图论

SPFA 相关

DFS 判负环

例题：求图中最小圈。

```
const int N=3030, M=20020;
const double eps=1e-10;

int n, m;
struct Edge{
    int to, next;
    double w;
}e[M];

int h[N], tot;

void add(int u, int v, double w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

struct Buf{
    int u, v;
    double w;
}buf[M];
```

```
double d[N];
bool vis[N], st[N];

bool spfa(int u, double x){
    vis[u]=true;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(d[go]>d[u]+e[i].w-x){
            d[go]=d[u]+e[i].w-x;
            if(vis[go] || spfa(go, x)) return true;
        }
    }
    vis[u]=false;
    return false;
}

bool ok(double x){
    rep(i,1,n) vis[i]=0, d[i]=0;
    rep(i,1,n) if(spfa(i, x)) return true;
    return false;
}

int main(){
    memset(h, -1, sizeof h);
    read(n), read(m);

    rep(i,1,m){
        int u, v; read(u), read(v);
        double w; scanf("%lf", &w);
        add(u, v, w);
    }

    double l=-1e7, r=1e7;
    while(l+eps<r){
        double mid=(l+r)/2;
        if(ok(mid)) r=mid;
        else l=mid;
    }

    printf("%.8lf\n", l);

    return 0;
}
```

差分约束

例题：

你需要构造一个整数集合 Z，使得 $\forall i \in [1,n]$，Z 中满足 $a_i \le x \le b_i$ 的整数 x 不少于 c_i 个。

求这样的整数集合 Z 最少包含多少个数。

```
const int n=50001, N=5e4+50, M=3*N;

struct Edge{
    int to, w, next;
```

```
}e[M];

int h[N], tot;

void add(int u, int v, int w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

int m;

int d[N];
bool vis[N];
int spfa(){
    memset(d, 0xcf, sizeof d);
    d[0]=0;
    queue<int> q;
    q.push(0);
    vis[0]=true;

    while(q.size()){
        auto u=q.front(); q.pop();
        vis[u]=false;

        for(int i=h[u]; ~i; i=e[i].next){
            int go=e[i].to;
            if(d[go]<d[u]+e[i].w){
                d[go]=d[u]+e[i].w;
                if(!vis[go]){
                    vis[go]=true;
                    q.push(go);
                }
            }
        }
    }

    return d[n];
}

int main(){
    memset(h, -1, sizeof h);
    cin>>m;
    rep(i,1,m){
        int u, v, w; read(u), read(v), read(w);
        add(u, v+1, w);
    }
    rep(i,1,n) add(i-1, i, 0), add(i, i-1, -1);

    cout<<spfa()<<endl;

    return 0;
}
```

连通分量

SCC

```
int dfn[N], low[N], ts;
int stk[N], top;
bool in_stk[N];
int id[N];
int scc_cnt;

void tarjan(int u){
    dfn[u]=low[u]=++ts;
    in_stk[u]=true;
    stk[++top]=u;

    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!dfn[go]){
            tarjan(go);
            low[u]=min(low[u], low[go]);
        }else if(in_stk[go]) low[u]=min(low[u], dfn[go]);
    }

    if(dfn[u]==low[u]){
        scc_cnt++;
        int y;
        do{
            y=stk[top--];
            in_stk[y]=false;
            id[y]=scc_cnt;
        }while(y!=u);
    }
}
```

边双连通分量

```
int dfn[N], low[N], ts;
int stk[N], top;
int id[N], bcc_cnt;
bool is_bridge[M];

void tarjan(int u, int from){  // 起始点和从前而来的边
    dfn[u]=low[u]=++ts;
    stk[++top]=u;

    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!dfn[go]){
            tarjan(go, i);
            low[u]=min(low[u], low[go]);
            if(dfn[u]<low[go]) is_bridge[i]=is_bridge[i^1]=true;
        }
        else if(i!=(from^1)) // 非反向边
            low[u]=min(low[u], dfn[go]);
    }
```

```
    if(dfn[u]==low[u]){
        ++bcc_cnt;
        int y;
        do{
            y=stk[top--];
            id[y]=bcc_cnt;
        }while(y!=u);
    }
}
```

