# 图论

## 建图

### 基本建图

```
const int N=1e5+5;
int head[N],cntt;
struct Edge{
    int to,next,val;
}edge[2*N];
void add(int u,int v,int x){
    edge[++cntt].to=v;
    edge[cntt].val=x;
    edge[cntt].next=head[u];
    head[u]=cntt;
}
```

### 线段树优化建图

```
//区间与区间连边：建立虚点，入树->虚点，虚点->出树
#define ls (p<<1)
#define rs ((p<<1)|1)
const int N=2e5+5;
int head[8*N],cntt=0;
struct Edge{
    int to,next,val;
}edge[40*N];    //边数量开40倍
void add(int u,int v,int x){
    edge[++cntt].to=v;
    edge[cntt].val=x;
    edge[cntt].next=head[u];
    head[u]=cntt;
}
int n,pos[8*N];   //点的个数为8n
void build(int p,int l,int r){
    if(l==r){
        pos[l]=p;       //对应叶子结点的编号
        return;
    }
    add(p,ls,0);       //出树
    add(p,rs,0);
    add(ls+4*n,p+4*n,0);    //入树
    add(rs+4*n,p+4*n,0);
    int mid=(l+r)>>1;
    build(ls,l,mid);
    build(rs,mid+1,r);
}
void Add1(int p,int l,int r,int u,int val,int x,int y){   //u->[x,y]
    if(x<=l&&y>=r){
        add(pos[u]+4*n,p,val);
```

```
        return;
    }
    int mid=(l+r)>>1;
    if(x<=mid)Add1(ls,l,mid,u,val,x,y);
    if(y>mid)Add1(rs,mid+1,r,u,val,x,y);
}
void Add2(int p,int l,int r,int u,int val,int x,int y){     //[x,y]->u
    if(x<=l&&y>=r){
        add(p+4*n,pos[u],val);
        return;
    }
    int mid=(l+r)>>1;
    if(x<=mid)Add2(ls,l,mid,u,val,x,y);
    if(y>mid)Add2(rs,mid+1,r,u,val,x,y);
}
void solve(){
    build(1,1,n);
    for(int i=1;i<=n;i++){
        add(pos[i],pos[i]+4*n,0);
        add(pos[i]+4*n,pos[i],0);
    }
}
```

### 前后缀优化建图

```
//常用于优化2-sat，a[i]向a'[j]连边 (i!=j)
void solve(){
    while(m--){
        int siz;
        cin>>siz;
        for(int i=1;i<=siz;i++){
            cin>>a[i];
            add(a[i]+n,a[i]+2*n,1);      //前缀
            add(a[i]+3*n,a[i],1);        //后缀
        }
        for(int i=1;i<siz;i++){
            add(a[i]+2*n,a[i+1]+2*n,1);      //打通前缀
            add(a[i+1]+3*n,a[i]+3*n,1);      //打通后缀
            add(a[i]+2*n,a[i+1],1);      //前缀连边
            add(a[i+1]+n,a[i]+3*n,1);    //后缀连边
        }
    }
}
```

## 欧拉回路

```
int in[N],out[N];
int vis[2*N],cur[N],ans[N];
int n,m,cnt,op;           // op=1 无向，op=2 有向
void dfs(int x){
    for(int &i=cur[x];i;i=edge[i].next){       // 当前弧优化
        int y=edge[i].to;
        int tmp=i;
        if(!vis[tmp>>1]){
```

```
            vis[tmp>>1]=1;
            dfs(y);
            ans[++cnt]=tmp;
        }
    }
    /*
    while(!G[x].empty()){
        auto [y,id]=G[x].back();
        G[x].pop_back();
        if(vis[id])continue;
        vis[id]=1;
        dfs(y);
        ans[++cnt]=id;
    }
    */
}
int Euler(){         // 返回是否存在欧拉回路
    for(int i=1;i<=n;i++){
        if(op==1&&in[i]%2==1)return 0;
        if(op==2&&in[i]!=out[i])return 0;
    }
    cnt=0;
    for(int i=1;i<=n;i++)cur[i]=head[i];
    for(int i=1;i<=n;i++)if(in[i]||out[i]){
        dfs(i);
        break;
    }
    if(cnt<m)return 0;
    else{
        cout<<"YES"<<endl;
        for(int i=cnt;i>=1;i--){
            if(ans[i]%2==1)ans[i]=-(ans[i]>>1);
            else ans[i]=ans[i]/2;
            cout<<ans[i]<<' ';
        }
        return 1;
    }
}
void solve(){
    cin>>op>>n>>m;
    cntt=1;
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        add(u,v,1);
        if(op==1){
            add(v,u,1);
            in[u]++,in[v]++;
        }
        else{
            cntt++;
            out[u]++,in[v]++;
        }
    }
    if(!Euler()){
```

```
        cout<<"NO"<<endl;
        return;
    }
}
```

# 最短路

## Dijkstra

```
struct node{
    int x,dis;
    bool operator<(const node&a)const{return dis>a.dis;}
};
int vis[N],dis[N],n;
int dij(){
    priority_queue<node>q;
    memset(dis,0x3f,sizeof(dis));
    q.push({1,0});
    dis[1]=0;
    while(!q.empty()){
        int x=q.top().x;
        q.pop();
        if(vis[x])continue;
        vis[x]=1;
        for(int i=head[x];i;i=edge[i].next){
            int y=edge[i].to,v=edge[i].val;
            if(vis[y])continue;
            if(dis[y]>dis[x]+v){
                dis[y]=dis[x]+v;
                q.push({y,dis[y]});
            }
        }
    }
    return dis[n];
}
```

## Johnson

```
// 有负权边
int n,m,h[N],vis[N],cnt[N],dis[N];
bool spfa(int S){
    for(int i=0;i<=n;i++){
        vis[i]=cnt[i]=0,h[i]=1e9;
    }
    queue<int>q;
    q.push(S);
    h[S]=0,vis[S]=1;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        vis[x]=0;
        for(int i=head[x];i;i=edge[i].next){
            int y=edge[i].to,v=edge[i].val;
            if(h[y]>h[x]+v){
```

```
                h[y]=h[x]+v;
                if(!vis[y]){
                    q.push(y);
                    vis[y]=1;
                    cnt[y]++;
                    if(cnt[y]>n)return false;       // 负环
                }
            }
        }
    }
    return true;
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);
    }
    for(int i=1;i<=n;i++)add(0,i,0);    // 源点
    if(!spfa(0)){       // 负环
        cout<<-1<<endl;
        return;
    }
    for(int i=1;i<=n;i++){
        for(int j=head[i];j;j=edge[j].next){
            int y=edge[j].to;
            edge[j].val+=h[i]-h[y];     // 偏移
        }
    }
    for(int i=1;i<=n;i++){
        dij(i);     // n 遍 dij
        for(int j=1;j<=n;j++){
            if(dis[j]!=1e9)dis[j]-=h[i]-h[j];   // 减去偏移
            cout<<dis[j]<<' ';
        }
        cout<<endl;
    }
}
```

