21. 网络流

21.0 基本概念

26.0.1 流网络与流

流网络$G(V,E)$是一个有向图,图中每条边$(u,v)\in E$有一个非负的**容量值**$c(u,v)\geq 0$.若边集E包含一条边(u,v),则图中不存在反向边(事实上包含反向边的流网络可等价为一个不含反向边的流网络),且图中不存在自环.为方便,若$(u,v)\notin E$,定义$c(u,v)=0$.流网络中有两个特殊的节点:源点s和汇点t.为方便,假定每个节点都在s到t的某条路径上,即对$\forall$节点$v\in V$,流网络至少包含一条路径$s\to v\to t$,即流网络是连通的.因流网络中除源点外每个节点都至少有一条入边,则$|E|\geq |V|-1$.

设流网络$G(V,E)$的容量函数为c,源点为s,汇点为t,则G中的**流**是一个实值函数$f:V\times V\to\mathbb{R}$,满足如下两条性质:

①[**容量限制**] 对$\forall$节点$u,v\in V$,有$0\leq f(u,v)\leq c(u,v)$.

②[**流量守恒**] 对$\forall$节点$v\in V\backslash\{s,t\}$,有$\sum\limits_{v\in V}f(v,u)=\sum\limits_{v\in V}f(u,v)$,即除了源点和汇点外,其他节点不储存流量,即流入=流出.

若$(u,v)\notin E$,则节点u到节点v间无流,即$f(u,v)=0$.

称非负值$f(u,v)$为从节点u到节点v的流.流f的值$|f|$(不表示绝对值)定义为$|f|=\sum\limits_{v\in V}f(s,v)-\sum\limits_{v\in V}f(v,s)$,即$f$的值是从源点流出的总流量减流入源点的总流量.因此,流量守恒中$\sum\limits_{v\in V}f(v,s)=0$,将其保留在公式中为了后续讨论残量网络,在其中关心流入源点的流量.

对给定的流网络G、源点s和汇点t,所有的流中值最大的流成为G的**最大流**.最大流问题中未必只有一个源点和一个汇点.在多源点和多汇点的网络中,确定最大流问题可归约为单源点和单汇点的普通流网络,转化方法是建立一个超级源点s,对所有源点$S_i\ \ (i=1,2,\cdots)$,加入有向边(s,S_i),容量$c(s,S_i)=\infty$;建立一个超级汇点t,对所有汇点$T_i\ \ (i=1,2,\cdots)$,加入有向边(T_i,t),容量$c(T_i,t)=\infty$.此时,源点s为原来的多个源点S_i提供流量,汇点t消费原来的多个汇点T_i消费的流量.

Ford-Fulkerson方法(简称F-F方法)是解决最大流问题的方法之一,它包含了几种运行时间不相同的具体实现.该方法循环增加流的值.初始时,对所有节点$u,v\in V$,$f(u,v)=0$,即给出的初始流值为0.每次迭代增加图G的流值,具体地,在残量网络G_f中寻找增广路,若找到图G_f中一条增广路的边,即可辨别出G中具体的边,并修改这些边上的流量.虽F-F方法每次迭代都增加流值,但图G上的一条特定边的流量可能增加也可能减少.缩减某些边的流是必要的,这可让更多的流从与源点流向汇点.重复迭代过程,直至残留网络中不存在增广路.最大流最小割定理表明:该算法结束时将获得最大流.

给定流网络G和流量f,直观上,**残量网络**G_f由仍有空间对流量进行调整的边构成.流网络的一条边可允许的额外流量等于该边的容量减该边的流量,若差值为正,则将其加入G_f中,并将其**剩余容量**定义为$c_f(u,v)=c(u,v)-f(u,v)$.对图G中的边,只有能允许额外流量的边才能加入G_f中,流量=容量的边$\notin G_f$.此外,G_f中还可能包含G中不存在的边.因算法对流量进行操作的目标是增加总流量,则可能缩减某些特定边上的流量.为表示对一个正流量$f(u,v)$的缩减,将边(v,u)加入G_f中,定义其剩余容量$c_f(v,u)=f(u,v)$,即一条边能允许的反向流量最多抵消其正向流量.残量网络中的反向边允许将已发出的流量发送回去,流量从同一条边发送回去等价于缩减该边的流量.形式化地,对流网络$G=(V,E)$,设源点为s,汇点为t,f为

图中的一个流.对节点$u,v\in V$,定义剩余容量$c_f(u,v)=\begin{cases}c(u,v)-f(u,v),(u,v)\in E\\ f(v,u),(v,u)\in E\\ 0,\text{其他}\end{cases}$.因假定$(u,v)\in E$,则$(v,u)\notin E$,这表明对任一边,上述公式只有一种情况成立.对给定的流网络$G(V,E)$和一个流f,由f诱导的图G的残量网络$G_f=(V,E_f)$,其中$E_f=\{(u,v)\in V\times V:c_f(u,v)>0\}$,即残量网络的每条边(称为**残边**)应允许$>0$的流量通过.注意到$E_f$中的边要么是原$E$中的边,要么是其反向边,故$|E_f|\le 2|E|$.

残量网络G_f类似于一个容量为c_f的流网络,但它不符合流网络的定义,因为它可能同时包含边及其反向边,除这点区别外,残量网络与流网络性质相同,可在流网络的参量网络中定义一个流,它满足流的定义,但它针对的是残量网络G_f中的容量c_f.若f是G的一个流,f'是其对应残量网络G_f中的一个流,定义$f\uparrow f'$为流f'对流f的**递增**,$f\uparrow f':V\times V\to\mathbb{R}$定义为$(f\uparrow f')(u,v)=\begin{cases}f(u,v)+f'(u,v)-f'(v,u),(u,v)\in E\\ 0,\text{其他}\end{cases}$.该定义的只管解释遵循残量网络的定义,具体地,因残量网络中将流量发送到反向边上等价于在原网络中缩减流量,则将边(u,v)的流量增加$f'(u,v)$并减少$f'(v,u)$.残量网络中将流量发送回去称为**抵消操作**.

[**定理21.1**] 设流网络$G=(V,E)$的源点为s,汇点为t,f为G中的一个流,G_f是由f诱导的G的残量网络,f'是G_f中的一个流，则$f\uparrow f'$是G的一个流,其值$|f\uparrow f'|=|f|+|f'|$.

[**证**] (1)先证明$f\uparrow f'$满足容量限制和流量守恒.

①容量限制:注意到若边$(u,v)\in E$,则$c_f(v,u)=f(u,v)$,且$f'(v,u)\le c_f(v,u)=f(u,v)$,

则$(f\uparrow f')(u,v)=f(u,v)+f'(u,v)-f'(v,u)\ge f(u,v)+f'(u,v)-f(u,v)=f'(u,v)\ge 0$.

又$(f\uparrow f')(u,v)=f(u,v)+f'(u,v)-f'(v,u)\le f(u,v)+f'(u,v)$

$\le f(u,v)+c_f(u,v)=f(u,v)+c(u,v)-f(u,v)=c(u,v)$.

②流量守恒:因f和f'都遵循流量守恒,则对$\forall$节点$u\in V\backslash\{s,t\}$,有:

$$\sum_{v\in V}(f\uparrow f')(u,v)=\sum_{v\in V}f(u,v)+\sum_{v\in V}f'(u,v)-\sum_{v\in V}f'(v,u)=\sum_{v\in V}f(v,u)+\sum_{v\in V}f'(v,u)-\sum_{v\in V}f'(u,v)$$

$$=\sum_{v\in V}[f(v,u)+f'(v,u)-f'(u,v)]=\sum_{v\in V}(f\uparrow f')(v,u).$$

(2)求$|f\uparrow f'|$.

注意到图G中不允许包含反向边,则对每个节点$v\in V$,可有边(s,v)或(v,s),但不能同时存在.

定义$V_1=\{v:(s,v)\in E\}$为源点s有边可到达的节点的集合,$V_2=\{v:(v,s)\in E\}$为有边通往源点s的节点的集合,

则$V_1\bigcup V_2\subseteq V$.又因不存在反向边,则$V_1\bigcap V_2=\varnothing$.

$$|f\uparrow f'|=\sum_{v\in V}(f\uparrow f')(s,v)-\sum_{v\in V}(f\uparrow f')(v,s)=\sum_{v\in V_1}(f\uparrow f')(s,v)-\sum_{v\in V_2}(f\uparrow f')(v,s).$$

其中第二个等号成立是因为$(f\uparrow f')(w,x)$的值在$(w,x)\notin E$时为0.

