网络流 _4 上下界可行流

对于容量有上下界规范的网络流问题

无源汇上下界可行流

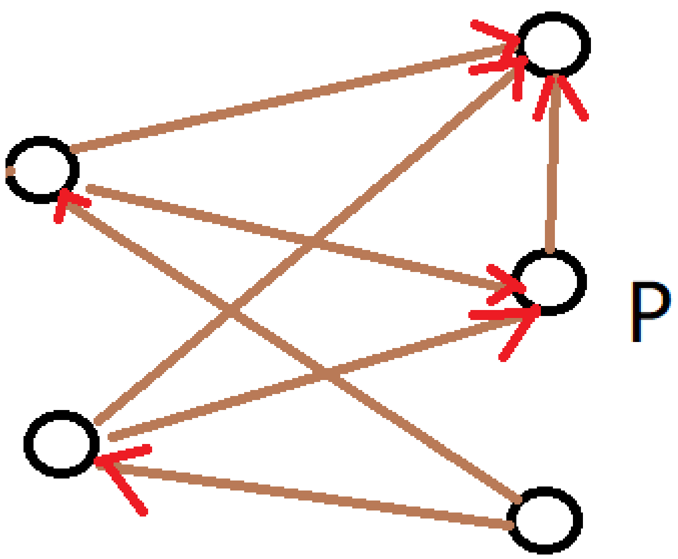
> n个点m条边的有向图，每条边有一个流量下界和流量上界规范，求是否存在一个可行流

设原网络为`(G,F)`,变换后网络为`(G',S')`,每条边上界`cl(u,v)`， 下界`cu(u,v)`,流量`f(u,v)`.对于上下界，我们很容易将等式变形：

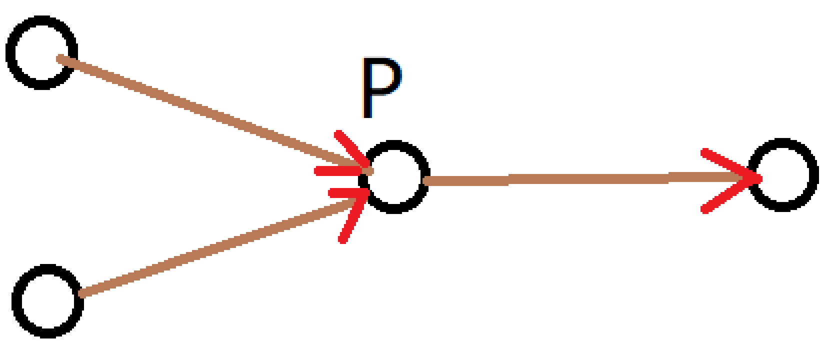
$$c_l(u,v) \le f(u,v) \le c_u(u,v) \to 0 \le f(u,v) - c_l(u,v) \le c_u(u,v) - c_l(u,v)$$

考察变形之后的网络，满足容量限制，但是并不满足流量守恒，原因：

对于一个有向图G



考察图上P点：



$$\text{变换前}: f(v1,p) + f(v2,p) = f(p,v3)$$

$$\text{变换后}: (f(v1,p) - c) + (f(v2,p) - c) \neq f(p,v3) - c$$

因此我们需要进行构造来使得变换后的仍为一个流网络

- 构造方法：（此处略与y总不同，但是更好理解）

> `C入`:进入P点，因变换而减少的流量（相当于进来的补给变少了）；`C出`:从P点出，因变换而增加的流量 （相当于出去的消耗变少了）；`C`：`C = C出 - C入`

对于每一个点，若其`C > 0`则说明补给减少的量小于消耗减少的量（虽然给我少了，但是我用的更少），说明该点有余量，因此从此点向汇点T连接一条容量为`C`的边来释放多余的量

若`C < 0`则说明消耗减少的量小于补给减少的量（流出了很多但是流进来很少），说明该点缺少流量，因此需要从源点S流入`C`的流量才能满足流出的流量。因此从源点S向此点连接一条容量为`C`的边来补充缺少的量

> 注意：此时可以发现对于任意P点,若S与之相连，那条边是满流的，之若与T相连，那条边也是满流（因为构造就是这么构造的），因此**形成了一个从源点到汇点的最大流**