## Floyd

```
// 最小环dp
for(int k=1;k<=n;k++){
    for(int i=1;i<k;i++){       //枚举k-i的出边
        if(dis[k][i]==INT_MAX||dp[i][k]==INT_MAX)continue;
        if(dis[k][i]+dp[i][k]==mn){
            ans+=cnt[i][k];
        }
        else if(dis[k][i]+dp[i][k]<mn){
            mn=dis[k][i]+dp[i][k];
            ans=cnt[i][k];
        }
    }
    for(int i=1;i<=n;i++){          //以k为中转点，更新最短路
```

```
        for(int j=1;j<=n;j++){
            if(i==j||i==k||j==k)continue;
            if(dp[i][k]==INT_MAX||dp[k][j]==INt_MAX)continue;
            if(dp[i][j]==dp[i][k]+dp[k][j]){
                cnt[i][j]+=cnt[i][k]*cnt[k][j];
            }
            else if(dp[i][k]+dp[k][j]<dp[i][j]){
                dp[i][j]=dp[i][k]+dp[k][j];
                cnt[i][j]=cnt[i][k]*cnt[k][j];
            }
        }
    }
}
// 若环长度>=3
for(int k=1;k<=n;k++){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i!=j && i!=k && j!=k){
                ans=min(ans,dis[i][k]+dis[k][j]+dp[i][j]);
            }
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i!=j && i!=k && j!=k){
                dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);
            }
        }
    }
}
```

## 差分约束

```
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        add(0,i,0);
    }
    while(m--){
        int u,v,x;
        cin>>u>>v>>x;
        add(v,u,x);
    }
    queue<int>q;
    vector<int>dis(n+1,0x3f3f3f3f),cnt(n+1);
    vector<bool>vis(n+1);
    q.push(0);
    dis[0]=0;
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        vis[tmp]=0;
        for(int i=head[tmp];i;i=edge[i].next){
            int y=edge[i].to,x=edge[i].val;
            if(dis[tmp]+x<dis[y]){
```

```
                cnt[y]++;
                if(cnt[y]>n){
                    cout<<"NO"<<endl;
                    return;
                }
                dis[y]=dis[tmp]+x;
                if(!vis[y]){
                    q.push(y);
                    vis[y]=1;
                }
            }
        }
    }
    for(int i=1;i<=n;i++)cout<<dis[i]<<' ';
    return;
}
```

## 最小生成树

```
// 最小瓶颈树（最大边最小）一定是最小生成树，最小生成树不一定是最小瓶颈树
const int N=1e5+5;
int f[N];
struct node{
    int u,v,x;
    bool operator <(const node&aa)const{
        return x<aa.x;
    }
};
vector<node>tr;
int find(int x){return f[x]==x?x:f[x]=Find(f[x]);}
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)f[i]=i;
    while(m--){
        int u,v,w;
        cin>>u>>v>>w;
        tr.push_back({u,v,w});
    }
    sort(tr.begin(),tr.end());
    ll ans=0;
    for(auto i:tr){
        if(find(i.u)!=find(i.v)){
            f[find(i.u)]=find(i.v);
            ans+=i.x;
        }
    }
    return;
}
```

## kruskal重构树

```
int ff[N],a[N];
int find(int x){return ff[x]==x?x:ff[x]=find(ff[x]);};
struct node{
    int u,v,x;
    bool operator<(const node&aa)const{
        return x<aa.x;
    }
};
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n*2;i++)ff[i]=i;
    vector<node>vv;
    while(m--){
        int u,v,x;
        cin>>u>>v>>x;
        vv.push_back({u,v,x});
    }
    sort(vv.begin(),vv.end());
    tot=n;
    for(auto i:vv){
        int u=i.u,v=i.v,x=i.x;
        u=find(u),v=find(v);
        if(u==v)continue;
        a[++tot]=x;
        ff[tot]=tot;
        ff[u]=ff[v]=tot;
        add(u,tot),add(tot,u);
        add(v,tot),add(tot,v);
    }
    t=log2(tot);
    set<int>ss;
    for(int i=1;i<=n;i++){
        ss.insert(find(i));
    }
    for(auto i:ss)init(i);
    while(q--){
        int u,v;
        cin>>u>>v;
        int x=lca(u,v);
        cout<<a[x]<<endl;
    }
    return;
}
```

## 最小斯坦纳树

```
// n 个点的图，给 k 个点，求最小树使得 k 个点联通
// 复杂度 O(n*3^k+nlog(n)*2^k)
int dp[N][1<<10];   // 以 i 为根的树，包含 j 点集的最小代价
int vis[N];
void solve(){
```

```
    int n,m,k;
    cin>>n>>m>>k;
    for(int i=1;i<=m;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);
        add(v,u,x);
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<(1<<k);j++){
            dp[i][j]=1e10;
        }
    }
    for(int i=0;i<k;i++){
        int x;
        cin>>x;
        dp[x][1<<i]=0;
    }
    for(int j=1;j<(1<<k);j++){
        queue<int>q;
        for(int i=1;i<=n;i++){      // 根的度大于1，拆成两个部分
            for(int p=j;p;p=(p-1)&j){       // 枚举子集
                dp[i][j]=min(dp[i][j],dp[i][p]+dp[i][j-p]);
            }
            vis[i]=0;
            if(dp[i][j]<1e10)q.push(i),vis[i]=1;
        }
        // 采用spfa转移，可以使用dijkstra
        // dp[x][j]=min(dp[x][j],dp[y][j]+val[x][y])
        while(!q.empty()){      // 根的度为1,
            int x=q.front();
            q.pop();
            vis[x]=0;
            for(int i=head[x];i;i=edge[i].next){
                int y=edge[i].to,val=edge[i].val;
                if(dp[y][j]>dp[x][j]+val){
                    dp[y][j]=dp[x][j]+val;
                    if(!vis[y])vis[y]=1,q.push(y);
                }
            }
        }
    }
    int ans=1e10;
    for(int i=1;i<=n;i++)ans=min(ans,dp[i][(1<<k)-1]);
    cout<<ans<<endl;
}
```