代入$f\uparrow f'$的定义,并对项重新排序和重组得:

$$|f\uparrow f'|=\sum_{v\in V_1}[f(s,v)+f'(s,v)-f'(v,s)]-\sum_{v\in V_2}[f(v,s)+f'(v,s)-f'(s,v)]$$

$$=\sum_{v\in V_1}f(s,v)+\sum_{v\in V_1}f'(s,v)-\sum_{v\in V_1}f'(v,s)-\sum)v\in V_2f(v,s)-\sum_{v\in V_2}f'(v,s)+\sum_{v\in V_2}f'(s,v)$$

$$=\sum_{v\in V_1}f(s,v)-\sum_{v\in V_2}f(v,s)+\sum_{v\in V_1}f'(s,v)+\sum_{v\in V_2}f'(s,v)-\sum_{v\in V_1}f'(v,s)-\sum_{v\in V_2}f'(v,s)$$

$$= \sum_{v\in V_1} f(s,v) - \sum_{v\in V_2} f(v,s) + \sum_{v\in V_1 \bigcup V_2} f'(s,v) - \sum_{v\in V_1 \bigcup V_2} f'(v,s).$$

可将上述四个和式的v的范围都扩展到V上,因为每个额外的项的值都为0,

故$|f \uparrow f'| = \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) + \sum_{v\in V} f'(s,v) - \sum_{v\in V} f'(v,s) = |f| + |f'|.$

给定流网络$G(V,E)$和流f,**增广路径**(简称增广路)p是残量网络G_f中一条从源点s到汇点t的简单路径.由残量网络的定义,对一条增广路上的边(u,v),其流量至多可增加$c_f(u,v)$,不会违反原网络G中对边(u,v)或(v,u)的容量限制.称在一条增广路p上能为每条边增加的流量的最大值为路径p的**剩余容量**,定义为$c_f(p) = \min\{c_f(u,v) : (u,v) \in 路径p\}$.

[**定理21.2**] 设f是流网络$G(V,E)$中的一个流,p为其残量网络G_f中的一条增广路.定义函数$f_p : V \times V \to \mathbb{R}$为$f_p(u,v) = \begin{cases} c_f(p), (u,v) \in p \\ 0, 其他 \end{cases}$,则$f_p$是$G_f$中的一个流,其值$|f_p| = c_f(p) > 0$.

[**推论21.1**] 设f是流网络$G(V,E)$中的一个流,p为其残量网络G_f中的一条增广路.若将f增加f_p的量,则函数$f \uparrow f_p$是G中的一个流,其值$|f \uparrow f_p| = |f| + |f_p| > |f|$.

[**注**] 该推论表明:若将流f增加f_p的量,将得到G的另一个流,该流的值更接近最大值.

流网络$G(V,E)$中的一个**切割**(简称**割**)(S,T)将节点集V划分为S和$T = V - S$两集合,使得$s \in S, t \in T$.对G中的一个流f,定义横跨切割(S,T)的**净流量**$f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u)$,定义切割$(S,T)$的**容量**$c(S,T) = \sum_{u\in S}\sum_{v\in T} c(u,v)$.一个流网络中容量最小的切割称为该流网络的**最小割**.显然一个含n个节点的流网络可分为2^{n-2}个割.

[**注**] 流的定义和切割容量的定义不对称,这种不对称性是有意而为.对容量,只考虑从集合S出发进入集合T的边的容量,不考虑反向边上的容量;对流,考虑从S到T的流量减从T到S反方向的流量.

[**定理21.3**] 设流网络$G = (V,E)$的源点为s,汇点为t,f为G中的一个流,(S,T)是G的任一切割,则横跨(S,T)的净流量$f(S,T) = |f|$.

[**证**] 对$\forall$节点$u \in V\backslash\{s,t\}$,重写流量守恒的性质为$\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u) = 0$.

代入$|f|$的定义,对所有$S\backslash\{s\}$中的节点求和得:

$$|f| = \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) + \sum_{v\in S\backslash\{s\}} \left[\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u)\right].$$

展开RHS的求和项并重新组合得:

$$|f| = \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) + \sum_{v\in S\backslash\{s\}}\sum_{v\in V} f(u,v) - \sum_{v\in S\backslash\{s\}}\sum_{v\in V} f(v,u)$$

$$= \sum_{v\in V}\left[f(s,v) + \sum_{v\in S\backslash\{s\}} f(u,v)\right] + \sum_{v\in V}\left[f(v,s) + \sum_{v\in S\backslash\{s\}} f(v,u)\right] = \sum_{v\in V}\sum_{u\in S} f(u,v) - \sum_{v\in V}\sum_{u\in S} f(v,u).$$

因$V = S \bigcup T$,且$S \bigcap T = \varnothing$,则可将上式对$V$的求和分解为对$S$和$T$的求和,即:

$$|f| = \sum_{v\in S}\sum_{u\in S} f(u,v) + \sum_{v\in T}\sum_{u\in S} f(u,v) - \sum_{v\in S}\sum_{u\in S} f(v,u) - \sum_{v\in T}\sum_{u\in S} f(v,u)$$

$$=\sum_{v\in T}\sum_{u\in S}f(u,v)-\sum_{v\in T}\sum_{u\in S}f(v,u)+\left[\sum_{v\in S}\sum_{u\in S}f(u,v)-\sum_{v\in S}\sum_{i\in S}f(v,u)\right].$$

因对所有节点$x,y\in S$,项$f(x,y)$在每个和式中恰出现一次,故上述括号中的两项相等,

故$|f|=\sum_{u\in S}\sum_{v\in T}f(u,v)-\sum_{u\in S}\sum_{v\in T}f(v,u)=f(S,T).$

[**推论21.2**] 流网络G中任一流f的值不超过G中任一切割的容量.

[**证**] 设(S,T)为G的一个切割.

$$|f|=f(S,T)=\sum_{u\in S}\sum_{v\in T}f(u,v)-\sum_{u\in S}\sum_{v\in T}f(v,u)\le\sum_{u\in S}\sum_{v\in T}f(u,v)\le\sum_{u\in S}\sum_{v\in T}c(u,v)=c(S,T).$$

[**注**] 本推论说明如何用切割的容量来限定一个流的值,它表明:一个流网络中最大流的值不超过该网络最小割的容量.

[**定理21.4**] [**最大流最小割定理**] 设流网络$G=(V,E)$的源点为s,汇点为t,f为G中的一个流,则下述条件等价:①f是G的一个最大流;②残量网络G_f不存在增广路;③$|f|=c(S,T)$,其中$c(S,T)$是G的一个切割.

[**证**] ①⇒② 若f是G的一个最大流,且G_f中有一条增广路p,

由**推论21.1**:对f增加流量f_p可得到G中值更大的流,与f是最大流矛盾.

②⇒③ G_f中无增广路即不存在从源点s到汇点t的路径.

定义集合$S=\{u\in V:\text{在}G_f\text{中存在一条从}s\text{到}v\text{的路径}\},T=V-S.$

显然$s\in S$,而G_f中不存在从s到t的路径,故$t\notin S$.故划分(S,T)是G的一个切割.

考虑一对节点$u\in S,v\in T$:

(i)若边$(u,v)\in E$,则$f(u,v)=c(u,v)$,否则边$(u,v)\in E_f$,这将使得$v\in S$.

(ii)若边$(v,u)\in E$,则$f(v,u)=0$,否则$c_f(u,v)=f(v,u)>0$,则边$(u,v)\in E_f$,这将使得$v\in S$.

(iii)若边$(u,v),(v,u)\notin E$,则$f(u,v)=f(v,u)=0$,此时:

$$f(S,T)=\sum_{u\in S}\sum_{v\in T}f(u,v)-\sum_{v\in T}\sum_{u\in S}f(v,u)=\sum_{u\in S}\sum_{v\in T}c(u,v)-\sum_{v\in T}\sum_{u\in S}0=c(S,T).$$

由**定理21.3**:$|f|=f(S,T)=c(S,T)$.