点双连通分量

```
int dfn[N], low[N], ts;
int stk[N], top;
int bcc_cnt; // v-bcc cnt
vector<int> bcc[N];
bool cut[N];
int root;

void tarjan(int u){
    dfn[u]=low[u]=++ts;
    stk[++top]=u;

    if(u==root && h[u]==-1){
        bcc[++bcc_cnt].push_back(u);
        return;
    }

    int cnt=0;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!dfn[go]){
            tarjan(go);
            low[u]=min(low[u], low[go]);

            if(dfn[u]<=low[go]){
                cnt++;
                if(u!=root || cnt>1) cut[u]=true;

                ++bcc_cnt;
                int y;
                do{
                    y=stk[top--];
                    bcc[bcc_cnt].push_back(y);
                }while(y!=go);
                bcc[bcc_cnt].push_back(u);
            }
        }
        else low[u]=min(low[u], dfn[go]);
    }
}
```

基环树

例题：求基环树森林直径和。

```
const int N=3e6+5, M=N<<1;

struct Edge{
    int to, w, next;
}e[M];

int h[N], tot;

void add(int u, int v, int w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

int n;
int res, del;

int dfn[N], ts, fa[N];
vector<int> Lo, Le;
bool inLo[N];
void findLo(int u){
    dfn[u]=++ts;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa[u]) continue;

        if(!dfn[go]) fa[go]=u, findLo(go);
        else if(dfn[go]>dfn[u]){
            Le.pb(e[i].w);
            for(; go!=fa[u]; go=fa[go]) Lo.pb(go), inLo[go]=true;
        }
    }
}

int D[N];
void getD(int u, int fa, int &res){
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa || inLo[go]) continue;

        getD(go, u, res);
        res=max(res, D[u]+e[i].w+D[go]);
        D[u]=max(D[u], e[i].w+D[go]);
    }
}

int q[N];
int cal(vector<int> &w, vector<int> &Le){
    int res=0;
    int n=w.size();
    int r=-1, l=0;
    w.insert(end(w), all(w)), Le.insert(end(Le), all(Le));
    rep(i,1,Le.size()-1) Le[i]+=Le[i-1];
```

```
    q[++r]=0;
    rep(i,1,w.size()-1){
        while(r>=l && i-q[l]>=n) l++;
        res=max(res, w[i]+Le[i]+w[q[l]]-Le[q[l]]);
        while(r>=l && w[i]-Le[i]>=w[q[r]]-Le[q[r]]) r--;
        q[++r]=i;
    }
    return res;
}

void work(int rt){
    Le.clear(), Lo.clear();
    findLo(rt);
    del=0;
    vector<int> w;
    for(auto u: Lo) getD(u, -1, del), w.pb(D[u]);
    rep(j,0,Lo.size()-2){
        int u=Lo[j];
        for(int i=h[u]; ~i; i=e[i].next){
            int go=e[i].to;
            if(go==fa[u]) Le.pb(e[i].w);
        }
    }

    del=max(del, cal(w, Le));
    res+=del;
}

signed main(){
    memset(h, -1, sizeof h);
    cin>>n;
    rep(i,1,n){
        int go, w; read(go), read(w);
        add(go, i, w), add(i, go, w);
    }

    rep(i,1,n) if(!dfn[i]) work(i);
    cout<<res<<endl;

    return 0;
}
```

