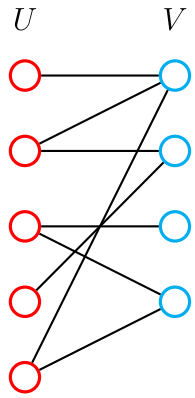
二分图

定义

二分图，又称二部图，英文名叫 Bipartite graph。

二分图是什么？节点由两个集合组成，且两个集合内部没有边的图。

换言之，存在一种方案，将节点划分成满足以上性质的两个集合。



性质

1. 如果两个集合中的点分别染成黑色和白色，可以发现二分图中的每一条边都一定是连接一个黑色点和一个白色点。
2. 二分图不存在长度为奇数的环
   因为每一条边都是从一个集合走到另一个集合，只有走偶数次才可能回到同一个集合。

二分图判定

染色法 $O(m+n)$

题目链接

首先随意选取一个未染色的点进行染色，然后尝试将其相邻的点染成相反的颜色。
如果邻接点已经被染色并且现有的染色与它应该被染的颜色不同，那么就说明不是二分图。
而如果全部顺利染色完毕，则说明是二分图。
染色结束后的情况（记录在数组中）便将图中的所有节点分为了两个部分，即为二分图的两个点集。

```
bool dfs(int x,int c) {
   color[x]=c;
    for(int i=head[x]; i; i=e[i].next) {
        int to=e[i].to;
        if(!color[to]) {
            if(!dfs(to,3-c))
                return 0;
        } else if(color[to]==c)
            return 0;
    }
    return 1;
}
for(int i=1; i<=n; i++)
    if(!color[i]) if(!dfs(i,1)) flag=0;break;}
```

应用

常见问题

1. **二分图最大匹配**
2. 二分图带权匹配
3. 二分图最小覆盖点
4. 二分图最大独立集
5. 有向无环图的最小路径覆盖

二分图最大匹配

题目链接

一些概念

> 二分图的匹配：
> 给定一个二分图 G，在 G 的一个子图 M 中，M 的边集 {E} 中的任意两条边都不依附于同一个顶点，则称 M 是一个匹配。
> 换句话说：任意两条边没有公共端点的边的集合被称为为图的一组匹配
>
> 二分图的最大匹配：
> 所有匹配中包含边数最多的一组匹配被称为二分图的最大匹配，其边数即为最大匹配数。

> 完美匹配：
> 如果一个图的某个匹配中，所有的顶点都是匹配点，那么它就是一个完美匹配。
>
> 交替路：
> 从一个未匹配点出发，依次经过非匹配边、匹配边、非匹配边…形成的路径叫交替路。
>
> 增广路：
> 从一个未匹配点出发，走交替路，如果途径另一个未匹配点（出发的点不算），则这条交替 路称为增广路（agumenting path）。

每次找到一条增广路，把path上的所有边的状态取反（匹配边变成非匹配边，非匹配边变成匹配边），那么得到的新的边集S'还是一组匹配，并且匹配边数增加了1.所以可以得到推论：二分图的一组匹配S是最大匹配，当且仅当图中不存在S的增广路

匈牙利算法

1. 设$S=\varnothing$
2. 寻找增广路path，把路径上所有边的匹配状态取反，得到一个更大的匹配。
3. 重复2，直至图中不存在增广路

复杂度：

1. 时间复杂度：邻接矩阵最坏为$O(n^3)$
   邻接表：$O(mn)$
2. 空间复杂度：邻接矩阵：$O(n^2)$
   邻接表：$O(n+m)$

匈牙利算法基于贪心思想，重要特点是：一个节点成为匹配点之后，最多只会因为找到增广路而更换匹配对象，但是绝不会再变回匹配点。

详细说明链接

```
int dfs(int x) {
```

```
    for(int i=head[x]; i ; i=e[i].next) {
        int to=e[i].to;
        if(!vis[to]) { //如果在这一轮模拟匹配中,这个点尚未被预定
            vis[to]=1;
            //如果to没有匹配，或者它原来的匹配点能够匹配其它的点。配对成功
            if(!match[to]||dfs(match[to])) {
                match[to]=x;
                return 1;
            }
        }
    }
    return 0;
}
for(int i=1; i<=n1 ; i++) {
    memset(vis,0,sizeof vis);
    if(dfs(i))ans++;
}
```