③⇒① 由**推论21.2**:对所有切割(S,T),有$|f|\le c(S,T)$,又$|f|=c(S,T)$,故f是一个最大流.

26.0.2 费用流

费用流指一个流网络的所有最大可行流中费用的最小者或最大者.可行流的费用定义为流量乘路径长度.

残留网络中的反向边的费用定义为原网络中的正向边的费用的相反数.

一个有最大流的流网络必有费用流,但EK算法无法求出流网络中有孤立的环的情况的费用流.费用流的EK算法在最大流的EK算法的基础上将BFS换为SPFA,即求源点到汇点的最短路,同时受SPFA限制,流网络中不能出现负权回路.

[**证**] 设当前可行流f_1费用最小,通过SPFA在G_f中找到了一条新的增广路(它是当前G_f中的最短路),对应可行流f_2.

令可行流$f=f_1+f_2$,若f的费用非最小,设流f'的流量与f相等,但费用比f更小.

令可行流$f_2'=f'-f_1$,则$f'=f_1+f_2'$.因$|f'|=|f|$,则$|f_2|=|f_2'|,w(f_2')<w(f_2)$,

而费用 $=$ 流量 $\times$ 路径长度,这表明f_2'是G_f中比f_2更短的路径,与f_2是最短路矛盾.

故每次找最短路,可使流量相同的情况下费用最小.

[**注**] 当流网络中有孤立的环时不能用EK算法求费用流.

因有流量的边的反向边才能走,而任一可行流不出现环,则若原图中无负权回路,建反向边后也无负权回路.若原图的边权可能为负,则无法用SPFA的EK算法解决,只能用消圈法.

21.1 最大流

Ford-Fulkerson增广路算法的主流实现是Edmonds-Karp动能算法、Dinic算法、Improved Shortest Augmenting Path(ISAP)算法,还有效率更高但用得少的Highest Label Preflow Push算法(HLPP,最高标号预留推进算法).

FF增广路算法的思想:维护流网络的残留网络,每次在残留网络中找增广路,每次找到后将该增广路从残留网络中去掉,直至找不到增广路,此时的可行流是最大流.

网络流算法的时间复杂度上界很宽松,实际运行效率非常高.

[**EK算法**] 找增广路的过程用BFS实现(防止出现环),用一个$pre[]$数组记录每个节点的前驱即可记录路径.更新残留网络即计算反向边的流量:先求出从源点到汇点的流k,即从源点到汇点的路径上的边的容量最小值,则增广路中的正向边流量$-k$,反向边流量$+k$.时间复杂度$O(nm^2)$,1 s内能跑的点数和边数之和在$1\mathrm{e}3 \sim 1\mathrm{e}4$间.

[**Dinic算法**] 在EK算法的基础上引入分层图,每次增广多条路径,时间复杂度$O(n^2m)$,1 s内能跑的点数和边数之和在$1\mathrm{e}4 \sim 1\mathrm{e}5$间.

[**最大流建图思路**] 对给定的原问题构建流网络,证明原问题的可行解的集合与流网络的可行流的集合一一对应,即①任一个可行解都能在流网络里找到一条对应的可行流,即证容量限制和流量守恒;②任一条可行流都对应原问题的一个可行解.这样求原问题的最大值转化为求最大可行流.

21.1.1 模板

21.1.1.1 EK求最大流

题意

给定一个含n $(2 \le n \le 1000)$个节点、m $(1 \le m \le 1\mathrm{e}4)$条边的有向图(可能存在重边和自环)及每条边的容量(非负),求从源点s到汇点t $(s \ne t)$的最大流,若无法到达,输出0.

有向图用m行输入描述,每个输入包含三个正数u, v, c,表示从点u到点v存在一条容量为c $(0 \le c \le 1\mathrm{e}4)$的有向边,点编号$1 \sim n$.

思路

用邻接表存图.因更新残留网络时需加反向边,则可在存图时连续加边,如下标0存正向边,下标1存0的反向边,下标2存正向边,下标3存2的反向边,$\cdots$,下标i存的正向边的反向边是下标$i \ \hat{}\ 1$.

总时间复杂度$O(nm^2)$.

代码

```
struct EKMaximumFlow {
    int n, m;  // 节点数、边数
    int S, T;  // 源点、汇点
    vector<int> head, edge, capa, nxt;  // capa[i]表示边i的容量
    int idx;  // 当前用到的边的编号
    vector<int> minCapa;  // minCapa[i]表示到节点i的所有边的容量的最小值
    vector<int> pre;  // pre[i]表示节点i的前驱边的编号
    vector<bool> vis;

    EKMaximumFlow(int _n, int _m, int _S, int _T)
        :n(_n), m(_m), S(_S), T(_T), idx(0),
        head(n + 5, -1), edge(2 * (m + 5)), capa(2 * (m + 5)), nxt(2 * (m + 5)),
        minCapa(n + 5), pre(n + 5), vis(n + 5) {
        build();
    }

    void add(int x, int y, int z) {  // 建边x -> y, 容量为z
        edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
        edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[x = y] = idx++;  // 反向边,
流量初始为0
    }

    void build() {  // 构建流网络
        while (m--) {
            int x, y, z; cin >> x >> y >> z;
            add(x, y, z);
        }
    }

    bool bfs() {  // 返回是否找到增广路
        fill(all(vis), false);

        queue<int> que;
        que.push(S);
        vis[S] = true, minCapa[S] = INF;  // 初始时未经过边

        while (que.size()) {
            int u = que.front(); que.pop();
            for (int i = head[u]; ~i; i = nxt[i]) {
                int v = edge[i];
                if (!vis[v] && capa[i]) {  // 节点未遍历过且边还有容量
                    vis[v] = true;
                    minCapa[v] = min(minCapa[u], capa[i]);
                    pre[v] = i;  // 记录前驱边的编号

                    if (v == T) return true;  // 存在增广路
                    que.push(v);
                }
            }
        }
        return false;  // 不存在增广路
    }

    ll ek() {
        ll res = 0;
```

```
        while (bfs()) {  // 存在增广路
            res += minCapa[T];
            for (int i = T; i != S; i = edge[pre[i] ^ 1]) {  // 更新残量网络
                capa[pre[i]] -= minCapa[T];  // 正向边减容量
                capa[pre[i] ^ 1] += minCapa[T];  // 反向边加容量
            }
        }
        return res;
    }
};

void solve() {
    int n, m, S, T; cin >> n >> m >> S >> T;

    EKMaximumFlow solver(n, m, S, T);
    cout << solver.ek() << endl;
}

int main() {
    solve();
}
```

21.1.1.2 Dinic求最大流

题意

给定一个含n $(2 \le n \le 1e4)$个节点、m $(1 \le m \le 1e5)$条边的有向图(可能存在重边和自环)及每条边的容量(非负),求从源点s到汇点t $(s \ne t)$的最大流,若无法到达,输出0.

有向图用m行输入描述,每个输入包含三个正数u, v, c,表示从点u到点v存在一条容量为c $(0 \le c \le 1e4)$的有向边,点编号$1 \sim n$.

思路

若流网络中存在环,用EK算法每次只增广一条路径可能会死循环.Dinic算法引入分层图,要求增广时只能从前一层走到后一层,这保证不会出现环路.从源点开始BFS,则每个节点所在的层数为其到源点的最短距离.

Dinic算法的步骤:①从源点开始BFS,建立分层图;②在分层图上DFS,将所有能增广的路径都增广.此处的DFS无回溯的过程,故是多项式级别的时间复杂度.总时间复杂度$O(n^2m)$.

[**当前弧优化**] 注意到从节点u增广路径时会以某种顺序枚举u的出边,若本次增广中已将一条出边的容量填满,则下次增广时可跳过已满的出边.称每次增广的第一条边为**当前弧**.