AC代码

难在建图，建图建好了直接上板子就行

```
const int N = 510 , M = 2*(11000+N) , INF = 1e8;
int n,m,S,T;
int e[M],ne[M],h[N],f[M],l[M],idx=0;
int cur[M],q[M],d[M],A[M];

void add(int a,int b,int c,int d){
    e[idx] = b , f[idx] = d-c , l[idx] = c , ne[idx] = h[a] , h[a] = idx ++ ;
    e[idx] = a , f[idx] = 0 , ne[idx] = h[b] , h[b] = idx ++ ;
}

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

    while(hh <= tt){
        int u = q[hh ++];
        for(int i = h[u]; ~i ; i = ne[i]){
            int ver = e[i];
            if(d[ver] == -1 && f[i]){
                d[ver] = d[u] + 1;
                cur[ver] = h[ver];
                if(ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    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=ne[i]){
        cur[u] = i;
        int ver = e[i];
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit-flow));
            if(t == 0) d[ver] = -1;
            f[i] -= t , f[i^1] += t , flow += t;
```

```
        }
    }
    return flow;
}

int dinic(){
    int ans = 0 , flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
    return ans;
}
//=================================
int main(){
    memset(h,-1,sizeof h);

    n = read() , m = read();
    S = 0 , T = n + 1;

    int tot = 0;
    rep(i,1,m){
        int a = read() , b = read() , c = read() , d = read();
        add(a,b,c,d);
        // tot += d - c; 最大流流量应该与中间怎么流的没有关系
        A[a] += c , A[b] -= c;  //每有一条出边变换之后会少消耗c（相当于+c），每有一条入边
会少补给c（相当于-c）
    }
    rep(i,1,n)
        if(A[i] > 0) add(i,T,0,A[i]) ;//还有余量，因此流向汇点
        else if(A[i] < 0 ) add(S,i,0,-A[i]),tot-=A[i]; //缺少补给，因此从源点流入
    /*y总写法，一样的道理，反过来考虑
    rep(i,1,m){
        int a = read() , b = read() , c = read() , d = read();
        add(a,b,c,d);
        A[a] -= c , A[b] += c;   //每有一条入边变换之后就能多消耗c,每有一条出边变换之后就
能少消耗c
    }
    rep(i,1,n)
        if(A[i] > 0 ) add(S,i,0,A[i]), tot += A[i];
        else if(A[i] < 0) add(i,T,0,-A[i]);
    */
    if(dinic() != tot) puts("NO");
    else{
        puts("YES");
        for(int i=0;i<2*m;i+=2)
            print(f[i^1] + l[i]);
    }
    return 0;
}
```

有源汇上下界最大流

> n个点m条边的有向图，每一条边都有一个流量下界和流量上界，求从源点S到汇点T的最大流

- 算法:

> 1.记原来的源点为`s`，汇点为`t`,同无源汇的方式，建立虚拟源点S和汇点T，通过式子变形将其转化为一个新图`G'`
>
> $$变换前: f(v1,p)+f(v2,p)=f(p,v3)$$
>
> $$变换后: (f(v1,p)-c)+(f(v2,p)-c)\neq f(p,v3)-c$$
>
> 由于建立新图之后，此时原来的源点`s`和汇点`t`会变成中间结点，因此为了使之流量守恒，我们需要从`t`向`s`连一条容量为正无穷的边。
>
> 2.从S向T跑一遍`Dinic`算法，如果从S出的流量是满流，说明新图存在可行流，即此上下行约束下存在可行流
>
> 3.若存在可行流，再原来的源点s到原来的汇点t的跑一遍`dinic`算法（在残留网络上不断的搞掉增广路径），此时得到的便是s到t的最大流