二分图其他常见问题

可转化为求最大匹配问题

1. **二分图带权匹配**
   二分图每条边都有一个权值，求该二分图的一组最大匹配，并且匹配边的权值和最大。
   解法：KM算法（KM必需在满足"带权最大匹配一定是完备匹配"的二分图中求解）或者转换成网络流模型。如果使用 Dinic 算法 求该网络的最大流，可在$O(\sqrt{n}m)$求出。
2. **二分图最小覆盖点**
   给定一张二分图，求出一个最小的点集S,使得图中任意一条边都有至少一个端点属于S。
   即：以最少点覆盖所有的边
   *Konig*定理:二分图最小点覆盖包含的点数等于二分图最大匹配包含的边数。
   例题：泥泞的区域
3. **二分图最大独立集**
   选最多的点，满足两两之间没有边相连。
   结论：n个节点的二分图，最大独立集的大小等于n − 最大匹配数
   例题：骑士放置
4. **有向无环图的最小路径覆盖**
   给定一张有向无环图，要求用尽量少的不相交简单路劲，覆盖有向无环图的所有顶点。
   需要建立拆点二分图，然后求解最大匹配。
   结论：最小路径点覆盖（最小路径覆盖） = 总点数 - 最大匹配
   例题：舞动的夜晚

网络流

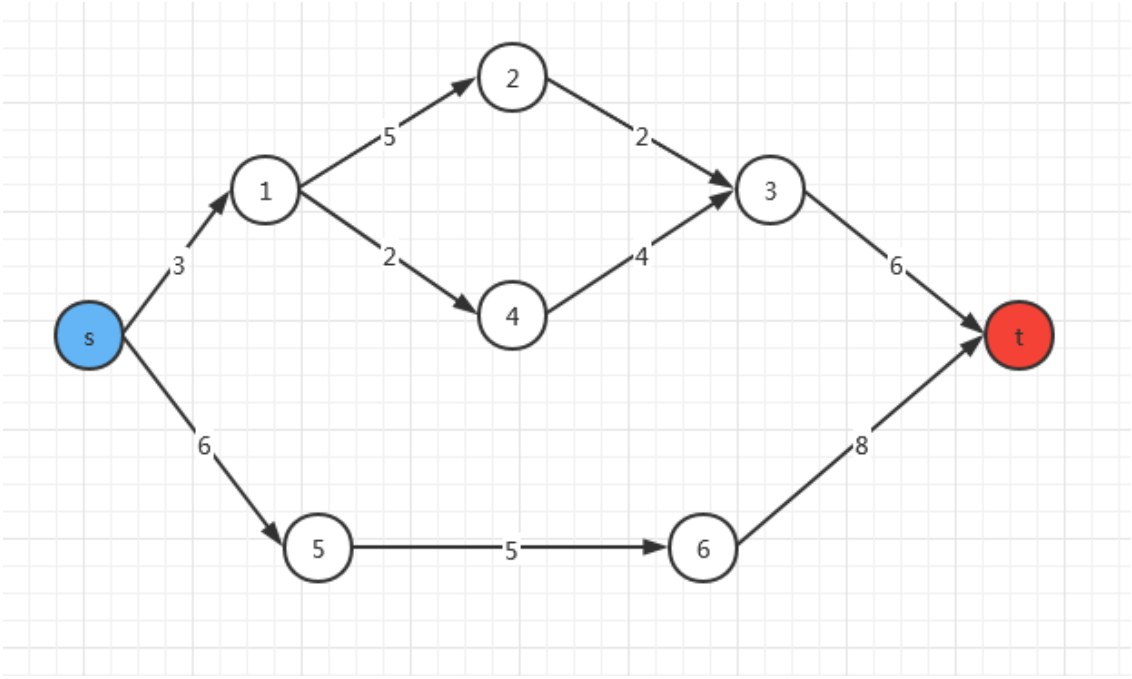
资料参考1
资料参考2

概念介绍

1. 定义
   对于任意一张**有向图**（也就是**网络**），其中有N个点、M条边以及源点S和汇点T
   然后我们把$c(x,y)$称为边的**容量**

2. 转化
   为了通俗易懂，我们来结合生活实际理解上面网络的定义：

   将有向图理解为我们城市的水网，有N户家庭、M条管道以及供水点S和汇合点T
   是不是好理解一点？现在给出一张网络



3. 流函数
   和上面的c差不多，我们把$f(x,y)$称为边的流量，则f称为网络的流函数，它满足三个条件：
   $s(x,y) \leq c(x,y)$
   $f(x,y) = -f(y,x)$
   $\forall x \neq S,\ x \neq T, \sum_{(u,x)\in E} f(u,x) = \sum_{(x,v)\in E} f(x,v)$
   这三个条件其实也是流函数的三大性质：