## 三/四元环计数

```
// 三元环计数   O(m\sqrt{m})
vector<int>G[N];
int deg[N],vis[N];
void solve(){
    int n,m;
    cin>>n>>m;
```

```
    vector<pair<int,int>>vec;
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        deg[u]++,deg[v]++;
        vec.emplace_back(u,v);
    }
    // 建有向无环图，计数 <u,v>,<v,w>,<u,w>
    for(auto[u,v]:vec){
        if(deg[u]>deg[v] || (deg[u]==deg[v] && u>v))swap(u,v);
        G[u].push_back(v);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        for(auto j:G[i])vis[j]=i;
        for(auto j:G[i])
            for(auto k:G[j])if(vis[k]==i)ans++;
    }
    cout<<ans<<endl;
}


// 四元环计数   O(m\sqrt{m})
vector<int>G[N],G2[N];
int deg[N],cnt[N];
void solve(){
    int n,m;
    cin>>n>>m;
    vector<pair<int,int>>vec;
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        deg[u]++,deg[v]++;
        G[u].push_back(v);
        G[v].push_back(u);
        vec.emplace_back(u,v);
    }
    for(auto[u,v]:vec){
        if(deg[u]>deg[v] || (deg[u]==deg[v] && u>v))swap(u,v);
        G2[u].push_back(v);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        for(auto j:G[i])
            for(auto k:G2[j])if(deg[i]<deg[k] || (deg[i]==deg[k] &&
i<k))ans+=cnt[k]++;
        for(auto j:G[i])
            for(auto k:G2[j])cnt[k]=0;
    }
    cout<<ans<<endl;
}
```

# LCA

倍增LCA

```
void init(int root){
    queue<int>q;
    q.push(root);
    dis[root]=1;
    int t=(int)log2(n);
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        for(int i=head[tmp];i;i=edge[i].next){
            int y=edge[i].to;
            if(dis[y])continue;
            dis[y]=dis[tmp]+1;
            f[y][0]=tmp;
            q.push(y);
            for(int j=1;j<=t;j++){
                f[y][j]=f[f[y][j-1]][j-1];
            }
        }
    }
}
int Lca(int u,int v){
    if(dis[u]>dis[v])swap(u,v);
    t=log2(n);
    for(int i=t;i>=0;i--){
        if(dis[f[v][i]]>=dis[u])v=f[v][i];
    }
    if(u==v)return u;
    for(int i=t;i>=0;i--){
        if(f[u][i]!=f[v][i]){
            u=f[u][i],v=f[v][i];
        }
    }
    return f[u][0];
}
```

树剖LCA

```
int siz[N],top[N],son[N],f[N],dep[N];
void dfs1(int x){
    siz[x]=1;
    dep[x]=dep[f[x]]+1;
    for(auto y:G[x]){
        if(y==f[x])continue;
        f[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
void dfs2(int x,int id){
```

```
    top[x]=id;
    if(son[x])dfs2(son[x],id);
    for(auto y:G[x]){
        if(y==f[x]||y==son[x])continue;
        dfs2(y,y);
    }
}
int lca(int u,int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]])v=f[top[v]];
        else u=f[top[u]];
    }
    return (dep[u]<dep[v])?u:v;
}
```

dfn 序 LCA

```
// nlog(n) 预处理，O(1)求lca
int dfn[N],id;
int st[N][25];
void dfs(int x,int f){
    dfn[x]=++id;
    st[dfn[x]][0]=f;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        dfs(y,x);
    }
}
void init(){
    for(int j=1;j<=__lg(N-1);j++){
        for(int i=1;i+(1<<j)-1<N;i++){
            int x=st[i][j-1],y=st[i+(1<<(j-1))][j-1];
            st[i][j]=dfn[x]<dfn[y]?x:y;
        }
    }
}
int lca(int u,int v){
    if(u==v)return u;
    u=dfn[u],v=dfn[v];
    if(u>v)swap(u,v);
    int d=__lg(v-u);
    int x=st[u+1][d],y=st[v-(1ll<<d)+1][d];
    return dfn[x]<dfn[y]?x:y;
}
```

## 基环树最远两点距离

```
//基环树+单调队列
const int N=2e6+5;
vector<pair<int,int>>G[N];
int d[N],vis[N],on_cir[N];
int dp[N][2],mx[N],t;
void dfs(int x,int f){
```

```
    dp[x][0]=dp[x][1]=0;
    for(auto [y,v]:G[x]){
        if(y==f || on_cir[y])continue;
        dfs(y,x);
        if(dp[y][0]+v>dp[x][0]){
            dp[x][1]=dp[x][0];
            dp[x][0]=dp[y][0]+v;
        }
        else dp[x][1]=max(dp[x][1],dp[y][0]+v);
    }
    t=max(t,dp[x][0]+dp[x][1]);
}
vector<int>vec;
int val[N];
void dfs2(int x){
    vec.emplace_back(x);
    vis[x]=1;
    for(auto[y,v]:G[x]){
        if(vis[y])continue;
        val[vec.size()]=val[vec.size()-1]+v;
        dfs2(y);
    }
}
void solve(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        int x,v;
        cin>>x>>v;
        G[i].emplace_back(x,v);
        G[x].emplace_back(i,v);
        d[i]++,d[x]++;
    }
    queue<int>q;
    for(int i=1;i<=n;i++){
        if(d[i]==1)q.push(i);
    }
    while(!q.empty()){
        int x=q.front();
        q.pop();
        vis[x]=1;
        for(auto [y,v]:G[x]){
            if(vis[y])continue;
            d[y]--;
            if(d[y]<=1){
                q.push(y);
            }
        }
    }
    for(int i=1;i<=n;i++)if(!vis[i])on_cir[i]=1;

    for(int i=1;i<=n;i++){
        if(on_cir[i]){
            t=0;
            dfs(i,i);
```

```
            dp[i][1]=t;
        }
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        if(!vis[i]){
            vec.clear();
            val[0]=t=0;
            dfs2(i);
            int m=vec.size();
            int s=val[m-1],c=0;
            for(auto[y,v]:G[vec[0]]){
                if(y==vec[m-1])s+=v,c++;
            }
            if(c==2)s-=val[m-1];
            for(int
i=0;i<m;i++)vec.emplace_back(vec[i]),val[i+m]=val[i]+s,t=max(t,dp[vec[i]][1]);
            deque<int>dq;
            for(int i=0;i<2*m;i++){
                while(!dq.empty() && dq.front()<=i-m)dq.pop_front();
                int x=vec[i];
                if(i>0)t=max(t,val[i]+dp[x][0]-
val[dq.front()]+dp[vec[dq.front()]][0]);
                while(!dq.empty() && dp[vec[dq.back()]][0]-val[dq.back()]<=dp[x]
[0]-val[i])dq.pop_back();
                dq.push_back(i);
            }
            ans+=t;
        }
    }
    cout<<ans<<endl;
}
```