例题 2：

上帝手中有 N 种世界元素，每种元素可以限制另外 1 种元素，把第 i 种世界元素能够限制的那种世界元素记为 $A[i]$。

为了世界的和平与安宁，上帝希望所有被投放的世界元素都有至少一个没有被投放的世界元素限制它。

上帝希望知道，在此前提下，他最多可以投放多少种世界元素？

```
const int N=1e6+5, M=N, INF=0x3f3f3f3f;

struct Edge{
    int to, next;
}e[M];

int h[N], tot;
```

```cpp
void add(int u, int v){
    e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

int n, fa[N];
bool vis[N];

int f[N][2];
int rt, fl;
void dp(int u){
    f[u][0]=f[u][1]=0;
    vis[u]=true;
    int del=INF;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==rt) continue;

        dp(go);
        f[u][0]+=max(f[go][0], f[go][1]);
        del=min(del, max(f[go][0], f[go][1])-f[go][0]);
    }
    f[u][1]=f[u][0]-del+1;
    if(u==fa[rt] && fl) f[u][1]=f[u][0]+1;
}

int main(){
    memset(h, -1, sizeof h);
    cin>>n;
    rep(i,1,n){
        read(fa[i]);
        add(fa[i], i);
    }

    int ans=0;
    rep(i,1,n) if(!vis[i]){
        rt=i;
        while(!vis[fa[rt]]) vis[rt]=true, rt=fa[rt];

        fl=0;
        dp(rt);
        int res=max(f[rt][0], f[rt][1]);

        fl=1;
        dp(rt);
        res=max(res, f[rt][0]);
        ans+=res;
    }
    cout<<ans<<endl;

    return 0;
}
```

仙人掌

例题：查询任意两点最短路。

```
const int N=2e4+5, M=1e5+5;

int n, m, Q, nwn;

struct Edge{
    int to, w, next;
}e[M];

int h1[N], h2[N], tot;

void add(int *h, int u, int v, int w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

int scir[N], s[N];
int fu[N], fw[N], fe[N];

void build_cir(int x, int y, int W){
    int S=W;
    for(int u=y; u!=x; u=fu[u]){
        s[u]=S;
        S+=fw[u];
    }
    s[x]=scir[x]=S;
    ++nwn;
    for(int u=y; u!=x; u=fu[u]){
        scir[u]=S;
        add(h2, nwn, u, min(S-s[u], s[u]));
    }
    add(h2, x, nwn, 0);
}

int dfn[N], low[N], ts;

void dfs(int u, int from){
    dfn[u]=low[u]=++ts;
    for(int i=h1[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!dfn[go]){
            fu[go]=u, fw[go]=e[i].w, fe[go]=i;
            dfs(go, i);
            low[u]=min(low[u], low[go]);
            if(dfn[u]<low[go]) add(h2, u, go, e[i].w);
        }
        else if(i!=(from^1)) low[u]=min(low[u], dfn[go]);
    }

    for(int i=h1[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(dfn[u]<dfn[go] && fe[go]!=i) build_cir(u, go, e[i].w);
    }
}

int fa[N][15], d[N], dep[N];
void get_lca(int u, int p){
```

```
    dep[u]=dep[p]+1, fa[u][0]=p;
    rep(i,1,14) fa[u][i]=fa[fa[u][i-1]][i-1];
    for(int i=h2[u]; ~i; i=e[i].next){
        int go=e[i].to;
        d[go]=d[u]+e[i].w;
        get_lca(go, u);
    }
}

int U, V;
int lca(int u, int v){
    if(dep[u]<dep[v]) swap(u, v);
    dwn(i,14,0) if(dep[fa[u][i]]>=dep[v]) u=fa[u][i];
    if(u==v) return u;
    dwn(i,14,0) if(fa[u][i]!=fa[v][i]) u=fa[u][i], v=fa[v][i];
    U=u, V=v;
    return fa[U][0];
}

int main(){
    memset(h1, -1, sizeof h1);
    memset(h2, -1, sizeof h2);

    cin>>n>>m>>Q;
    nwn=n;
    rep(i,1,m){
        int u, v, w; read(u), read(v), read(w);
        add(h1, u, v, w), add(h1, v, u, w);
    }

    dfs(1, -1);
    get_lca(1, 0);
    while(Q--){
        int u, v; read(u), read(v);
        int p=lca(u, v);
        if(p<=n){
            cout<<d[u]+d[v]-(d[p]<<1)<<endl;
        }
        else{
            int du=d[u]-d[U], dv=d[v]-d[V];
            int t=abs(s[U]-s[V]);
            cout<<(du+dv+min(t, scir[U]-t))<<endl;
        }
    }

    return 0;
}
```