   1. 容量限制：每条边的流量总不可能大于该边的容量的（不然水管就爆了）
   2. 斜对称：正向边的流量=反向边的流量（反向边后面会具体讲）
   3. 流量守恒：正向的所有流量和=反向的所有流量和（就是总量始终不变）

4. 残量网络
   在任意时刻，网络中所有节点以及剩余容量大于00的边构成的子图被称为残量网络

网络流模型可以形象地描述为：在不超过容量限制的前提下，“流”从源点源源不断地产生，流经整个网络，最终全部归于汇点。

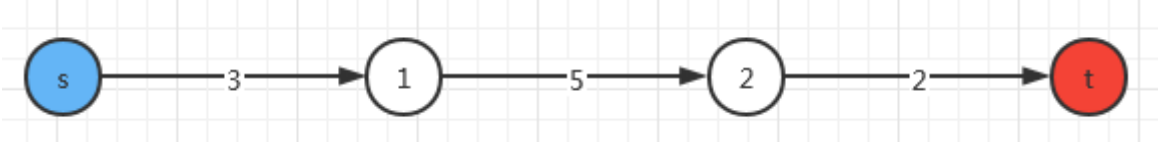
最大流

题目链接

最大流问题(maximum flow problem)，一种组合最优化问题，就是要讨论如何充分利用装置的能力，使得运输的流量最大，以取得最好的效果。

一种通俗的理解就是：把源点看成是自来水场，汇点看成你家，边就是水管，流量就是水管最多能流多少单位的水。自来水厂源源不断的放水，问你家最多能收到几个单位的水。

首先，我们从一条路径来考虑（如下图）：



显然，最大流为2。我们发现，一条路径的流量是由这条路径的最小值决定的。

Edmonds-Karps增广路算法

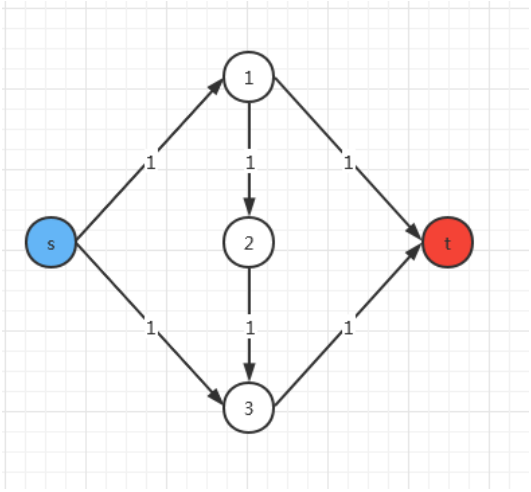
时间复杂度$O(nm^2)$。然而实际运用中则远远达不到这个上界，效率较高，一般能处理10^3-10^4规模的网络

首先，引入增广路的概念（于二分图的增广路不同）：一条从s到t的路径，水流流过这条路，使得当前可以到达t的流量可以增加。

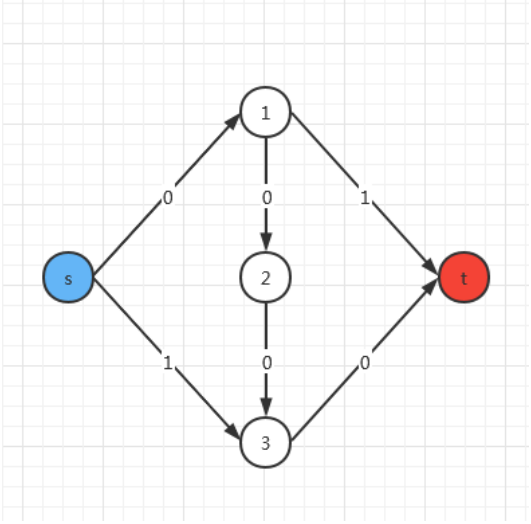
知道了增广路的概念，就可以很显然的想出一种做法，**不断寻找增广路并处理和累加答案，直到找不到增广路，答案就是最大流。**

那如何寻找增广路呢？从s开始bfs，条件是边权不为0（不为0才能增加流量），当搜到t时，就找到了一条增广路。然后，将答案加上这条增广路的流量的最小值，将这条增广路上所有边的流量减掉最小值（因为已经使用了），直到找不到增广路。