## 长链剖分 (k级祖先)

```
vector<int>G[N],s1[N],f1[N];
int f[N][25],d[N],dep[N],top[N],son[N],high_bit[N];
// d[i] 表示点 i 的深度，dep[i] 表示点 i 所在链的深度
void dfs1(int x){
    d[x]=dep[x]=d[f[x][0]]+1;
    for(auto y:G[x]){
        f[y][0]=x;
        for(int i=1;i<=20;i++)f[y][i]=f[f[y][i-1]][i-1];
        dfs1(y);
        if(dep[y]>dep[x]){
            dep[x]=dep[y];
            son[x]=y;
        }
    }
}
void dfs2(int x,int pos){
    top[x]=pos;
    if(x==pos){
```

```
        int now=x;
        for(int i=0;i<=dep[x]-d[x];i++){
            s1[x].push_back(now);
            now=son[now];
        }
        now=x;
        for(int i=0;i<=dep[x]-d[x];i++){
            f1[x].push_back(now);
            now=f[now][0];
        }
    }
    if(son[x])dfs2(son[x],pos);
    for(auto y:G[x]){
        if(y!=son[x])dfs2(y,y);
    }
}
int ask(int x,int k){       // x 的 k 级祖先
    if(k==0)return x;
    x=f[x][high_bit[k]],k-=(1<<high_bit[k]);
    k-=d[x]-d[top[x]],x=top[x];
    if(k>=0)return f1[x][k];
    else return s1[x][-k];
}
void solve(){
    int n,rt;
    cin>>n;
    for(int i=1;i<=n;i++){
        int x;
        cin>>x;
        if(x==0)rt=i;
        else G[x].push_back(i);
    }
    dfs1(rt);
    dfs2(rt,rt);
    high_bit[0]=-1;
    for(int i=1;i<=n;i++){
        high_bit[i]=high_bit[i-1];
        if(i==(i&-i))high_bit[i]++;
    }
}
```

## 一般图最大匹配

```
// O(nm + n^2logn) 一般图最大匹配，mat[i] 表示和 i 匹配的点
vector<int>G[N];
int f[N];
int find(int x){return f[x]==x?f[x]:f[x]=find(f[x]);}
int mat[N],vis[N],dep[N],lk[N];
int lca(int u,int v){
    u=find(u),v=find(v);
    while(u!=v){
        if(dep[u]<dep[v])swap(u,v);
        u=find(lk[mat[u]]);
    }
    return u;
```

```
}
void blossom(int u,int v,int p,queue<int>&q){
    while(find(u)!=p){
        lk[u]=v;
        v=mat[u];
        if(vis[v]==0){
            vis[v]=1;
            q.push(v);
        }
        f[u]=f[v]=p;
        u=lk[v];
    }
}
void aug(int p,int n){
    queue<int>q;
    for(int i=1;i<=n;i++)f[i]=i,vis[i]=-1;
    q.push(p);
    vis[p]=1,dep[p]=0;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        for(auto y:G[x]){
            if(vis[y]==-1){
                vis[y]=0;
                lk[y]=x;
                dep[y]=dep[x]+1;
                if(mat[y]==0){
                    int u=y,v=x,tmp;
                    while(v!=-1 && u!=0){
                        tmp=mat[v];
                        mat[u]=v,mat[v]=u;
                        u=tmp,v=lk[u];
                    }
                    return;
                }
                vis[mat[y]]=1;
                dep[mat[y]]=dep[x]+2;
                q.push(mat[y]);
            }
            else if(vis[y]==1 && find(y)!=find(x)){
                int fa=lca(x,y);
                blossom(x,y,fa,q);
                blossom(y,x,fa,q);
            }
        }
    }
}
void get_match(int n){
    for(int i=1;i<=n;i++){
        if(mat[i])continue;
        for(auto y:G[i]){
            if(!mat[y]){
                mat[i]=y,mat[y]=i;
                break;
            }
```

```
        }
    }
    for(int i=1;i<=n;i++)
        if(mat[i]==0)aug(i,n);
    int cnt=0;
    for(int i=1;i<=n;i++)if(mat[i])cnt++;
    cnt/=2;
    cout<<cnt<<endl;
    for(int i=1;i<=n;i++)cout<<mat[i]<<' ';
}
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    get_match(n);
}
```

## 虚树

```
int siz[N],top[N],son[N],f[N],dep[N],dfn[N];
int id=0;
int pre[N];
void dfs1(int x){       //先预处理一个转移数组pre
    siz[x]=1;
    dfn[x]=++id;
    dep[x]=dep[f[x]]+1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x])continue;
        f[y]=x;
        pre[y]=min(pre[x],edge[i].val);
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
//省略树剖lca
int vis[N],dp[N];
int Stack[N],Top;
void ins(int x){        //插入关键点
    if(Top<=1){
        Stack[++Top]=x;
        return;
    }
    int lca=Lca(x,Stack[Top]);
    if(lca==Stack[Top]){
        Stack[++Top]=x;
        return;
    }
    while(Top>1&&dfn[lca]<=dfn[Stack[Top-1]]){
```

```
        add(Stack[Top-1],Stack[Top],1);
        add(Stack[Top],Stack[Top-1],1);
        Top--;
    }
    if(Stack[Top]!=lca){
        add(lca,Stack[Top],1);
        add(Stack[Top],lca,1);
        Stack[Top]=lca;
    }
    Stack[++Top]=x;
}
void dfs(int x,int f){      //虚树dp
    dp[x]=0;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        dfs(y,x);
        dp[x]+=min(dp[y],pre[y]);
    }
    if(vis[x])dp[x]=pre[x];          //最后再返回，一定要把树遍历完
    else dp[x]=min(dp[x],pre[x]);
    head[x]=0;
}
void solve(){
    int n,m;
    cin>>n;
    pre[1]=INT_MAX;
    for(int i=1;i<n;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);
        add(v,u,x);
    }
    dfs1(1);
    dfs2(1,1);  //树剖
    cin>>m;
    while(m--){
        int k;
        cin>>k;
        cntt=0;         //清空
        vector<int>a(k+1);
        for(int i=1;i<=k;i++){
            cin>>a[i];
            vis[a[i]]=1;        //标记关键点
        }
        sort(a.begin()+1,a.end(),[&](int x,int y){return dfn[x]<dfn[y];});
        Top=1;
        Stack[Top]=1;
        for(int i=1;i<=k;i++)ins(a[i]);          //入栈
        while(Top>1){
            add(Stack[Top-1],Stack[Top],1);
            add(Stack[Top],Stack[Top-1],1);
            Top--;
        }
        dfs(1,1);
```

```
        cout<<dp[1]<<endl;
        for(int i=1;i<=k;i++)vis[a[i]]=0;       //清空
    }
}
```

# prufer序列

## 树转序列

```
int f[N],deg[N];
int prufer[N];  // 存所有被删节点的父亲
void dfs(int x){
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x])continue;
        f[y]=x;
        dfs(y);
    }
}
void solve(){
    dfs(n);
    int now=0;      // 指针维护叶子最小编号
    for(int i=1;i<=n;i++){
        if(deg[i]==1){
            now=i;
            break;
        }
    }
    int leaf=now;   // 即将删的编号最小的叶子
    for(int i=1;i<=n-2;i++){
        prufer[i]=f[leaf];
        deg[f[leaf]]--;
        if(deg[f[leaf]]==1 && f[leaf]<now)
            leaf=f[leaf];
        else{
            now++;
            while(deg[now]>1)now++;
            leaf=now;
        }
    }
}
```