网络流

最大流

dinic 算法

```
#include<bits/stdc++.h>
using namespace std;

#define gc() (st==ed&&(ed=(st=buf)+fread(buf,1,100000,stdin),st==ed)?EOF:*st++)
char buf[100001],*st=buf,*ed=buf;
void read(int &a){
    a=0;char c=gc();
    while(c>'9'||c<'0')c=gc();
    while(c>='0'&&c<='9')a=a*10+c-48,c=gc();
}

const int INF=0x3f3f3f3f;
const int N=10010, M=1e5+5;

struct node{
    int to, c, next;
}e[M<<1];
int h[N], tot;

void add(int u, int v, int cap){
    e[tot].to=v, e[tot].c=cap, e[tot].next=h[u], h[u]=tot++;
    e[tot].to=u, e[tot].c=0, e[tot].next=h[v], h[v]=tot++;
}

int n, m, S, T;

int d[N], q[N], cur[N];

bool bfs(){
    memset(d, -1, sizeof d);
    int tt=-1, hh=0;
    q[++tt]=S, d[S]=0, cur[S]=h[S];

    while(tt>=hh){
        int hd=q[hh++];
        for(int i=h[hd]; ~i; i=e[i].next){
            int go=e[i].to;
            if(d[go]==-1 && e[i].c){
                d[go]=d[hd]+1;
                cur[go]=h[go];
                if(go==T) return true;
                q[++tt]=go;
            }
        }
    }
    return false;
}

int find(int u, int limit){
    if(u==T) return limit;
    int flow=0;
    for(int i=cur[u]; ~i && flow<limit; i=e[i].next){
        cur[u]=i;
```

```
        int go=e[i].to;
        if(d[go]==d[u]+1 && e[i].c){
            int t=find(go, min(e[i].c, limit-flow));
            if(!t) d[go]=-1;
            e[i].c-=t, e[i^1].c+=t, flow+=t;
        }
    }
    return flow;
}

int dinic(){
    int res=0, flow;
    while(bfs()) while(flow=find(S, INF)) res+=flow;
    return res;
}

int main(){
    memset(h, -1, sizeof h);
    read(n), read(m), read(S), read(T);
    while(m--){
        int u, v, cap; read(u), read(v), read(cap);
        add(u, v, cap);
    }

    cout<<dinic()<<endl;

    return 0;
}
```

有源汇上下界最大流

```
int main(){
    memset(h, -1, sizeof h);
    cin>>n>>m>>s>>t;

    while(m--){
        int u, v, uc, lc; cin>>u>>v>>lc>>uc;
        add(u, v, uc-lc);
        imo[u]-=lc, imo[v]+=lc;
    }

    S=0, T=n+1;

    int cnt=0;
    for(int i=1; i<=n; i++)
        if(imo[i]>0) add(S, i, imo[i]), cnt+=imo[i];
        else if(imo[i]<0) add(i, T, -imo[i]);

    add(t, s, INF);

    int res=0;
    if(dinic()!=cnt) puts("No Solution");
    else{
        res+=e[tot-1].c;
        e[tot-1].c=0, e[tot-2].c=0;
        S=s, T=t;
        res+=dinic();
```

```
        cout<<res<<endl;
    }
    return 0;
}
```

最大权闭合图

```
int main(){
    memset(h, -1, sizeof h);
    cin>>n>>m;

    S=0, T=n+m+1;
    for(int i=1; i<=n; i++){
        int w; cin>>w;
        add(m+i, T, w);
    }

    int cnt=0;
    for(int i=1; i<=m; i++){
        int v1, v2, w; cin>>v1>>v2>>w;
        cnt+=w;
        add(i, m+v1, INF), add(i, m+v2, INF);
        add(S, i, w);
    }

    cout<<cnt-dinic()<<endl;

    return 0;
}
```