这个做法看上去很对，但是他有缺陷，看下图：

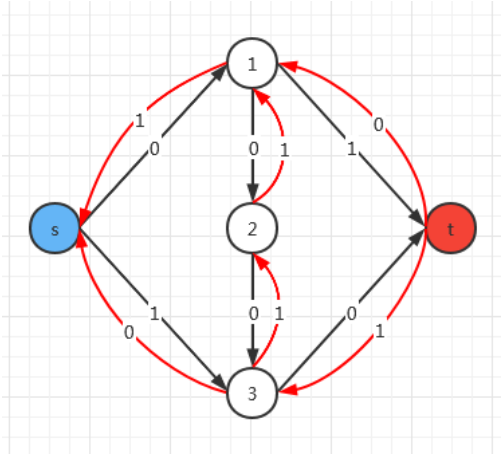


显然，有两条增广路，最大流为2。但是，如果找到了这条增广路：$s\rightarrow 1\rightarrow 2\rightarrow 3\rightarrow t$，这幅图就会变成这样子：

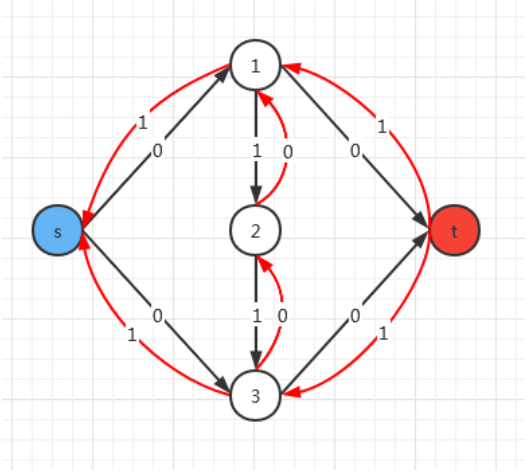


最大流就变成了1,对于这个问题，可以通过建立反向边来解决。在建图时加入反向边，流量为0，在流

量减去最小值的时候，将反向边加上最小值。那么，上面那幅图就变成了这样：



于是，就可以找到另一条增广路 s→3→2→1→t ，图变成了：



发现成功求出了正确的解。可见，反向边的用处就是标记处理过的边，在有更好情况下把原来的操作给撤销。相当于一个反悔操作。

小细节：用邻接表存图时，要使第一条边编号为2，且反向边一定要在正向边建完以后就建，这样可以很方便的通过异或1来得到反向边。

代码：

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#define ll long long
using namespace std;

const int N=2510,M=520010;
int n,m,s,t,u,v;
long long w,ans,dis[M];
int tot=1,vis[M],pre[M],head[M],flag[N][N];

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

void add(int u,int v,ll w) {
    e[++tot].to=v;
```

```
    e[tot].w=w;
    e[tot].next=head[u];
    head[u]=tot;
}

int bfs() {  //bfs寻找增广路
    for(int i=1; i<=n; i++) vis[i]=0;
    queue<int> q;
    q.push(s);
    vis[s]=1;
    dis[s]=1e9+7;
    while(!q.empty()) {
        int x=q.front();
        q.pop();
        for(int i=head[x]; i; i=e[i].next) {
            int to=e[i].to;
            if(e[i].w&&!vis[to]) {
                dis[to]=min(dis[x],e[i].w);
                pre[to]=i;  //记录前驱，方便修改边权
                q.push(to);
                vis[to]=1;
                if(to==t) return 1;  //找到了一条增广路
            }
        }
    }
    return 0;
}

void update() {  //更新所经过边的正向边权以及反向边权
    int x=t;
    while(x!=s) {
        int v=pre[x];
        e[v].w-=dis[t];
        e[v^1].w+=dis[t];
        x=e[v^1].to;
    }
    ans+=dis[t];   //累加每一条增广路经的最小流量值
}