## 序列转树

```
int f[N],deg[N];
int prufer[N];  // 存所有被删节点的父亲
void solve(){
    for(int i=1;i<=n;i++)deg[i]=1;
    for(int i=1;i<=n-2;i++){
        cin>>prufer[i];
        deg[prufer[i]]++;
    }
    int now=0;
    for(int i=1;i<=n;i++){
```

```
        if(deg[i]==1){
            now=i;
            break;
        }
    }
    int leaf=now;
    for(int i=1;i<=n-2;i++){
        add(leaf,prufer[i],1);
        add(prufer[i],leaf,1);
        deg[prufer[i]]--;
        if(deg[prufer[i]]==1 && prufer[i]<now){
            leaf=prufer[i];
        }
        else{
            now++;
            while(deg[now]>1)now++;
            leaf=now;
        }
    }
    add(leaf,n,1);
    add(n,leaf,1);
}
```

# 弦图

## 极大团/颜色数

```
vector<int>G[N],vec[N];
int deg[N];     // 完美消除序列上，与 x 相连并且在 x 后的点的个数
int  vis[N],n,m;
void mcs(){
    int p=0;
    for(int i=1;i<=n;i++)vec[0].push_back(i);
    for(int i=1;i<=n;i++){
        int now=0;
        while(!now){
            while(!vec[p].empty() && vis[vec[p].back()]){
                vec[p].pop_back();
            }
            if(vec[p].empty())p--;
            else now=vec[p].back();
        }
        vis[now]=1;
        for(auto k:G[now]){
            if(!vis[k]){
                deg[k]++;
                if(deg[k]>p)p++;
                vec[deg[k]].push_back(k);
            }
        }
    }
}
void solve(){
    cin>>n>>m;
    while(m--){
```

```
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    mcs();
    int col=0;  // 弦图的颜色数 | 极大团大小
    for(int i=1;i<=n;i++)col=max(col,deg[i]+1);
    cout<<col<<endl;
}
```

# Tarjan

## 割点

```
int n,m;
int low[N],dfn[N];
int ans[N],Tm;      //ans标记是否为割点
void tarjan(int x,int f,bool root){
    int cnt=0;
    dfn[x]=low[x]=++Tm;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        if(!dfn[y]){
            tarjan(y,x,0);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x]&&root==0)ans[x]=1;         //非根的割点判定
            cnt++;
        }
        else low[x]=min(low[x],dfn[y]);
    }
    if(root==1&&cnt>=2)ans[x]=1;         //根的割点判定
}
void solve(){
    cin>>n>>m;
    while(m--){
        int u,v;
        cin>>u>>v;
        add(u,v);
        add(v,u);
    }
    for(int i=1;i<=n;i++){
        if(!dfn[i])tarjan(i,i,1);
    }
    int num=0;
    for(int i=1;i<=n;i++)num+=ans[i];
    cout<<num<<endl;
    for(int i=1;i<=n;i++){
        if(ans[i])cout<<i<<' ';
    }
    return;
}
```

## 割边（桥）

```
const int N=1e5+5;
int low[N],dfn[N];
int Tm;
set<pair<int,int>>cut;  //cut储存所有割边
void tarjan(int x,int lst){     //注意cntt从1开始
    dfn[x]=low[x]=++Tm;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(i==(lst^1))continue;     //判重边
        if(!dfn[y]){
            tarjan(y,i);
            low[x]=min(low[x],low[y]);
            if(low[y]>dfn[x]){      // 割边判定条件
                cut.emplace(min(x,y),max(x,y));
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
void solve(){
    int n,m;
    cin>>n>>m;
    cntt=1;
    while(m--){
        int u,v;
        cin>>u>>v;
        add(u,v);
        add(v,u);
    }
    for(int i=1;i<=n;i++)if(!dfn[i])tarjan(i,0);
}
```

## 点双（圆方树）

```
int low[N],dfn[N];
int cut[N],Tm;      //cut标记是否为割点
int Stack[N],Top;
vector<int>Vdcc[N];
int id=0;
void tarjan(int x,int f,bool root){
    int cnt=0;
    dfn[x]=low[x]=++Tm;
    Stack[++Top]=x;
    if(head[x]==0){     //孤立点
        Vdcc[++id].push_back(x);
        return;
    }
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        if(!dfn[y]){
            tarjan(y,x,0);
```

```
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x]){
                if(root==0)cut[x]=1;
                cnt++;
                id++;
                while(true){
                    int tmp=Stack[Top--];
                    Vdcc[id].push_back(tmp);
//                    add2(n+id,tmp,1);     //建圆方树
//                    add2(tmp,n+id,1);     //注意：head2[x],edge2[x]
                    if(tmp==y)break;
                }
                Vdcc[id].push_back(x);
//                add2(n+id,x,1);
//                add2(x,n+id,1);
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
    if(root==1&&cnt>=2)cut[x]=1;        //根的割点判定
}
```

## 边双

```
vector<int>Edcc[N];
int pos[N];
void get_Edcc(int x,int id){
    pos[x]=id;
    Edcc[id].push_back(x);
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(pos[y])continue;
        if(cut.count(make_pair(min(x,y),max(x,y))))continue;
        get_Edcc(y,id);
    }
}
void solve(){
    cntt=1;
    for(int i=1;i<=n;i++)if(!dfn[i])tarjan(i,0);    //割边
    int id=0;
    for(int i=1;i<=n;i++)if(!pos[i])get_Edcc(i,++id);
}
```

## 强连通

```
// 缩点后注意判断自环和重边的情况
const int N=1e5+5;
int n,m,id;
int a[N],low[N],dfn[N],pos[N],val[N];
int Stack[N],vis[N],Top,Tm;
vector<int>G[N];    //强连通分量
void tarjan(int x){
    dfn[x]=low[x]=++Tm;
    Stack[++Top]=x;
```

```
    vis[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(vis[y]){
            low[x]=min(low[x],dfn[y]);
        }
    }
    if(low[x]==dfn[x]){
        id++;
        while(true){
            int tmp=Stack[Top];
            pos[tmp]=id;
            val[id]+=a[tmp];
            G[id].push_back(tmp);

            vis[tmp]=0;     //出栈
            Top--;
            if(tmp==x)break;
        }
    }
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    while(m--){
        int u,v;
        cin>>u>>v;
        add(u,v,1);
    }
    for(int i=1;i<=n;i++){
        if(dfn[i]==0)tarjan(i);
    }
    for(int i=1;i<=id;i++){
        for(auto j:G[i]){
            cout<<j<<' ';
        }
        cout<<endl;
    }
    return;
}
```