代码

```
struct DinicMaximumFlow {
    int n, m;  // 节点数、边数
    int S, T;  // 源点、汇点
    vector<int> head, edge, capa, nxt;  // capa[i]表示边i的容量
    int idx;  // 当前用到的边的编号
    vector<int> minCapa;  // minCapa[i]表示到节点i的所有边的容量的最小值
    vector<int> level;  // level[i]表示节点i在分层图中的层数
    vector<int> cur;  // cur[i]表示节点i的当前弧

    DinicMaximumFlow(int _n, int _m, int _S, int _T)
```

```
        :n(_n), m(_m), S(_S), T(_T), idx(0),
        head(n + 5, -1), edge(2 * (m + 5)), capa(2 * (m + 5)), nxt(2 * (m + 5)),
        minCapa(n + 5), level(n + 5), cur(n + 5) {
        build();
    }

    void add(int x, int y, int z) {  // 建边x -> y, 容量为z
        edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
        edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[x = y] = idx++;  // 反向边, 流量初始为0
    }

    void build() {  // 构建流网络
        while (m--) {
            int x, y, z; cin >> x >> y >> z;
            add(x, y, z);
        }
    }

    bool bfs() {  // 返回是否找到增广路
        fill(all(level), -1);

        queue<int> que;
        que.push(S);
        level[S] = 0;  // 源点的层数为0
        cur[S] = head[S];  // 记录当前弧

        while (que.size()) {
            int u = que.front(); que.pop();
            for (int i = head[u]; ~i; i = nxt[i]) {
                int v = edge[i];
                if (level[v] == -1 && capa[i]) {  // 节点未遍历过且边还有容量
                    level[v] = level[u] + 1;
                    cur[v] = head[v];  // 更新当前弧

                    if (v == T) return true;  // 存在增广路
                    que.push(v);
                }
            }
        }
        return false;  // 不存在增广路
    }

    int find(int u, int lim) {  // 从节点u开始增广, 流量限制为lim
        if (u == T) return lim;  // 到达汇点

        int flow = 0;  // 总流量
        for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广, 流量不超过限制
            cur[u] = i;  // 更新当前弧
            int v = edge[i];
            if (level[v] == level[u] + 1 && capa[i]) {  // 节点v在u的下一层, 且边还有容量
                ll tmp = find(v, min(capa[i], lim - flow));  // 递归增广v
                if (!tmp) level[v] = -1;  // 节点v无法到达汇点, 将其删去, 通过将层数置为-1实现

                capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
            }
        }
```

```
        return flow;
    }

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

void solve() {
    int n, m, S, T; cin >> n >> m >> S >> T;

    DinicMaximumFlow solver(n, m, S, T);
    cout << solver.dinic() << endl;
}

int main() {
    solve();
}
```

21.1.1.3 Delivery Bears

原题指路:https://codeforces.com/problemset/problem/653/D

题意 (2 s)

给定一张包含编号$1 \sim n$的n $(2 \leq n \leq 50)$个节点和编号$1 \sim m$的m $(1 \leq m \leq 500)$条边的有向图,其中i $(1 \leq i \leq m)$号边从节点a_i $(1 \leq a_i \leq n)$指向节点b_i $(1 \leq b_i \leq n)$,该边的容量为整数c_i $(1 \leq c_i \leq 1\mathrm{e}6)$,图中至少存在一条从节点1到节点$n$的路径.现有$x$个人,每个人要将相等重量的货物从节点1运输到节点$n$,每个人可选择不同的路径,通过每条边的人的货物总量不超过该边的容量.问所有人能运输的货物的重量之和的最大值,误差不超过$1\mathrm{e}-6$.

思路

注意到每个人运输的重量可能是小数,不方便处理,考虑将其转化为整数.设每人运输的货物的重量为d,则第i条边能经过$\left\lfloor \frac{c_i}{d} \right\rfloor$个人,问题转化为求$x \cdot d$的最大值.这样每个人转化为1的流量,而$c_i$也是整数,方便处理.

注意到每个人运输的货物的重量d越小越容易存在合法方案,故d具有二段性,二分d值后,检查是否存在合法方案.

对固定的d值,将每条边的容量定义为该边最多能经过的次数,问题转化为求源点1到汇点n的最大流,则存在合法方案当且仅当该流网络的最大流$\geq x$.

若本题直接二分到对应精度的方法,精度过低会WA,精度过高会TLE.故需用限制二分次数的方法,二分至精度$(1\mathrm{e}-10) \sim (1\mathrm{e}-12)$约$60 \sim 70$次.

代码

```
struct DinicMaximumFlow {
  int n, m, s, t;  // 点数、边数、源点、汇点
  int idx;  // 当前用到的边的编号
  vector<int> head, edge, nxt;
  vector<ll> capa;  // capa[i]表示边i的容量
  vector<int> level;  // level[i]表示节点i在分层图中的层数
```

```
  vector<int> cur;  // cur[i]表示节点i的当前弧

  DinicMaximumFlow(int _n, int _m, int _s, int _t) :
  n(_n + 5), m(2 * (_m + 5)), s(_s), t(_t) {  // 边开两倍
    idx = 0;
    head.resize(n + 1), edge.resize(m + 1), capa.resize(m + 1), nxt.resize(m + 1);
    level.resize(n + 1), cur.resize(n + 1);
    fill(all(head), -1);
  }

  void add(int x, int y, ll z) {  // 建边x->y,容量为z
    edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
    edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[x = y] = idx++;  // 反向边,流量初始为0
  }

  bool bfs() {  // 返回是否找到增广路,若有增广路则建立分层图
    fill(all(level), -1);

    queue<int> que;
    que.push(s);
    level[s] = 0;  // 源点的层数为0
    cur[s] = head[s];  // 记录当前弧

    while (que.size()) {
      int u = que.front(); que.pop();
      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (level[v] == -1 && capa[i]) {  // 当前还未被搜过且边还有流量
          level[v] = level[u] + 1;
          cur[v] = head[v];  // 记录当前弧

          if (v == t) return true;  // 能到达汇点,即存在增广路
          que.push(v);
        }
      }
    }
    return false;  // 不能到达汇点,即不存在增广路
  }

  int find(int u, int lim) {  // 从节点u开始增广,流量限制为lim
    if (u == t) return lim;  // 到达汇点

    int flow = 0;  // 总流量
    for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广,流量不超过限制
      cur[u] = i;  // 更新当前弧
      int v = edge[i];
      if (level[v] == level[u] + 1 && capa[i]) {  // v在u的下一层,且边还有容量
        ll tmp = find(v, min(capa[i], (ll)lim - flow));  // 递归增广节点v
        if (!tmp) level[v] = -1;  // 节点v无法到达汇点,将其删去,可通过将其层数置为-1实现

        capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
      }
    }
    return flow;
  }

  ll dinic() {
```

```
    ll res = 0;
    ll flow;  // 流量
    while (bfs())  // 当前还有增广路
      while (flow = find(s, INF)) res += flow;  // 将所有能增广的路径增广
    return res;
  }
};

const double eps = 1e-8;

int cmp(double a, double b) {
  if (fabs(a - b) < eps) return 0;
  else return a < b ? -1 : 1;
}

void solve() {
  int n, m, x; cin >> n >> m >> x;
  vector<tuple<int, int, int>> edges(m);
  for (auto& [a, b, c] : edges) cin >> a >> b >> c;

  auto check = [&](double d)->bool {
    DinicMaximumFlow solver(n, m, 1, n);
    for (auto [a, b, c] : edges)
      solver.add(a, b, c / d);
    return cmp(solver.dinic(), x) >= 0;
  };

  double l = 0, r = 1e6;
  /*while (cmp(l, r)) {  // 这样二分会TLE
    double mid = (l + r) / 2;
    if (check(mid)) l = mid;
    else r = mid;
  }*/
  for (int i = 0; i < 60; i++) {  // 限制次数
    double mid = (l + r) / 2;
    if (check(mid)) l = mid;
    else r = mid;
  }
  cout << fixed << setprecision(12) << l * x << endl;
}

int main() {
  solve();
}
```

21.1.2 二分图匹配

21.1.2.1 飞行员配对方案

题意

现两人组队,其中一人来自A组,另一人来自B组.每一个B组的人能与若干个A组的人组队,问最多能组多少队.

第一行输入两个整数m, n $(1 < m < n < 100)$,分别表示B组、A组的人数,B组人编号$1 \sim m$,A组人编号$(m+1) \sim n$.接下来输入若干行,每行包含两个整数i, j,表示B组的i号能与A组的j号组队.最后一行输入两个-1表示输入结束.

