_centroid(int u,int fa,int tot) {
    int maxx=0;
    sz[u]=1;
    for(int v:adj[u]) {
        if(v!=fa) {
            get_centroid(v,u,tot);
            sz[u]+=sz[v];
            maxx=max(maxx,sz[v]);
        }
    }
    maxx=max(maxx,tot-sz[u]);
    if(maxx<=tot/2) centroid[centroid[0]!=0]=u;
}
```

最近公共祖先

基于树链剖分，预处理复杂度 $\mathcal{O}(n)$，查询复杂度 $\mathcal{O}(\log n)$，常数较小。

```
namespace hpd {
    constexpr int N=1e5+10; // ***
    vector<int> adj[N];
    int dep[N],sz[N],top[N],p[N],hch[N];

    void dfs1(int u,int fa,int d) {
        dep[u]=d,p[u]=fa,sz[u]=1;
        for(int v:adj[u]) {
            if(v==fa) continue;
            dfs1(v,u,d+1);
            sz[u]+=sz[v];
            if(sz[hch[u]]<sz[v]) hch[u]=v;
        }
    }

    void dfs2(int u,int t) {
        top[u]=t;
        if(!hch[u]) return;
        dfs2(hch[u],t);
        for(int v:adj[u])
            if(v!=p[u]&&v!=hch[u]) dfs2(v,v);
    }

    int lca(int x,int y) {
        while(top[x]!=top[y]) {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            x=p[top[x]];
        }
        if(dep[x]<dep[y]) swap(x,y);
        return y;
    }

    void init() {
        dfs1(1,-1,1); dfs2(1,1);
    }

    void clear(int n) {
        fill(hch, hch+n+1, 0);
    }
}
```

基于倍增，预处理复杂度 $\mathcal{O}(n\log n)$，查询复杂度 $\mathcal{O}(\log n)$。

倍增常数相比树剖更大，但是维护路径信息更方便。

```
constexpr int N=1e5+10,M=__lg(N);
int fa[N][M+1],dep[N];
```

```
void lca_init(int u,int p) {
    dep[u]=dep[p]+1;
    for(int v:adj[u]) {
        if(v==p) continue;
        fa[v][0]=u;
        for(int i=1;i<=M;i++)
            fa[v][i]=fa[fa[v][i-1]][i-1];
        lca_init(v,u);
    }
}

int lca(int u,int v) {
    if(dep[u]<dep[v]) swap(u,v);
    for(int k=M;~k;k--)
        if(dep[fa[u][k]]>=dep[v])
            u=fa[u][k];
    if(u==v) return u;
    for(int k=M;~k;k--)
        if(fa[u][k]!=fa[v][k])
            u=fa[u][k],v=fa[v][k];
    return fa[u][0];
}
```

强连通分量

使用 Tarjan 算法求强连通分量，时间复杂度 $\mathcal{O}(n)$。

按照 `scc_cnt` 倒序遍历便是拓扑序。

```
namespace scc {
    int dfn[N],low[N],id[N],sz[N],scc_cnt,tsp;
    vector<int> stk;
    bool ins[N];

    void tarjan(int u) {
        dfn[u]=low[u]=++tsp;
        stk.push_back(u),ins[u]=1;
        for(int v:adj[u]) {
            if(!dfn[v]) {
                tarjan(v);
                low[u]=min(low[u],low[v]);
            }
            else if(ins[v]) low[u]=min(low[u],dfn[v]);
        }
        if(dfn[u]==low[u]) {
            scc_cnt++;
            int x;
            do {
                x=stk.back();
                stk.pop_back();
                ins[x]=0;
```

```
                id[x]=scc_cnt;
                sz[scc_cnt]++;
            } while(x!=u);
        }
    }

    void init(int n) {
        if(tsp) {
            scc_cnt=tsp=0;
            for(int i=1;i<=n;i++) dfn[i]=sz[i]=0;
        }
        for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
    }
} using scc::id,scc::sz,scc::scc_cnt;
```

网络流

最大流

Dinic 算法，复杂度 $\mathcal{O}(n^2m)$。在单位网络运作的复杂度为 $\mathcal{O}(m\sqrt{n})$。