## SCC缩点

```
int n,m,id;
int a[N],low[N],dfn[N],pos[N],val[N];
int Stack[N],vis[N],top,Tm;
vector<int>G[N];    //缩点后新DAG
int deg[N],ans,dp[N];
void tarjan(int x){          //求出所有强连通
    dfn[x]=low[x]=++Tm;
    Stack[++top]=x;
```

```
    vis[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(vis[y]){
            low[x]=min(low[x],dfn[y]);
        }
    }
    if(low[x]==dfn[x]){
        id++;
        while(true){
            int tmp=Stack[top];
            pos[tmp]=id;
            val[id]+=a[tmp];
            vis[tmp]=0;
            top--;
            if(tmp==x)break;
        }
    }
}
void tuopu(){
    queue<int>q;
    for(int i=1;i<=id;i++){
        if(deg[i]==0)q.push(i);
        dp[i]=val[i];
    }
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        ans=max(ans,dp[tmp]);
        for(auto y:G[tmp]){
            dp[y]=max(dp[y],dp[tmp]+val[y]);
            deg[y]--;
            if(deg[y]==0)q.push(y);
        }
    }
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    while(m--){
        int u,v;
        cin>>u>>v;
        add(u,v,1);
    }
    for(int i=1;i<=n;i++){
        if(dfn[i]==0)tarjan(i);
    }
    for(int i=1;i<=n;i++){          //缩点建DAG
        for(int j=head[i];j;j=edge[j].next){
            int y=edge[j].to;
            if(pos[i]==pos[y])continue;
```

```
            G[pos[i]].push_back(pos[y]);
            deg[pos[y]]++;
        }
    }
    tuopu();
    cout<<ans<<endl;
    return;
}
```

# 网络流

## dinic 最大流

```
// 判断一条边(u->v)是否流满: vis[u]==1 && vis[v]==0
int S,T,dis[N],cur[N];
int bfs(){
    // 注意 memset，数组不要过大
    memset(dis,0,sizeof(dis));
    queue<int>q;
    dis[S]=1;
    cur[S]=head[S];
    q.push(S);
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        for(int i=head[tmp];i;i=edge[i].next){
            int y=edge[i].to;
            if(dis[y]||edge[i].val==0)continue;
            q.push(y);
            dis[y]=dis[tmp]+1;
            cur[y]=head[y];
            if(y==T)return 1;
        }
    }
    return 0;
}
int dfs(int x,int f){
    if(x==T)return f;
    int rest=f;
    for(int i=cur[x];i;i=edge[i].next){
        cur[x]=i;
        int y=edge[i].to;
        if((dis[y]==dis[x]+1)&&(edge[i].val)){
            int tmp=dfs(y,min(edge[i].val,rest));
            if(tmp==0)dis[y]=0;
            edge[i].val-=tmp;
            edge[i^1].val+=tmp;      //加边的cntt要从1开始
            rest-=tmp;
            if(rest==0)return f;
        }
    }
    dis[x]=0;
    return f-rest;
}
int dinic(){
```

```
    int ans=0;
    while(bfs()){
        ans+=dfs(S,INT_MAX);
    }
    return ans;
}
void add1(int u,int v,int x){add(u,v,x),add(v,u,0);}
void solve(){
    cin>>n>>m>>S>>T;
    cntt=1;
    while(m--){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);
        add(v,u,0);
    }
    cout<<dinic()<<endl;
}
```

## hlpp 最大流

```
int S,T,all;
int h[N],e[N],gap[2*N],inq[N];
struct cmp{
    bool operator()(int a,int b)const{return h[a]<h[b];}
};
priority_queue<int,vector<int>,cmp>pq;
int bfs(){
    for(int i=1;i<=all;i++)h[i]=INT_MAX;
    queue<int>q;
    h[T]=0;
    q.push(T);
    while(!q.empty()){
        int x=q.front();
        q.pop();
        for(int i=head[x];i;i=edge[i].next){
            int y=edge[i].to;
            if(edge[i^1].val && h[y]>h[x]+1){
                h[y]=h[x]+1;
                q.push(y);
            }
        }
    }
    return h[S]!=INT_MAX;
}
void push(int x){
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to,val=edge[i].val;
        if(val && h[y]+1==h[x] && h[y]<INT_MAX){
            int d=min(e[x],val);
            edge[i].val-=d,edge[i^1].val+=d;
            e[x]-=d,e[y]+=d;
            if(y!=S && y!=T && !inq[y]){
                pq.push(y);
                inq[y]=1;
```

```
            }
            if(!e[x])break;
        }
    }
}
void relabel(int x){
    h[x]=INT_MAX;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to,val=edge[i].val;
        if(val && h[y]+1<h[x])h[x]=h[y]+1;
    }
}
int hlpp(){
    if(!bfs())return 0;
    h[S]=all;
    memset(gap,0,sizeof(gap));
    for(int i=1;i<=all;i++)if(h[i]<INT_MAX)gap[h[i]]++;
    for(int i=head[S];i;i=edge[i].next){
        int y=edge[i].to,val=edge[i].val;
        if(val && h[y]<INT_MAX){
            edge[i].val-=val,edge[i^1].val+=val;
            e[S]-=val,e[y]+=val;
            if(y!=S && y!=T && !inq[y]){
                pq.push(y);
                inq[y]=1;
            }
        }
    }
    while(!pq.empty()){
        int x=pq.top();
        pq.pop();
        inq[x]=0;
        push(x);
        if(e[x]){
            if(!--gap[h[x]])
                for(int i=1;i<=all;i++)
                    if(i!=S && i!=T && h[i]>h[x] && h[i]<all+1)h[i]=all+1;
            relabel(x);
            gap[h[x]]++;
            pq.push(x);
            inq[x]=1;
        }
    }
    return e[T];
}
void add1(int u,int v,int x){add(u,v,x),add(v,u,0);}
```

## 最小割可行边/必须边

```
int low[N],dfn[N],pos[N],id;
int Stack[N],vis[N],top,Tm;
void tarjan(int x){
    dfn[x]=low[x]=++Tm;
    Stack[++top]=x;
    vis[x]=1;
```

```
    for(int i=head[x];i;i=edge[i].next){
        if(edge[i].val==0)continue;
        int y=edge[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(vis[y])low[x]=min(low[x],dfn[y]);
    }
    if(low[x]==dfn[x]){
        id++;
        while(true){
            int tmp=Stack[top];
            pos[tmp]=id;
            vis[tmp]=0;
            top--;
            if(tmp==x)break;
        }
    }
}
void solve(){
    int n,m;
    cin>>n>>m>>S>>T;
    cntt=1;
    vector<pair<int,int>>query;
    for(int i=1;i<=m;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add1(u,v,x);
        query.emplace_back(u,v);
    }
    dinic();
    for(int i=1;i<=n;i++){
        if(!dfn[i])tarjan(i);
    }
    for(int i=0;i<m;i++){
        auto[u,v]=query[i];
        if(pos[u]!=pos[v] && edge[2*(i+1)].val==0){
            // 是最小割可行边
            // 这条边满流 && 两端不在同一个强连通分量
        }
        if(pos[u]==pos[S] && pos[v]==pos[T] && edge[2*(i+1)].val==0){
            // 是最小割必须边
            // 这条边满流 && 左端在 S 联通分量，右端在 T 联通分量
        }
    }
}
```