int main() {
    int x,y;
    ll z;
    cin>>n>>m>>s>>t;
    for(int i=1; i<=m; i++) {
        scanf("%d%d%lld",&x,&y,&z);
        if(flag[x][y]==0) {  //处理重边的操作（加上这个模板题就可以用Ek算法过了）
            add(x,y,z);
            add(y,x,0);
            flag[x][y]=tot;
        } else {
            e[flag[x][y]-1].w+=z;
        }
    }
    while(bfs()!=0) {  //直到网络中不存在增广路
        update();
    }
    printf("%lld",ans);
    return 0;
```

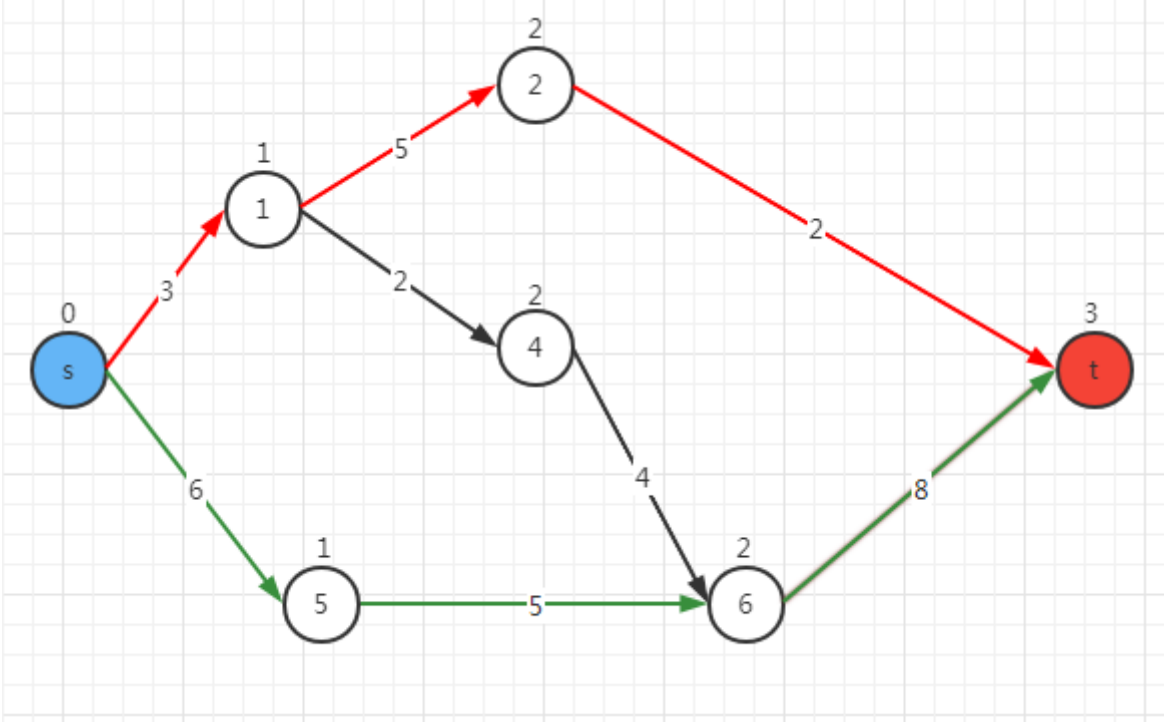
```
}
```

Dinic算法

时间复杂度为$O(n^2m)$。实际运用中远远达不到这个上界，可以说是比较容易实现的效率最高的网络流算法之一，一般能够处理10^4-10^5规模的网络。

特别的$Dinic$算法求解二分图最大匹配的时间复杂度为$O(m\sqrt{n})$,实际表现则更快。

在EK算法中，我们发现每次dfs只能找到一条最短路。那能不能一次dfs就找到多条增广路呢？答案是可以。首先，用bfs将图分层。为什么要分层呢？因为这样可以保持找到的多条增广路是最短的。如下图，一次找出了两条增广路。



找到增广路后，用dfs遍历多条增广路更新答案，dfs看上去慢，其实它能一次性处理多条增广路，增加了效率。

$Dinic$算法不断重复一下步骤，指导残量网络中s不能到达t

1. 在残量网络上BFS求出节点的层次，构造分层图
2. 在分层图上DFS寻找增广路，在回溯时实时更新剩余容量。另外，每个点可以流向多条出边，同时还加入了若干剪枝。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#define ll long long
using namespace std;

const ll inf=1e9+7;
const int N=510,M=100010;
int n,m,s,t;
ll w,ans,dis[M];
int tot=1,now[M],head[M],cnt;
```

```
struct node {
    int to,next;
    ll w;
} e[M];

void add(int u,int v,ll w) {
    e[++tot].to=v;
    e[tot].w=w;
    e[tot].next=head[u];
    head[u]=tot;
}
int bfs() {  //在残量网络中构造分层图
    for(int i=1; i<=n; i++)
        dis[i]=inf;
    queue<int> q;
    q.push(s);
    dis[s]=0;
    now[s]=head[s];
    while(!q.empty()) {
        int x=q.front();
        q.pop();
        for(int i=head[x]; i; i=e[i].next) {
            int v=e[i].to;
            if(e[i].w>0&&dis[v]==inf) {
                q.push(v);
                now[v]=head[v];
                dis[v]=dis[x]+1;
                if(v==t) return 1;
            }
        }
    }
    return 0;
}

int dfs(int x,ll sum) {  //sum是整条增广路对最大流的贡献
    if(x==t) return sum;
    ll k,res=0;  //k是当前最小的剩余容量
    for(int i=now[x]; i&&sum; i=e[i].next) {
        now[x]=i;  //当前弧优化
        int v=e[i].to;
        if(e[i].w>0&&(dis[v]==dis[x]+1)) {
            k=dfs(v,min(sum,e[i].w));
            if(k==0) dis[v]=inf;  //剪枝，去掉增广完毕的点
            e[i].w-=k;
            e[i^1].w+=k;
            res+=k;  //res表示经过该点的所有流量和（相当于流出的总量）
            sum-=k;  //sum表示经过该点的剩余流量
        }
    }
    return res;
}

int main() {
    int x,y;
    ll z;
    cin>>n>>m>>s>>t;
    for(int i=1; i<=m; i++) {
        scanf("%d%d%lld",&x,&y,&z);
```

```
        add(x,y,z);
        add(y,x,0);
    }
    while(bfs()) {
        ans+=dfs(s,inf);  //流量守恒（流入=流出）
    }
    printf("%lld",ans);
    return 0;
}
```