第一行输出最多能组的队数M.接下来M行每行输出两个整数i, j,表示B组的i号与A组的j号配对.

思路

显然求二分图的最大匹配.匈牙利算法的时间复杂度$O(nm)$,可过.下面讨论最大流的做法.

设二分图的左半边是B组的人对应的节点,因每个B组的人至多组队依次,故源点S分别向它们连一条容量为1的边;右半边是A组的人对应的节点,它们向汇点T连一条容量为1的边.若B组的i号能与A组的j号组队,则二分图左半边的i号节点连一条到右半边j号节点的容量为1的边.

下证原问题的可行解与流网络中整数值的可行流一一对应.

(1)原问题的一个可行解对应到流网络中是选择二分图两半边之间的若干条无公共节点的边,将它们流量设为1.

将原点到二分图左半边选中的节点的流量、二分图右半边选中的节点到汇点的流量设为1.

下证构造出来的流是可行流.

[**证**] ①流量限制显然满足.

②若节点未被选中,则它流入和流出的流量都为0;否则它流入和流出的流量都为1.

(2)流网络中二分图两半边之间所有有流量的边的两端点即为二分图的最大匹配.

下证上述匹配是合法匹配.显然若它是合法匹配,则它是最大匹配.

[**证**] 因只考虑整数值可行流,且边的容量都为1,则每条边的容量是0或1.

①对流量为0的边,由流量守恒,流入其端点的流量和从其端点流出的流量都为0,即它不在二分图的匹配边中.

②对流量为1的边,流入其端点的流量和从其端点流出的流量都为1,即它在二分图的匹配边中.

综上,流网络中二分图两半边的节点至多属于一条匹配边.

问题转化为求流网络的整数值可行流的最大值.因Dinic算法中用到的变量都是整型,故它求得的所有可行流的最优解也是整数值可行流的最优解.总时间复杂度$O\left(\sqrt{n}m\right)$.类似于EK算法,匈牙利算法找匹配边的过程即流网络中找增广路的过程,即匈牙利算法是每次只找一条增广路的最大流算法.

输出方案时,只需遍历二分图两半边之间的正向边,输出满流量的边的两端点即可.

从源点、汇点到二分图的节点有n条边,二分图两半边之间最多$50^2 = 2500$条边,则最多2600条边,加上反向边,最多需要5200条边.

代码

```
namespace Dinic_Maximum_Flow {
  static const int MAXN = 105, MAXM = 5205;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
```

```
  int level[MAXN];  // level[i]表示节点i在分层图中的层数
  int cur[MAXN];  // cur[i]表示节点i的当前弧

  void add(int x, int y, int z) {  // 建边x->y,容量为z
    edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
    edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[x = y] = idx++;  // 反向边,流量初始为0
  }

  bool bfs() {  // 返回是否找到增广路,若有增广路则建立分层图
    memset(level, -1, so(level));

    qi que;
    que.push(s);
    level[s] = 0;  // 源点的层数为0
    cur[s] = head[s];  // 记录当前弧

    while (que.size()) {
      int u = que.front(); que.pop();
      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (level[v] == -1 && capa[i]) {  // 当前还未被搜过且边还有流量
          level[v] = level[u] + 1;
          cur[v] = head[v];  // 记录当前弧

          if (v == t) return true;  // 能到达汇点,即存在增广路
          que.push(v);
        }
      }
    }
    return false;  // 不能到达汇点,即不存在增广路
  }

  int find(int u, int lim) {  // 从节点u开始增广,流量限制为lim
    if (u == t) return lim;  // 到达汇点

    int flow = 0;  // 总流量
    for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广,流量不超过限制
      cur[u] = i;  // 更新当前弧
      int v = edge[i];
      if (level[v] == level[u] + 1 && capa[i]) {  // v在u的下一层,且边还有容量
        ll tmp = find(v, min(capa[i], lim - flow));  // 递归增广节点v
        if (!tmp) level[v] = -1;  // 节点v无法到达汇点,将其删去,可通过将其层数置为-1实现

        capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
      }
    }
    return flow;
  }

  ll dinic() {
    ll res = 0;
    ll flow;  // 流量
    while (bfs())  // 当前还有增广路
      while (flow = find(s, INF)) res += flow;  // 将所有能增广的路径增广
    return res;
  }
}
```

```cpp
using namespace Dinic_Maximum_Flow;

void solve() {
  memset(head, -1, so(head));

  cin >> m >> n;
  s = 0, t = n + 1;  // 源点、汇点
  for (int i = 1; i <= m; i++) add(s, i, 1);  // 从源点到二分图左半边节点连一条容量为1的边
  for (int i = m + 1; i <= n; i++) add(i, t, 1);  // 从二分图左半边的节点到汇点连一条容量为1的边
  int a, b;
  while (cin >> a >> b, ~a) add(a, b, 1);  // 从二分图左半边的节点到右半边的节点连一条容量为1的边

  cout << dinic() << endl;
  for (int i = 0; i < idx; i += 2) {  // 枚举正向边
    if (edge[i] > m && edge[i] <= n && !capa[i])  // 是二分图两半边之间的满流量的边
      cout << edge[i ^ 1] << ' ' << edge[i] << endl;
  }
}

int main() {
  solve();
}
```

21.1.2.2 圆桌问题

题意

有来自m个不同单位的代表参加会议,每个单位的代表数为$r_1,\cdots,r_m$.会场有编号$1\sim n$的n张桌子,每张桌子可容纳$c_i\ \ (i=1,\cdots,n)$个代表.现希望来自同一单位的代表在不同桌子,求一个满足的方案.

第一行输入两个整数$m,n\ \ (1\le m\le 150,1\le n\le 270)$,分别表示单位数、桌子数.第二行输入$m$个整数$r_1,\cdots,r_m\ \ (1\le r_i\le 100)$.第三行输入$n$个整数$c_1,\cdots,c_n\ \ (1\le c_i\le 100)$.

若无解,第一行输出0;否则第一行输出1,接下来m行输出每个单位的代表所在的桌号.若有多组解,输出任一组.

思路

二分图的左半边节点为m个单位,右半边节点为n张桌子,左半边的每个节点向右半边的每个节点连一条容量为1的边,转化为二分图的多重匹配问题.因左半边的每个节点可能与多个右半边的节点匹配,设左半边的i号节点对应的单位有r_i个代表,则从源点S到左半边的i号节点连一条容量为r_i的边.设右半边的i号节点对应的桌子能容纳c_i个代表,则从右半边的i号节点到汇点T连一条容量为c_i的边.

下证原问题的可行解与流网络中整数值的可行流一一对应.

(1)原问题的可行解对应到流网络中是二分图两半边之间的一些边,将选中的边的流量设为1.未选中的边的流量设为0.

从源点到二分图左半边的节点的流量等于对应单位在最优匹配中的代表数.

从二分图右半边的节点到汇点的流量等于对应桌子在最优匹配中实际容纳的代表数.

显然构造出的流是可行流.

(2)流网络的可行流中二分图两半边之间的满流的边即为选择的边.

下证构造出来的方案是原问题的可行解.

[**证**] ①二分图两半边之间的边的容量都为1,故每个单位至多向每个桌子派出一个代表.

②显然从源点到二分图左半边的节点的流量不超过各单位的代表数,从二分图右半边的节点到汇点的流量不超过各桌子的最大容量.

若原问题有解,则所有代表都有桌子坐,即从源点到二分图左半边的节点的边都满流,亦即最大流等于总代表数.

节点数$MAXN$为单位数+桌子数≤ 420,边数为二分图的所有nm条边加上从源点、汇点到二分图的边,最多为$(150 \times 270 + MAXN) \times 2$.源点编号0,二分图左半边的节点编号$1 \sim m$,二分图右半边的节点编号$(m+1) \sim (n+m)$,汇点编号$(n+m+1)$.