## 最小费用最大流

```
#define int long long
const int N=5e5+5,M=1e6+5;
int head[N],cntt=1;
struct Edge{
    int to,next;
```

```
    int val,cost;   //val为流，cost为费用
}edge[2*M];
void add(int u,int v,int x,int y){
    edge[++cntt].to=v;
    edge[cntt].val=x;
    edge[cntt].cost=y;
    edge[cntt].next=head[u];
    head[u]=cntt;
}

int all,S,T,fee=0;      //all为节点个数
int dis[N],vis[N],cur[N];
bool spfa(){
    for(int i=0;i<=all;i++){
        dis[i]=INT_MAX;
        vis[i]=0;
    }
    deque<int>q;
    q.push_back(S);
    dis[S]=0,vis[S]=1;
    cur[S]=head[S];
    while(!q.empty()){
        int tmp=q.front();
        q.pop_front();
        vis[tmp]=0;
        for(int i=head[tmp];i;i=edge[i].next){
            int y=edge[i].to;
            if(edge[i].val==0||dis[y]<=(dis[tmp]+edge[i].cost))continue;
            dis[y]=dis[tmp]+edge[i].cost;
            cur[y]=head[y];
            if(!vis[y]){
                if(!q.empty()&&dis[y]<dis[q.front()])q.push_front(y);
                else q.push_back(y);
            }
        }
    }
    return dis[T]!=INT_MAX;
}
int dfs(int x,int flow){
    if(x==T)return flow;
    int rest=flow;
    vis[x]=1;
    for(int i=cur[x];i;i=edge[i].next){
        cur[x]=i;
        int y=edge[i].to;
        int cost=edge[i].cost;
        if((dis[y]==(dis[x]+cost))&&(edge[i].val)&&(!vis[y])){
            int tmp=dfs(y,min(edge[i].val,rest));
            edge[i].val-=tmp;
            edge[i^1].val+=tmp;
            rest-=tmp;
            fee+=tmp*cost;
            if(rest==0)break;
        }
    }
```

```
    vis[x]=0;
    return flow-rest;
}
int dinic(){
    int ans=0;
    while(spfa()){
        for(int i=0;i<=all;i++)vis[i]=0;
        ans+=dfs(S,INT_MAX);
    }
    return ans;
}
void add1(int u,int v,int x,int y){add(u,v,x,y),add(v,u,0,-y);}

void solve(){
    cin>>n>>m>>S>>T;
    cntt=1;
    while(m--){
        int u,v,x,y;
        cin>>u>>v>>x>>y;
        add(u,v,x,y);
        add(v,u,0,-y);
    }
    all=n;
    int ans=dinic();
    cout<<ans<<' '<<fee<<endl;
}
```

## 上下界最小费用可行流

```
int d[N];   // 所有入边的下界-所有流出边的下界
            // d 为正，则 SS--i  |  d 为负，则 i--TT ，流量为 d 的绝对值
void solve(){
    int n,m,ans=0;
    cin>>n;
    int S0=1,T0=n+1;        // 原图的源汇
    cntt=1;
    for(int i=2;i<=n;i++){
        add(i,T0,INT_MAX,0);
        add(T0,i,0,0);
    }
    for(int u=1;u<=n;u++){
        cin>>m;
        while(m--){
            int v,x;
            cin>>v>>x;
            int low=1,upp=INT_MAX;      // 流量上下界
            d[u]-=low,d[v]+=low;
            ans+=1*x;       // 下界*费用
            // u-v 有一条下界为 1 ，上界为无穷，费用为 x 的边
            add(u,v,upp-low,x);     // u-v 连一条费用为x，流量为 upp-low 的边
            add(v,u,0,-x);
        }
    }
    S=n+2,T=n+3,all=T;    // 虚拟源汇点
    for(int i=1;i<=n;i++){
```

```
        if(d[i]>0){
            add(S,i,d[i],0);
            add(i,S,0,0);
        }
        if(d[i]<0){
            add(i,T,-d[i],0);
            add(T,i,0,0);
        }
    }
    add(T0,S0,INT_MAX,0);        // 有源汇时需要闭合
    add(S0,T0,0,0);
    dinic();         // 最小费用最大流
    cout<<fee+ans<<endl;
}
```

## 上下界最小可行流

```
int d[N];
void solve(){
    int n,m;
    cin>>n;
    int S0=n+1,T0=n+2;       // 原图源汇
    S=n+3,T=n+4,cntt=1;          // 虚拟源汇
    for(int i=1;i<=n;i++){
        add(S0,i,INT_MAX);
        add(i,S0,0);
        add(i,T0,INT_MAX);
        add(T0,i,0);
    }
    for(int u=1;u<=n;u++){
        cin>>m;
        while(m--){
            int v;
            cin>>v;
            int low=1,upp=INT_MAX;  // 上下界
            d[u]-=low,d[v]+=low;
            add(u,v,upp-low);   // 连流量为 upp-low 的边
            add(v,u,0);
        }
    }
    for(int i=1;i<=n;i++){
        if(d[i]>0){
            add(S,i,d[i]);
            add(i,S,0);
        }
        if(d[i]<0){
            add(i,T,-d[i]);
            add(T,i,0);
        }
    }
    dinic();         // 先流一遍
    add(T0,S0,INT_MAX);      // 再联通原图的源汇
    add(S0,T0,0);
    dinic();         // 最小可行流即为最后一条T0-S0边的流量，即S0-T0边的权值
    cout<<edge[cntt].val<<endl;
```

```
}
```