最大密度子图

例题：

已知公司中一共有 n 名员工，员工之间共有 m 对两两矛盾关系。

如果将一对有矛盾的员工安排在同一个团队，那么团队的管理难度就会增大。

一个团队的管理难度系数等于团队中的矛盾关系对数除以团队总人数。

团队的管理难度系数越大，团队就越难管理。

约翰希望给儿子安排的团队的管理难度系数尽可能大。

```
#include<bits/stdc++.h>
using namespace std;

const int N=110, M=1000+2*N<<1;
const double INF=1e10, eps=1e-8;

struct edge{
    int u, v;
}edges[M];
int n, m, S, T;
int deg[N];

struct node{
    int to, next;
```

```
    double c;
}e[M];
int h[N], tot;

void add(int u, int v, double c1, double c2){
    e[tot].to=v, e[tot].c=c1, e[tot].next=h[u], h[u]=tot++;
    e[tot].to=u, e[tot].c=c2, e[tot].next=h[v], h[v]=tot++;
}

void build(double g){
    memset(h, -1, sizeof h), tot=0;
    for(int i=1; i<=m; i++) add(edges[i].u, edges[i].v, 1, 1);
    for(int i=1; i<=n; i++) add(S, i, m, 0), add(i, T, m+2*g-deg[i], 0);
}

int q[N], d[N], cur[N];
bool bfs(){
    memset(d, -1, sizeof d);
    int tt=-1, hh=0;
    q[++tt]=S, d[S]=0, cur[S]=h[S];

    while(tt>=hh){
        int hd=q[hh++];
        for(int i=h[hd]; ~i; i=e[i].next){
            int go=e[i].to;
            if(d[go]==-1 && e[i].c>eps){
                d[go]=d[hd]+1;
                cur[go]=h[go];
                if(go==T) return true;
                q[++tt]=go;
            }
        }
    }
    return false;
}

double find(int u, double limit){
    if(u==T) return limit;
    double flow=0;
    for(int i=cur[u]; ~i && limit>flow; i=e[i].next){
        int go=e[i].to;
        cur[u]=i;
        if(d[go]==d[u]+1 && e[i].c>eps){
            double t=find(go, min(limit-flow, e[i].c));
            if(t<eps) d[go]=-1;
            e[i].c-=t, e[i^1].c+=t, flow+=t;
        }
    }
    return flow;
}

double dinic(double g){
    build(g);
    double res=0, flow;
    while(bfs()) while(flow=find(S, INF)) res+=flow;
    return res;
}
```

```
int res=0;
bool vis[N];
void dfs(int u){
    vis[u]=true;
    if(u!=S) res++;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(e[i].c>0 && !vis[go]) dfs(go);
    }
}

int main(){
    cin>>n>>m;

    S=0, T=n+1;
    for(int i=1; i<=m; i++){
        int u, v; cin>>u>>v;
        edges[i]={u, v};
        deg[u]++, deg[v]++;
    }

    double l=0, r=m;
    while(l+eps<r){
        double mid=(l+r)/2;
        if(m*n-dinic(mid)>eps) l=mid;
        else r=mid;
    }

    dinic(l);
    dfs(S);

    if(!res){
        puts("1\n1");
        return 0;
    }

    cout<<res<<endl;
    for(int i=1; i<=n; i++)
        if(vis[i]) cout<<i<<endl;

    return 0;
}
```