代码

```cpp
namespace Dinic_Maximum_Flow {
  static const int MAXN = 425, MAXM = (150 * 270 + MAXN) << 1;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  int level[MAXN];  // level[i]表示节点i在分层图中的层数
  int cur[MAXN];  // cur[i]表示节点i的当前弧

  void add(int x, int y, int z) {  // 建边x->y,容量为z
    edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
    edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[x = y] = idx++;  // 反向边,流量初始为0
  }

  bool bfs() {  // 返回是否找到增广路,若有增广路则建立分层图
    memset(level, -1, so(level));

    qi que;
    que.push(s);
    level[s] = 0;  // 源点的层数为0
    cur[s] = head[s];  // 记录当前弧

    while (que.size()) {
      int u = que.front(); que.pop();
      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (level[v] == -1 && capa[i]) {  // 当前还未被搜过且边还有流量
          level[v] = level[u] + 1;
          cur[v] = head[v];  // 记录当前弧

          if (v == t) return true;  // 能到达汇点,即存在增广路
          que.push(v);
        }
      }
    }
    return false;  // 不能到达汇点,即不存在增广路
  }

  int find(int u, int lim) {  // 从节点u开始增广,流量限制为lim
    if (u == t) return lim;  // 到达汇点

    int flow = 0;  // 总流量
    for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广,流量不超过限制
```

```
      cur[u] = i;  // 更新当前弧
      int v = edge[i];
      if (level[v] == level[u] + 1 && capa[i]) {  // v在u的下一层,且边还有容量
        ll tmp = find(v, min(capa[i], lim - flow));  // 递归增广节点v
        if (!tmp) level[v] = -1;  // 节点v无法到达汇点,将其删去,可通过将其层数置为-1实现

        capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
      }
    }
    return flow;
  }

  ll dinic() {
    ll res = 0;
    ll flow;  // 流量
    while (bfs())  // 当前还有增广路
      while (flow = find(s, INF)) res += flow;  // 将所有能增广的路径增广
    return res;
  }
}
using namespace Dinic_Maximum_Flow;

void solve() {
  memset(head, -1, so(head));

  cin >> m >> n;

  int sum = 0;  // 总代表数
  s = 0, t = n + m + 1;  // 源点、汇点
  for (int i = 1; i <= m; i++) {
    int r; cin >> r;
    add(s, i, r);  // 从源点到二分图左半边的节点连一条容量为代表数的边
    sum += r;
  }
  for (int i = 1; i <= n; i++) {
    int c; cin >> c;
    add(m + i, t, c);  // 从二分图右半边的节点到汇点连一条容量为桌子最大容纳数的边
  }
  for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++)
      add(i, m + j, 1);  // 从二分图左半边的节点到右半边的节点连一条容量为1的边
  }

  if (dinic() != sum) cout << 0;  // 最大流不等于总代表数时无解
  else {
    cout << 1 << endl;
    for (int i = 1; i <= m; i++) {  // 枚举每个单位
      for (int j = head[i]; ~j; j = nxt[j]) {  // 枚举每条边
        if (edge[j] > m && edge[j] <= n + m && !capa[j])  // 是二分图两半边之间的满流量的边
          cout << edge[j] - m << ' ';  // 注意-m得到桌子的原编号
      }
      cout << endl;
    }
  }
}

int main() {
  solve();
```

```
101 }
```

21.1.3 上下界可行流

21.1.3.1 无源汇上下界可行流

题意

给定一个包含编号$1\sim n$的n个节点和编号$1\sim m$的m条边的有向图,每条边都有一个流量下界和流量上界.求一个可行流使得所有边的流量符合限制.

第一行输入两个整数n,m $(1\le n\le 200,1\le m\le 10200)$.接下来$m$行每行输入四个整数$a,b,c,d$ $(1\le a,b\le n,0\le c\le d\le 1e4)$,表示存在一条从节点$a$到节点$b$的有向边,流量下界为$c$,上界为$d$.

若无解,第一行输出"NO";否则第一行输出"YES".接下来m行每行输出一个整数,其中第i $(1\le i\le m)$行的输出表示第i条边的流量.若有多组解,输出任一组.

思路

设原流网络为G,其中的一条可行流为f.考虑将其它们变换为另一个流网络G'和另一条可行流f',使得G'是普通的流网络,即其中存在源点和汇点,且边的流量只有上界没有下界.

对边(u,v),设其容量上、下界分别为$c_u(u,v)$、$c_l(u,v)$,流量为$f(u,v)$,则$c_l(u,v)\le f(u,v)\le c_u(u,v)$,进而$0\le f(u,v)-c_l(u,v)\le c_u(u,v)-c_l(u,v)$,其中$f(u,v)-c_l(u,v)\ge 0$,即它符合普通的流网络的流量限制.这样构造出的流未必满足流量守恒,因为每个节点的所有入边$f_{入}$和所有出边的流量$f_{出}$都减去了流量下界,但入边数与出边数未必相等.为使得流量守恒,在G'中从源点到$f_{入}$不足的节点连一条容量为$c_{入}-c_{出}$的边,从$f_{出}$多余的点向汇点连一条容量为$c_{出}-c_{入}$的边.为保证流量守恒,新加入的源点的出边都要满流,即G中的无源汇上下界可行流对应G'的一个最大流.

下证(G',f')与(G,f)一一对应.

(1)G中的流$f(u,v)$满足$c_l(u,v)\le f(u,v)\le c_u(u,v)$,$G'$中的流$f'(u,v)$满足$0\le f'(u,v)=f(u,v)-f_l(u,v)\le c_u(u,v)$.

将$f'(u,v)$的每条边的流量限制加上$c_l(u,v)$即得$f(u,v)$的一条可行流,即满足容量限制.

(2)由G'中从源点到$f_{入}$不足的节点连一条容量为$c_{入}-c_{出}$的边,从$f_{出}$多余的点向汇点连一条容量为$c_{出}-c_{入}$的边知流量守恒.

边数为原图的边数+新图中源点的出边和汇点的入边,新加入的边共n条.

代码

```
1  namespace Dinic_Range_Flow {
2    static const int MAXN = 425, MAXM = (150 * 270 + MAXN) << 1;  // 边开两倍
3    int n, m, s, t;  // 点数、边数、源点、汇点
4    int head[MAXN], edge[MAXM], capa[MAXM], low[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容
   量、low[i]表示边i的流量下界
5    int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
6    int level[MAXN];  // level[i]表示节点i在分层图中的层数
7    int cur[MAXN];  // cur[i]表示节点i的当前弧
8    ll diff[MAXN];  // diff[i]表示节点i的流入的流量与流出的流量之差
9
10   void add(int a, int b, int c, int d) {  // 建边a->b,流量范围[c,d]
11     edge[idx] = b, capa[idx] = d - c, low[idx] = c, nxt[idx] = head[a], head[a] = idx++;
   // 正向边容量为上界-下界
```

```
    edge[idx] = a, capa[idx] = 0, nxt[idx] = head[b], head[b] = idx++;  // 反向边无需记录流量下界
  }

  bool bfs() {  // 返回是否找到增广路,若有增广路则建立分层图
    memset(level, -1, so(level));

    qi que;
    que.push(s);
    level[s] = 0;  // 源点的层数为0
    cur[s] = head[s];  // 记录当前弧

    while (que.size()) {
      int u = que.front(); que.pop();
      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (level[v] == -1 && capa[i]) {  // 当前还未被搜过且边还有流量
          level[v] = level[u] + 1;
          cur[v] = head[v];  // 记录当前弧

          if (v == t) return true;  // 能到达汇点,即存在增广路
          que.push(v);
        }
      }
    }
    return false;  // 不能到达汇点,即不存在增广路
  }

  int find(int u, int lim) {  // 从节点u开始增广,流量限制为lim
    if (u == t) return lim;  // 到达汇点

    int flow = 0;  // 总流量
    for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广,流量不超过限制
      cur[u] = i;  // 更新当前弧
      int v = edge[i];
      if (level[v] == level[u] + 1 && capa[i]) {  // v在u的下一层,且边还有容量
        ll tmp = find(v, min(capa[i], lim - flow));  // 递归增广节点v
        if (!tmp) level[v] = -1;  // 节点v无法到达汇点,将其删去,可通过将其层数置为-1实现

        capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
      }
    }
    return flow;
  }

  ll dinic() {
    ll res = 0;
    ll flow;  // 流量
    while (bfs())  // 当前还有增广路
      while (flow = find(s, INF)) res += flow;  // 将所有能增广的路径增广
    return res;
  }
}
using namespace Dinic_Range_Flow;

void solve() {
  memset(head, -1, so(head));