- 证明:
  - 证明思路：证明原图中的上下界可行流于新图中的可行流存在一一映射
- > 令
  >
  > $$(G原图, f_0可行上下界可行流) \rightarrow (G^{'}新图中的可行流, f_0^{'}新图中的可行流(从S出发的满流))$$
  >
  > $$G^{'}_{f_0^{'}}为f_0^{'}的残留网络, f^{'}_{0s\rightarrow t}为f_0^{'}中的s\rightarrow t的可行流$$
  >
  > 证:
  >
  > $$|f^{'}_{0s\rightarrow t}+f_0^{'}|仍然是新图上的可行流$$
  >
  > 流量守恒：由S出去的都是满流，进入T的也都是满流，因此对于s -> t的任意可行流都不经过S和T
  >
  > 容量限制：由于t向s连接了一条容量为无穷的边，因此每一条边都满足容量限制
  >
  > $$f^{'}对应一条s\rightarrow t的可行流$$
  >
  > `f'-f0'`可以构造出所有流量都在中间，因为S、T对于`f'`和`f0'`而言都是满流 。
  >
  > 综上,原图中的上下界可行流于新图中的可行流存在一一映射,因此算法X具有正确性

```
const int INF = 1e8,N=510,M=2*(N+10010);
int n,m,S,T;
int e[M],ne[M],h[N],f[M],idx=0;
int q[M],cur[M],d[M],A[M];

void add(int a,int b,int 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 ++;
}

int find(int u,int limit){
    if(u == T) return limit;
    int flow = 0;

    for(int i=cur[u];~i && flow < limit;i=ne[i]){
        cur[u] = i;

        int ver = e[i];
```

```
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit-flow));
            if(!t) d[ver] = -1;
            f[i] -= t , f[i^1] += t , flow += t;
        }
    }
    return flow;
}

bool bfs(){
    memset(d,-1,sizeof d);

    int hh = 0 , tt = 0;
    q[0] = S , cur[S] = h[S] , d[S] = 0;

    while(hh <= tt){
        int u = q[hh ++];
        for(int i=h[u];~i;i=ne[i]){
            int ver = e[i];
            if(d[ver] == -1 && f[i]){
                d[ver] = d[u] + 1;
                cur[ver] = h[ver];
                if(ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    return false;
}

int dinic(){
    int ans = 0 , flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
    return ans;
}
//=================================
int main(){
    memset(h,-1,sizeof h);

    int s,t,ans=0;
    n = read() , m = read() , s = read() , t = read();

    S = 0 , T = n + 1;

    rep(i,1,m){
        int a = read(), b = read() , c = read() , d = read();
        add(a,b,d-c);
        A[a] += c ,A[b] -= c;
    }
    int tot = 0;
    rep(i,1,n){
        if(A[i] > 0) add(i,T,A[i]);
        else if(A[i] < 0 ) add(S,i,-A[i]),tot-=A[i];
    }
    add(t,s,INF);
    if(dinic() < tot) puts("No Solution");
    else{
```

```
        ans = f[idx - 1];  //f[idx-2]为容量，因此其反向边f[(idx-2)^1] = f[idx-1]为流量
        f[idx-1] = f[idx-2] = 0; //删除这两条边
        S = s , T = t;
        print(ans+dinic());
    }
    return 0;
}
```

有源汇上下界最小流

有源汇上下界最小流只需要对有源汇上下界最大流进行变形即可，本质相同。

- 思路：

> 由有源汇最大流可知，原网络`G`上任意一可行流`f`经过变换得到的新网络上可行流`f'`都可以通过新网络上一条可行流（`S`出发的满流）`f0'`与`f s->t`之和得到，即
>
> $$|f^{'}| = |f_0^{'}| + |f_{s\to t}|$$
>
> 又有
>
> $$|f_{s\to t}| = -|f_{t\to s}|$$
>
> 因此，我们可以通过按照无源汇的方式建立好虚拟源点S流入虚拟汇点T流出的流网络。通过`dinic`算法验证此流网络是否满足`S`所有出边都是满流来考察新网络是否存在可行流。若存在，从原来的汇点`t`向原来的源点`s`做一遍`dinic`算法，从而消除`t -> s`路径上的所有增广路径以获得`t -> s`的最大流，由上述公式，`t -> s`为最大流，则`s -> t`为最小流，因此最终的答案即为：
>
> $$ans = |f_0^{'}| - |f_{max(t\to s)}|$$

代码奉上：

```
int n,m,S,T;
int e[M],ne[M],f[M],h[N],idx=0;
int q[M],d[N],cur[M],A[N];

void add(int a,int b,int 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 ++;
}

int find(int u,int limit){
    if(u == T) return limit;
    int flow = 0;

    for(int i=cur[u];~i && flow < limit ; i = ne[i]){
        cur[u] = i;

        int ver = e[i];
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit - flow));
            if(!t) d[ver] = -1;
            f[i] -= t , f[i^1] += t , flow += t;
        }
```

```
    }
    return flow;
}

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

    while(hh <= tt){
        int u = q[hh ++];
        for(int i=h[u];~i;i=ne[i]){
            int ver = e[i];
            if(d[ver] == -1 && f[i]){
                d[ver] = d[u] + 1;
                cur[ver] = h[ver];
                if(ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    return false;
}

int dinic(){
    int ans = 0 , flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
    return ans;
}
//=================================
int main(){
    memset(h,-1,sizeof h);

    int s , t , ans = 0;

    n = read() , m = read() , s = read() , t = read();
    S = 0 , T = n + 1;

    rep(i,1,m){
        int a = read() , b = read() , c = read() , d = read();
        add(a,b,d-c);
        A[a] += c , A[b] -= c;
    }
    int tot = 0;
    rep(i,1,n){
        if(A[i] > 0 ) add(i,T,A[i]);
        else if(A[i] < 0) add(S,i,-A[i]) , tot -= A[i];
    }
    add(t,s,INF);
    if(dinic() < tot) puts("No Solution");
    else{
        ans = f[idx - 1]; //s向t的流量
        f[idx - 1] = f[idx - 2] = 0;
        S = t, T = s;
        print(ans - dinic());
    }
    return 0;
}
```