最小点权覆盖集

首先，爱丽丝绘制一个 N 个点 M 条边的有向图。

然后，鲍勃试图毁掉它。

在每一步操作中，鲍勃都可以选取一个点，并将所有射入该点的边移除或者将所有从该点射出的边移除。

已知，对于第 i 个点，将所有射入该点的边移除所需的花费为 W_i^+，将所有从该点射出的边移除所需的花费为 W_i^-。

鲍勃需要将图中的所有边移除，并且还要使花费尽可能少。

```cpp
bool vis[N];
void dfs(int u){
    vis[u]=true;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(e[i].c && !vis[go]) dfs(go);
    }
}

int main(){
    memset(h, -1, sizeof h);
    cin>>n>>m;

    for(int i=n+1; i<=2*n; i++) cin>>w[i];
    for(int i=1; i<=n; i++) cin>>w[i];

    S=0, T=2*n+1;
    while(m--){
        int u, v; cin>>u>>v;
        add(u, v+n, INF);
    }

    for(int i=1; i<=n; i++) add(S, i, w[i]);
    for(int i=n+1; i<=2*n; i++) add(i, T, w[i]);

    cout<<dinic()<<endl;
    dfs(S);

    int cnt=0;
    for(int i=0; i<tot; i+=2){
        int u=e[i^1].to, v=e[i].to;
        if(vis[u] && !vis[v]) cnt++;
    }

    cout<<cnt<<endl;
    for(int i=0; i<tot; i+=2){
        int u=e[i^1].to, v=e[i].to;
        if(vis[u] && !vis[v] && u==S) cout<<v<<' '<<'-'<<endl;
        else if(vis[u] && !vis[v] && v==T) cout<<u-n<<' '<<'+'<<endl;
    }
    return 0;
}
```

最大点权独立集

给出一个 $n\times m$ 网格，每个格子上有一个价值。

Amber 可以自己决定起点，开始时刻为第 0 秒。

以下操作，在每秒内按顺序执行。

1. 若第 i 秒开始时，Amber 在 (x,y)，则 Amber 可以拿走 (x,y) 上的宝石。
2. 在偶数秒时（i 为偶数），则 Amber 周围 4 格的宝石将会消失。
3. 每一秒 Amber 可以移动到相邻的格子或原地不动。

求 Amber 最多能得到多大总价值的宝石。

```
#include<bits/stdc++.h>
using namespace std;

int id[105][105], cnt;

int main(){
    memset(h, -1, sizeof h);
    cin>>n>>m;

    for(int i=1; i<=n; i++) for(int j=1; j<=m; j++) id[i][j]=++cnt;

    S=0, T=cnt+1;
    int all=0;
    int dx[]={1, 0, -1, 0}, dy[]={0, 1, 0, -1};
    for(int i=1; i<=n; i++) for(int j=1; j<=m; j++){
        int w; cin>>w; all+=w;
        if(i+j&1){
            add(S, id[i][j], w);
            for(int k=0; k<4; k++){
                int kx=i+dx[k], ky=j+dy[k];
                if(kx<1 || kx>n || ky<1 || ky>m) continue;
                add(id[i][j], id[kx][ky], INF);
            }
        }else add(id[i][j], T, w);
    }

    cout<<all-dinic()<<endl;

    return 0;
}
```

费用流

不存在负圈

```
#include<bits/stdc++.h>
using namespace std;

inline int read()
{
    int x=0,y=1;char c=getchar();
    while (c<'0'||c>'9') {if (c=='-') y=-1;c=getchar();}
    while (c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
    return x*y;
}

const int N=205, M=10005<<1, INF=0x3f3f3f3f;

int n, m, S, T;
struct node{
    int to, c, rc, w, next; // c 表示容量（进行 capacity scaling ），rc 表示原图容量，
w 表示费用。
```

```
}e[M];

int h[N], tot;

void add(int u, int v, int rc, int w){
    e[tot].to=v, e[tot].rc=rc, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
    e[tot].to=u, e[tot].rc=0, e[tot].w=-w, e[tot].next=h[v], h[v]=tot++;
}

int d[N], pre[N];
bool vis[N];
int q[N];

void spfa(int s){
    memset(vis, false, sizeof vis);
    memset(d, 0x3f, sizeof d);
    memset(pre, -1, sizeof pre);
    int tt=0, hh=0;
    d[s]=0, vis[s]=true, q[tt++]=s;

    while(tt!=hh){
        int hd=q[hh++]; if(hh==N) hh=0;
        vis[hd]=false;

        for(int i=h[hd]; ~i; i=e[i].next){
            int go=e[i].to;
            if(e[i].c && d[go]>d[hd]+e[i].w){
                d[go]=d[hd]+e[i].w;
                pre[go]=i;
                if(!vis[go]){
                    vis[go]=true;
                    q[tt++]=go; if(tt==N) tt=0;
                }
            }
        }
    }
}

void add_one_cap(int id){
    // 优化，有流量的话，不可能关于这条边还存在负圈，直接更新后 return。
    if(e[id].c){
        e[id].c++;
        return;
    }
    int u=e[id^1].to, v=e[id].to; // from and to
    spfa(v);
    if(d[u]<INF && d[u]+e[id].w<0){
        e[id^1].c++;
        int x=u;
        while(x!=v){
            int t=pre[x];
            e[t].c--, e[t^1].c++;
            x=e[t^1].to;
        }
    }else e[id].c++;
}

int main(){
```

```
    memset(h, -1, sizeof h);
    n=read(), m=read(), S=read(), T=read();

    while(m--){
        int u, v, rc, w; u=read(), v=read(), rc=read(), w=read();
        add(u, v, rc, w);
    }

    add(T, S, INF, -INF);

    for(int i=32; i>=0; i--){ // 取决于 logU 的大小
        for(int j=0; j<tot; j++) e[j].c<<=1;
        for(int j=0; j<tot; j+=2) if(e[j].rc>>i&1) add_one_cap(j); // 传入边的编号
id，进行 +1 容量操作。
    }

    int cost=0;
    for(int i=0; i<tot-2; i+=2) cost+=e[i].w*e[i^1].c;
    cout<<e[tot-1].c<<' '<<cost<<endl;

    return 0;
}
```