```

```
  cin >> n >> m;

  s = 0, t = n + 1;  // 源点、汇点
  for (int i = 0; i < m; i++) {
    int a, b, c, d; cin >> a >> b >> c >> d;
    add(a, b, c, d);
    diff[a] -= c, diff[b] += c;  // 更新节点流入的流量与流出的流量之差
  }

  ll sum = 0;  // 新图中新加入的源点的出边的流量之和
  for (int i = 1; i <= n; i++) {
    if (diff[i] > 0) add(s, i, 0, diff[i]), sum += diff[i];  // 流量不足,从源点补流量
    else if (diff[i] < 0) add(i, t, 0, -diff[i]);  // 流量多余,向汇点出流量
  }

  if (dinic() != sum) cout << "NO";
  else {
    cout << "YES" << endl;
    for (int i = 0; i < m * 2; i += 2)  // 枚举正向边
      cout << capa[i ^ 1] + low[i] << endl;  // 原图的流量是新图的流量+流量下界
  }
}

int main() {
  solve();
}
```

21.1.3.2 有源汇上下界最大流

题意

给定一个包含编号$1 \sim n$的n个节点和编号$1 \sim m$的m条边的有向图,每条边都有一个流量下界和流量上界.给定源点s和汇点t,求一个从s到t的最大流.

第一行输入两个整数n, m $(1 \leq n \leq 202, 1 \leq m \leq 9999)$.接下来$m$行每行输入四个整数$a, b, c, d$ $(1 \leq a, b \leq n, 0 \leq c \leq d \leq 1e5)$,表示存在一条从节点$a$到节点$b$的有向边,流量下界为$c$,上界为$d$.

若无解输出"No Solution",否则输出最大流.

思路

从汇点连一条到源点的容量为INF的边,转化为无源汇上下界可行流,设其对应的新图G'中新加入的源点和汇点分别为S、T.

下面讨论如何求从s到t的最大流.注意G'中从S到T的最大流未必是从s到t的最大流,即使s到t不存在增广路.

代码

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<iomanip>
#include<cmath>
#include<cstring>
#include<algorithm>
```

```cpp
#include<complex>
#include<array>
#include<vector>
#include<queue>
#include<stack>
#include<set>
#include<map>
#include<list>
#include<unordered_set>
#include<unordered_map>
#include<bitset>
#include<valarray>
#include<sstream>
#include<functional>
#include<cassert>
#include<random>
#include<numeric>
using namespace std;
typedef unsigned int uint;
typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<bool> vb;
typedef queue<int> qi;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef pair<double, double> pdd;
typedef tuple<int, int, int> tiii;
typedef vector<pii> vii;
typedef vector<pll> vll;
typedef queue<pii> qii;
typedef complex<double> cp;
#define umap unordered_map
#define uset unordered_set
#define pque priority_queue
#define IOS ios::sync_with_stdio(0); cin.tie(0), cout.tie(0);
#define CaseT int CaseT; cin >> CaseT; while(CaseT--)
#define endl '\n'
#define npt nullptr
#define so sizeof
#define pb push_back
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define hashset_finetune(p) (p).reserve(1024); (p).max_load_factor(0.25);  // 哈希表防卡
const double eps = 1e-7;
const double pi = acos(-1.0);
const int INF = 0x3f3f3f3f;
const ll INFF = 0x3f3f3f3f3f3f3f3f;
// const int dx[4] = { -1,0,1,0 }, dy[4] = { 0,1,0,-1 };
// ------------------------------------------------------------
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll lcm(ll a, ll b) { return a * b / gcd(a, b); }
ll lowbit(ll x) { return x & (-x); }
int cmp(double a, double b) {
    if (fabs(a - b) < eps) return 0;
    else if (a > b) return 1;
    else return -1;
```

```
}
int sgn(double x) {
    if (fabs(x) < eps) return 0;
    else if (x > 0) return 1;
    else return -1;
}

template<typename T>
void read(T& x) {
    x = 0;
    T sgn = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') sgn = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = (x << 3) + (x << 1) + (ch & 15);
        ch = getchar();
    }
    x *= sgn;
}

template<typename T>
void write(T x) {
    if (x < 0) {
        putchar('-');
        x *= -1;
    }
    if (x < 10) putchar(x + 48);
    else write(x / 10), putchar(x % 10 + 48);
}

ll qpow(ll a, ll k, ll MOD) {
    ll res = 1;
    while (k) {
        if (k & 1) res = res * a % MOD;
        k >>= 1;
        a = a * a % MOD;
    }
    return res;
}

ll qpow(ll a, ll k) {  // 注意可能爆ll
    ll res = 1;
    while (k) {
        if (k & 1) res = res * a;
        k >>= 1;
        a *= a;
    }
    return res;
}
// ------------------------------------------------------------------

namespace Dinic_Maximum_Range_Flow {
  static const int MAXN = 205, MAXM = (MAXN + (int)1e4) << 1;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量
```

```cpp
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  int level[MAXN];  // level[i]表示节点i在分层图中的层数
  int cur[MAXN];  // cur[i]表示节点i的当前弧
  ll diff[MAXN];  // diff[i]表示节点i的流入的流量与流出的流量之差

  void add(int x, int y, int z) {  // 建边x->y,容量为z
    edge[idx] = y, capa[idx] = z, nxt[idx] = head[x], head[x] = idx++;  // 正向边
    edge[idx] = x, capa[idx] = 0, nxt[idx] = head[y], head[y] = idx++;  // 反向边,流量初始为0
  }

  bool bfs() {  // 返回是否找到增广路,若有增广路则建立分层图
    memset(level, -1, so(level));

    qi que;
    que.push(s);
    level[s] = 0;  // 源点的层数为0
    cur[s] = head[s];  // 记录当前弧

    while (que.size()) {
      int u = que.front(); que.pop();
      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (level[v] == -1 && capa[i]) {  // 当前还未被搜过且边还有流量
          level[v] = level[u] + 1;
          cur[v] = head[v];  // 记录当前弧

          if (v == t) return true;  // 能到达汇点,即存在增广路
          que.push(v);
        }
      }
    }
    return false;  // 不能到达汇点,即不存在增广路
  }

  int find(int u, int lim) {  // 从节点u开始增广,流量限制为lim
    if (u == t) return lim;  // 到达汇点

    int flow = 0;  // 总流量
    for (int i = cur[u]; ~i && flow <= lim; i = nxt[i]) {  // 从当前弧开始增广,流量不超过限制
      cur[u] = i;  // 更新当前弧
      int v = edge[i];
      if (level[v] == level[u] + 1 && capa[i]) {  // v在u的下一层,且边还有容量
        ll tmp = find(v, min(capa[i], lim - flow));  // 递归增广节点v
        if (!tmp) level[v] = -1;  // 节点v无法到达汇点,将其删去,可通过将其层数置为-1实现

        capa[i] -= tmp, capa[i ^ 1] += tmp, flow += tmp;
      }
    }
    return flow;
  }

  ll dinic() {
    ll res = 0;
    ll flow;  // 流量
    while (bfs())  // 当前还有增广路
      while (flow = find(s, INF)) res += flow;  // 将所有能增广的路径增广
    return res;
```