## 最小割树

```
// 无向图，多次询问两点的最小割
const int N=505,M=1505;
int head[N],cntt=0;
int head2[N],cntt2=0;       //最小割树
struct Edge{
    int to,next,val;
}edge[4*M],edge2[2*N];
void add(int u,int v,int x){
    edge[++cntt].to=v;
    edge[cntt].val=x;
    edge[cntt].next=head[u];
    head[u]=cntt;
}
void add2(int u,int v,int x){
    edge2[++cntt2].to=v;
    edge2[cntt2].val=x;
    edge2[cntt2].next=head2[u];
    head2[u]=cntt2;
}
int S,T,dis[N],cur[N];
int bfs(){
    memset(dis,0,sizeof(dis));
    queue<int>q;
    dis[S]=1;
    cur[S]=head[S];
    q.push(S);
    while(!q.empty()){
        int tmp=q.front();
        q.pop();
        for(int i=head[tmp];i;i=edge[i].next){
            int y=edge[i].to;
            if(dis[y]||edge[i].val==0)continue;
            q.push(y);
            dis[y]=dis[tmp]+1;
            cur[y]=head[y];
            if(y==T)return 1;
        }
    }
    return 0;
}
int dfs(int x,int f){
    if(x==T)return f;
    int rest=f;
    for(int i=cur[x];i;i=edge[i].next){
        cur[x]=i;
        int y=edge[i].to;
        if((dis[y]==dis[x]+1)&&(edge[i].val)){
            int tmp=dfs(y,min(edge[i].val,rest));
            if(tmp==0)dis[y]=0;
            edge[i].val-=tmp;
            edge[i^1].val+=tmp;     //加边的cntt要从1开始
```

```
            rest-=tmp;
            if(rest==0)return f;
        }
    }
    dis[x]=0;
    return f-rest;
}
int dinic(){
    int ans=0;
    while(bfs()){
        ans+=dfs(S,INT_MAX);
    }
    return ans;
}
int t[N],t1[N],t2[N];
void dvd(int l,int r){
    if(l==r)return;
    S=t[l],T=t[l+1];
    for(int i=2;i<=cntt;i+=2)edge[i].val+=edge[i^1].val,edge[i^1].val=0;
    int x=dinic();
    add2(S,T,x);
    int cnt1=0,cnt2=0;
    for(int i=l;i<=r;i++){
        if(dis[t[i]])t1[++cnt1]=t[i];
        else t2[++cnt2]=t[i];
    }
    for(int i=l;i<=l+cnt1-1;i++)t[i]=t1[i-l+1];
    for(int i=l+cnt1;i<=r;i++)t[i]=t2[i-l-cnt1+1];
    dvd(l,l+cnt1-1);
    dvd(l+cnt1,r);
}
int f[N][20],mn[N][20];
void dfs2(int x){
    for(int i=head2[x];i;i=edge2[i].next){
        int y=edge2[i].to;
        if(y==f[x][0])continue;
        dis[y]=dis[x]+1;
        f[y][0]=x;
        mn[y][0]=edge2[i].val;
        dfs2(y);
    }
}
void init(int n){
    for(int j=1;j<=9;j++){
        for(int i=1;i<=n;i++){
            f[i][j]=f[f[i][j-1]][j-1];
            mn[i][j]=min(mn[i][j-1],mn[f[i][j-1]][j-1]);
        }
    }
}
int query(int u,int v){     // 两点的最小割
    if(dis[u]<dis[v])swap(u,v);
    int del=dis[u]-dis[v];
    int ans=INT_MAX;
    for(int i=0;i<=9;i++){
```

```
        if(del&(1<<i)){
            ans=min(ans,mn[u][i]);
            u=f[u][i];
        }
    }
    if(u==v)return ans;
    for(int i=9;i>=0;i--){
        if(f[u][i]!=f[v][i]){
            ans=min(ans,mn[u][i]);
            ans=min(ans,mn[v][i]);
            u=f[u][i],v=f[v][i];
        }
    }
    ans=min(ans,min(mn[u][0],mn[v][0]));
    return ans;
}
void solve(){
    int n,m;
    cin>>n>>m;
    cntt=1;
    for(int i=1;i<=n;i++)t[i]=i;
    for(int i=1;i<=m;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);          // 无向图加四条边
        add(v,u,0);
        add(v,u,x);
        add(u,v,0);
    }
    dvd(1,n);
    memset(dis,0,sizeof(dis));
    dfs2(1);
    init(n);
    int q;
    cin>>q;
    while(q--){
        int x,y;
        cin>>x>>y;
        cout<<query(x,y)<<endl;
    }
}
```

## 2—SAT

```
void solve(){
    int n,m;
    cin>>n>>m;      // 1 - n 表示 0，1+n - 2n 表示 1
    while(m--){
        int u,v,x,y;
        cin>>u>>x>>v>>y;    //u==x||v==y
        if(x==0&&y==0){
            add(u+n,v,1);
            add(v+n,u,1);
        }
        if(x==1&&y==0){
```

```
            add(u,v,1);
            add(v+n,u+n,1);
        }
        if(x==0&&y==1){
            add(u+n,v+n,1);
            add(v,u,1);
        }
        if(x==1&&y==1){
            add(u,v+n,1);
            add(v,u+n,1);
        }
    }
    for(int i=1;i<=2*n;i++){
        if(!dfn[i])tarjan(i);   //强连通
    }
    for(int i=1;i<=n;i++){
        if(pos[i]==pos[i+n]){
            cout<<"No"<<endl;
            return;
        }
    }
    cout<<"Yes"<<endl;
    for(int i=1;i<=n;i++){
        if(pos[i]<pos[i+n])cout<<0<<' ';    //强连通编号小的更优
        else cout<<1<<' ';
    }
}
```

## 二分图最大匹配（匈牙利算法）

```
vector<int>G[N];
// mat[i] 表示右部第 i 个点匹配的左部点
int vis[N],mat[N];
int dfs(int u,int tag){
    if(vis[u]==tag)return 0;
    vis[u]=tag;
    for(auto v:G[u]){
        if(mat[v]==0 || dfs(mat[v],tag)){
            mat[v]=u;
            return 1;
        }
    }
    return 0;
}
void solve(){
    int n,m,e;      // n 是左部点数，m 是右部点数
    cin>>n>>m>>e;
    while(e--){
        int u,v;
        cin>>u>>v;
        G[u].emplace_back(v);
    }
    int ans=0;
    for(int i=1;i<=n;i++)if(dfs(i,i))ans++;
    cout<<ans<<endl;
```

```
}
```

## 二分图带权完美匹配（KM）

```
// O(n^3)
const int NN=505;
struct K_M{
    int n;
    int w[NN][NN],vis[NN];
    int lx[NN],ly[NN],linker[NN],slack[NN],pre[NN];
    void bfs(int k){
        int x,y=0,yy=0,del;
        memset(pre,0,sizeof(pre));
        for(int i=1;i<=n;i++)slack[i]=1e15;
        linker[y]=k;
        while(true){
            x=linker[y];
            del=1e15;
            vis[y]=1;
            for(int i=1;i<=n;i++){
                if(vis[i])continue;
                if(slack[i]>(lx[x]+ly[i]-w[x][i])){
                    slack[i]=lx[x]+ly[i]-w[x][i];
                    pre[i]=y;
                }
                if(slack[i]<del){
                    del=slack[i];
                    yy=i;
                }
            }
            for(int i=0;i<=n;i++){
                if(vis[i])lx[linker[i]]-=del,ly[i]+=del;
                else slack[i]-=del;
            }
            y=yy;
            if(linker[y]==-1)break;
        }
        while(y)linker[y]=linker[pre[y]],y=pre[y];
    }
    int KM(){       // 最大权
        memset(lx,0,sizeof(lx));
        memset(ly,0,sizeof(ly));
        memset(linker,-1,sizeof(linker));
        for(int i=1;i<=n;i++){
            memset(vis,0,sizeof(vis));
            bfs(i);
        }
        int ans=0;
        for(int i=1;i<=n;i++)ans+=w[linker[i]][i];
        return ans;
    }
};
```