费用流

题目链接

指在水流过水管时，每单位水需要交纳水费（可能为负数，就是水厂要付你钱），求最大流和在流量最大的情况下最小的费用。

显然，还是分层，遍历。那么要修改哪一步呢?

遍历是不用修改的，所以需要修改分层。那么要将分层的bfs修改成什么呢？想到费用，我们第一个想到的就是最短路。

所以，只要将分层的bfs改为已经死的$SPFA$（因为$Dijkstra$不能处理负权），就可以了。

例题：Kaka's Matrix Travels

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#define ll long long
using namespace std;

const int N=5001,M=50001;
int n, m, s, t, cnt=0;
int maxflow, mincost;
int dis[N], head[N], incf[N], pre[N];//dis表示最短路，incf表示当前增广路上最小流量，pre表示前驱
bool vis[N];
struct node {
    int next,to,w,flow;
} e[M*2];
void add(int u, int v, int flow, int w) {
    e[++cnt].next = head[u];
    e[cnt].to = v;
    e[cnt].w = w;
    e[cnt].flow = flow;
    head[u] = cnt;
}
bool spfa() {//关于SPFA，他诈尸了
    queue <int> q;
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    q.push(s);
```

```
    dis[s] = 0;
    vis[s] = 1;
    incf[s] = 1 << 30;
    while(!q.empty()) {
        int x = q.front();
        vis[x] = 0;
        q.pop();
        for(int i = head[x]; i; i = e[i].next) {
            if(!e[i].flow) continue;//没有剩余流量
            int to = e[i].to;
            if(dis[to] > dis[x] + e[i].w) {
                dis[to] = dis[x] + e[i].w;
                incf[to] = min(incf[x], e[i].flow);//更新incf
                pre[to] = i;
                if(!vis[to]) vis[to] = 1, q.push(to);
            }
        }
    }
    if(dis[t] == 1061109567) return 0;
    return 1;
}
void MCMF() {
    while(spfa()) {//如果有增广路
        int x = t;
        maxflow += incf[t];
        mincost += dis[t] * incf[t];
        int i;
        while(x != s) {//遍历这条增广路，正向边减流反向边加流
            i = pre[x];
            e[i].flow -= incf[t];
            e[i^1].flow += incf[t];
            x = e[i^1].to;
        }
    }
}
int main() {
    cin>>n>>m>>s>>t;
    for(int i = 1; i <= m; ++i) {
        int u,v,w,x;
        scanf("%d%d%d%d",&u,&v,&w,&x);
        add(u,v,w,x);
        add(v,u,0,-x);//反向边费用为-f[i]
    }
    MCMF();//最小费用最大流
    printf("%d %d\n",maxflow,mincost);
    return 0;
}
```