```
template<typename cap,int vertex,int edge> struct Flow {
    constexpr static int N=vertex,M=edge;
    constexpr static cap INF=cap(1)<<(8*sizeof(cap)-2);
    int e[M],ne[M],idx;
    int h[N],q[N],arc[N],d[N];
    cap f[M];
    int S,T=N-1;

    void add_edge(int a,int b,cap c) {
        e[idx]=b,f[idx]=c,ne[idx]=h[a],h[a]=idx++;
        e[idx]=a,f[idx]=0,ne[idx]=h[b],h[b]=idx++;
    }

    cap dfs(int u,cap lim) {
        if(u==T) return lim;
        cap flow=0;
        for(int i=arc[u];~i&&flow<lim;i=ne[i]){
            int v=e[i];
            arc[u]=i;
            if(f[i]&&d[v]==d[u]+1){
                cap t=dfs(v,min(f[i],lim-flow));
                if(!t) d[v]=-1;
                f[i]-=t,f[i^1]+=t,flow+=t;
            }
        }
        return flow;
    }

    bool bfs() {
```

```
        memset(d,-1,sizeof d);
        q[0]=S,arc[S]=h[S],d[S]=0;
        int hh=0,tt=1;
        while(hh<tt) {
            int ver=q[hh++];
            for(int i=h[ver];~i;i=ne[i]) {
                int t=e[i];
                if(f[i]&&d[t]==-1) {
                    d[t]=d[ver]+1;
                    arc[t]=h[t];
                    if(t==T) return 1;
                    q[tt++]=t;
                }
            }
        }
        return 0;
    }

    cap maxflow() {
        cap F=0,flow=0;
        while(bfs()) while(flow=dfs(S,INF)) F+=flow;
        return F;
    }

    void init() {
        idx=0;
        memset(h, -1, sizeof h);
    }

    Flow() { init(); }
};
```

费用流

EK 算法，复杂度 $\mathcal{O}(n^2m)$。

```
template<typename cap,typename cost,int vertex,int edge> struct Flow {
    constexpr static int N=vertex,M=edge,INF=cap(1)<<(8*sizeof(cap)-2);
    int S=0,T=N-1,idx;
    int ne[M],e[M];
    int h[N],q[N],pre[N];
    cap f[M],mf[N];
    cost d[N],w[M];
    bool inq[N];

    void add_edge(int a,int b,cap c,cost d) {
        e[idx]=b,f[idx]=c,w[idx]=d,ne[idx]=h[a],h[a]=idx++;
        e[idx]=a,f[idx]=0,w[idx]=-d,ne[idx]=h[b],h[b]=idx++;
    }

    bool spfa() {
```

```
        memset(d,0x3f,sizeof d);
        memset(mf,0,sizeof mf);
        int hh=0,tt=1;
        q[0]=S,d[S]=0,mf[S]=INF;
        while(hh!=tt) {
            int u=q[hh++];
            if(hh==N) hh=0;
            inq[u]=0;

            for(int i=h[u];~i;i=ne[i]) {
                int v=e[i];
                if(f[i]&&d[v]>d[u]+w[i]) {
                    d[v]=d[u]+w[i];
                    pre[v]=i;
                    mf[v]=min(mf[u],f[i]);
                    if(!inq[v]){
                        q[tt++]=v;
                        if(tt==N) tt=0;
                        inq[v]=1;
                    }
                }
            }
        }
        return mf[T]>0;
    }

    pair<cap,cost> maxflow() {
        cap flow=0; cost val=0;
        while(spfa()) {
            flow+=mf[T],val+=mf[T]*d[T];
            for(int i=T;i!=S;i=e[pre[i]^1]) {
                f[pre[i]]-=mf[T];
                f[pre[i]^1]+=mf[T];
            }
        }
        return {flow,val};
    }