存在负圈

```
#include<bits/stdc++.h>
using namespace std;

inline int read()
{
    int x=0,y=1;char c=getchar();
    while (c<'0'||c>'9') {if (c=='-') y=-1;c=getchar();}
    while (c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
    return x*y;
}

const int N=205, M=10005<<1, INF=0x3f3f3f3f;

int n, m, S, T;
struct node{
    int to, c, rc, w, next; // c 表示容量（进行 capacity scaling ），rc 表示原图容量，
w 表示费用。
}e[M];

int h[N], tot;

void add(int u, int v, int rc, int w){
    e[tot].to=v, e[tot].rc=rc, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
    e[tot].to=u, e[tot].rc=0, e[tot].w=-w, e[tot].next=h[v], h[v]=tot++;
}

int d[N], pre[N];
bool vis[N];
int q[N];

void spfa(int s){
    memset(vis, false, sizeof vis);
```

```
    memset(d, 0x3f, sizeof d);
    memset(pre, -1, sizeof pre);
    int tt=0, hh=0;
    d[s]=0, vis[s]=true, q[tt++]=s;

    while(tt!=hh){
        int hd=q[hh++]; if(hh==N) hh=0;
        vis[hd]=false;

        for(int i=h[hd]; ~i; i=e[i].next){
            int go=e[i].to;
            if(e[i].c && d[go]>d[hd]+e[i].w){
                d[go]=d[hd]+e[i].w;
                pre[go]=i;
                if(!vis[go]){
                    vis[go]=true;
                    q[tt++]=go; if(tt==N) tt=0;
                }
            }
        }
    }
}

void add_one_cap(int id){
    // 优化，有流量的话，不可能关于这条边还存在负圈，直接更新后 return。
    if(e[id].c){
        e[id].c++;
        return;
    }
    int u=e[id^1].to, v=e[id].to; // from and to
    spfa(v);
    if(d[u]<INF && d[u]+e[id].w<0){
        e[id^1].c++;
        int x=u;
        while(x!=v){
            int t=pre[x];
            e[t].c--, e[t^1].c++;
            x=e[t^1].to;
        }
    }else e[id].c++;
}

int main(){
    memset(h, -1, sizeof h);
    n=read(), m=read(), S=read(), T=read();

    while(m--){
        int u, v, rc, w; u=read(), v=read(), rc=read(), w=read();
        add(u, v, rc, w);
    }

    add(T, S, INF, -INF);

    for(int i=10; i>=0; i--){
        for(int j=0; j<tot; j++) e[j].c<<=1;
        for(int j=0; j<tot; j+=2) if(e[j].rc>>i&1) add_one_cap(j); // 传入边的编号
id，进行 +1 容量操作。
    }
```

```
    int cost=0;
    for(int i=0; i<tot-2; i+=2) cost+=e[i].w*e[i^1].c;
    cout<<e[tot-1].c<<' '<<cost<<endl;

    return 0;
}
```