    void init() {
        idx=0;
        memset(h, -1, sizeof h);
    }

    Flow() { init(); }
};
```

上下界可行流

无源汇上下界可行流

1. 建立虚拟的源汇点S',T'，对于任意一点 $u\in E$ ，当$x=\sum c_{下}(v,u)-\sum c_{下}(u,v)>0$ 时，从源点向u连一条容量为x的边，反之从u向汇点连接一条容量为$-x$的边

2. 原网络中的每条边的容量设为为$c_{上}(u,v)-c_{下}(u,v)$

对新网络G'跑最大流算法即得解。

有源汇上下界可行流

S,T可以看作两个特殊点，它们不满足流守恒。我们可以简单的建一条$T\to S$，容量下界为0,上界为∞的边使它们能够流守恒，这样就转化为了一个无源汇的上下界可行流问题。

有源汇上下界最大流

由于在做完一遍无源汇上下界可行流时，和 S',T' 相连的边都已经满流，所以$S\to T$ 的增广路上一定不包含 S',T'，所以我们不必拆除S',T'和与其相连的边，因为它们不影响结果，从S到T求一遍最大流再加上原本的可行流流量即可。注意求最大流的时候要拆掉新加的 $T\to S$ 的边，否则答案可能会偏大。

有源汇上下界最小流

和最大流类似，从S到T的流量表示剩下还可以追加的流量，在求最小流的时候，我们反向搜索从T到S的最大流，表示可以从可行流中退回的部分流量。注意同样要删去额外加上的$T\to S$的边，否则会退回无穷大的流量。

最小割模型

最大权闭合图

闭合图指一个对于有向图 $G=(V,E)$ 的点集 V'，使得 V' 中的所有点的出边指向 V' 中的点。最大权闭合图即权值和最大的闭合图。

将最大权闭合图转化为流网络

对于原图 G 中的边，将这些边的容量设为 ∞；建立一个虚拟源点 S，向每个权值 w_i 为正的边连一条容量为 w_i 的边；建立一虚拟汇点 T，每个权值 w_i 为负的点向T连一条容量为 $|w_i|$ 的边。

计算最大权值和

最大权闭合图的点权之和=所有正权值之和-最小割的容量

$$w(V')=\sum_{v\in V^+}w_v-c[S,T]$$

建图之后使用最大流算法求解

方案

求出最小割后 S 集合点即为选择的点。

最小点权覆盖集

对于一个带点权的图 $G=(E,V)$，一个点覆盖集是指在集合 V 中选择一个子集 V'，使得集合 E 中的每一条边的两个端点至少有一个在 V' 中。最小点权覆盖集即点权和最小的点覆盖集。

最小点权覆盖集是一个npc问题，网络流对于该问题的有效解法只对二分图有效。

最小点权覆盖集和二分图的简单割一一对应。

建图&求解

如果想要将问题转化为此模型，需要题目给定一个二分图或者想办法建出一个二分图。

对于原图的边，边的容量设为 ∞ ，以保证割为简单割，再建立虚拟源点 S 和虚拟汇点 T ，源汇点和原图中的点建容量等于点权的边。

根据最小点权覆盖集的点权和等于最小割的容量，在流网络上跑最大流算法即得解。

方案

最小割中割边的两端点 s,t ， 因为是简单割，s,t 必定存在一个源点或汇点，若 $s=S$ 则覆盖集选择的点为 t ，若 $t=T$ 则选择的是 s 。通过 dfs 找出割边再判断是 s/t 即可。

最大点权独立集

对于一个带点权的图 $G=(E,V)$ ，点独立集是指在集合 V 中选择一个子集 V'，使得V' 中的点两两没有一条边相连。最大点权独立集即点权和最大的点独立集。

最大点独立集和最小点权覆盖集是一个对偶问题，可以使用反证法证明点覆盖集的补集为点独立集。点独立集和点覆盖集的点权和为所有点权和sum，所以最大点权独立集的点权和等于$sum-$最小点权覆盖集的点权和。

建图&求解

同样的，最大点权独立集也是一个npc问题，建出二分图跑最小点权覆盖集再求差即可。