```
  }
}
using namespace Dinic_Maximum_Range_Flow;

void solve() {
  memset(head, -1, so(head));

  int S, T; cin >> n >> m >> S >> T;  // S、T为原图的源点、汇点

  s = 0, t = n + 1;  // 新图的源点、汇点
  while (m--) {
    int a, b, c, d; cin >> a >> b >> c >> d;
    add(a, b, d - c);  //
    diff[a] -= c, diff[b] += c;
  }

  ll sum = 0;  // 新图中新加入的源点的出边的流量之和
  for (int i = 1; i <= n; i++) {
    if (diff[i] > 0) add(s, i, diff[i]), sum += diff[i];  // 流量不足,从源点补流量
    else if (diff[i] < 0) add(i, t, -diff[i]);  // 流量多余,向汇点出流量
  }
  add(T, S, INF);  // 连一条从原图的汇点到原图的起点的容量为INF的边,转化为无源汇

  if (dinic() != sum) cout << "No Solution";
  else {
    ll tmp = capa[idx - 1];
    s = S, t = T;
    capa[idx - 1] = capa[idx - 2] = 0;
    cout << tmp + dinic();
  }
}

int main() {
    IOS;
#ifndef ONLINE_JUDGE
    clock_t my_clock = clock();
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
    // ------------------------------------------------------------------
    // init();
    // CaseT  // 单测时注释掉该行
    solve();
    // ------------------------------------------------------------------
#ifndef ONLINE_JUDGE
    cout << endl << "Time used " << clock() - my_clock << " ms." << endl;
#endif
    return 0;
}
```

21.1.3.3 Shoot the Bullet|东方文花帖

原题指路:https://www.luogu.com.cn/problem/P5192

题意

有编号$0\sim(m-1)$的m个少女.接下来n天中,要为$i\ (1\le i\le m)$号少女拍摄g_i张照片.第$k\ (1\le k\le n)$天时有c_k个取材对象,且每个取材对象拍的照片数需在$[l_{k_i},r_{k_i}]$范围内,且第k天的拍照总数不能超过d_k张.问最多能拍多少张照片.

有不超过10组测试数据.每组测试数据第一行输入两个整数$n,m\ (1\le n\le 365,1\le m\le 1000)$.第二行输入$m$个整数$g_1,\cdots,g_m\ (0\le g_i\le 1e5)$.接下来$n$段,每段的第一行输入两个整数$c,d\ (1\le c\le 300,0\le d\le 3e4)$.接下来$c$行每行输入三个整数$t,l,r\ (0\le t<m,0\le l\le r\le 100)$,其中$t$为少女的编号.

思路

代码

```
1
```

21.1.3.4 有源汇上下界最小流

题意

思路

代码

```
1
```

21.1.4 多元汇最大流

21.1.4.1 多元汇最大流

题意

思路

代码

```
1
```

21.1.5 关键边

21.1.5.1 伊基的故事I-道路重建

题意

思路

代码

```
1
```

21.1.6 最大流判定

21.1.6.1 秘密挤奶机

题意

思路

代码

```
1
```

21.1.6.2 星际转移

题意

思路

代码

```
1
```

21.1.7 拆点

21.1.7.1 餐饮

题意

思路

代码

```
1
```

21.1.7.2 最长上升子序列

题意

思路

代码

```
1
```

21.1.7.3 企鹅游行

题意

思路

代码

```
1
```

21.1.8 建图

21.1..8.1 猪

题意

思路

代码

```
1
```

21.2 最小割

21.2.1 模板

21.2.1.1 Dinic/ISAP求最小割

题意

思路

代码

```
1
```

21.2.1.2 网络战阵

题意

思路

代码

```
1
```

21.2.1.3 最优标号

题意

思路

代码

```
1
```

21.2.2 平面图转最短路

21.2.2.1 引水入城

题意

思路

代码

```
1
```

21.2.3 最大权闭合图

21.2.3.1 最大获利

题意

思路

代码

```
1
```

21.2.4 最大密度子图

21.2.4.1 生活的艰辛

题意

思路

代码

```
1
```

21.2.5 最小点权覆盖集

21.2.5.1 有向图破坏

题意

思路

代码

```
1
```

21.2.6 最大点权独立集

21.2.6.1 王者之剑

题意

思路

代码

```
1
```

21.2.7 建图

21.2.7.1 有线电视网络

题意

思路

代码

```
1
```

21.2.7.2 太空飞行计划

题意

思路

代码

```
1
```

21.2.7.3 骑士共存

题意

思路

代码

```
1
```

21.3 费用流

21.3.1 模板

21.3.1.1 费用流

题意

给定一个含n $(2 \leq n \leq 5000)$个点、m $(1 \leq m \leq 5e4)$条边的有向图,点编号$1 \sim n$.给定每条边的容量和费用,边的容量非负,图中可能存在重边或自环,保证费用不出现负环.求点s到点t $(s \neq t)$的最大流,以及流量最大时的最小费用,若无法从s到t,输出0 0.

有向图用m行输入描述,每行包含四个整数u, v, c, w $(0 \leq c \leq 100, -100 \leq w \leq 100)$,表示存在一条从点$u$到点$v$的有向边,容量为$c$,费用为$w$.

代码

```
namespace SPFA_Cost_Flow {
  static const int MAXN = 5005, MAXM = 1e5 + 10;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], cost[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量,cost[i]表示边i的费用
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  int dis[MAXN];  // dis[i]表示源点到节点i的最短路
  int pre[MAXN];  // pre[i]表示节点i的前驱边的编号
  bool state[MAXN];  // SPFA中记录每个节点是否在队列中

  void add(int a, int b, int c, int d) {  // 建边a->b,容量为c,费用为d
    edge[idx] = b, capa[idx] = c, cost[idx] = d, nxt[idx] = head[a], head[a] = idx++;  // 正向边
    edge[idx] = a, capa[idx] = 0, cost[idx] = -d, nxt[idx] = head[b], head[b] = idx++;  // 反向边,流量初始为0,费用为正向边的相反数
  }

  bool spfa() {  // 返回是否找到增广路
    memset(dis, INF, so(dis));
    memset(min_capa, 0, so(min_capa));

    qi que;
    que.push(s);
    dis[s] = 0, min_capa[s] = INF;  // 源点处的流量无限制

    while (que.size()) {
```

```
      int u = que.front(); que.pop();
      state[u] = false;

      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (capa[i] && dis[v] > dis[u] + cost[i]) {  // 边还有容量
          dis[v] = dis[u] + cost[i];
          pre[v] = i;  // 记录前驱边
          min_capa[v] = min(min_capa[u], capa[i]);

          if (!state[v]) {
            que.push(v);
            state[v] = true;
          }
        }
      }
    }
    return min_capa[t];  // 汇点的流量非零即可以到达汇点,亦即存在增广路
  }

  pll EK() {  // first为最大流、second为最小费用
    pll res(0, 0);
    while (spfa()) {  // 当前还有增广路
      int tmp = min_capa[t];
      res.first += tmp, res.second += (ll)tmp * dis[t];
      for (int i = t; i != s; i = edge[pre[i] ^ 1])
        capa[pre[i]] -= tmp, capa[pre[i] ^ 1] += tmp;  // 正向边减,反向边加
    }
    return res;
  }
}
using namespace SPFA_Cost_Flow;

void solve() {
  memset(head, -1, so(head));

  cin >> n >> m >> s >> t;
  while (m--) {
    int a, b, c, d; cin >> a >> b >> c >> d;
    add(a, b, c, d);
  }

  auto ans = EK();
  cout << ans.first << ' ' << ans.second;
}

int main() {
  solve();
}
```

21.3.1.2 运输问题

题意

思路

代码

```
1
```

21.3.1.3 负载平衡问题

题意

思路

代码

```
1
```

21.3.2 二分图最优匹配

21.3.2.1 分配

题意

思路

代码

```
1
```

21.3.3 最大权不相交路径

21.3.3.1 数字梯形

题意

思路

代码

```
1
```

21.3.4 网格图

21.3.4.1 K取方格数

题意

思路

代码

```
1
```

21.3.4.2 深海机器人

题意

思路

代码

```
1
```

21.3.5 拆点

21.3.5.1 餐巾计划

题意

思路

代码

```
1
```

21.3.5.2 Binary Tree on Plane

原题指路:https://codeforces.com/problemset/problem/277/E

题意 (3 s)

在平面上给定n个点作为树的节点,连接这些点构成一棵有向二叉树,使得其边权和最小.节点u能向节点v连一条有向边$u \to v$当且仅当$y_u > y_v$.若有解,输出最小边权和,误差不超过$1\mathrm{e}-6$;否则输出-1.

第一行输入一个整数n $(2 \le n \le 400)$.接下来n行每行输入两个整数x, y $(|x|, |y| \le 1000)$,表示平面上一个点.

思路

将平面上的每个点i $(1 \le i \le n)$拆成一个入点i和一个出点$(i+n)$.

源点s向每个点的入点i连一条容量为2、费用为0的边,表示一个节点至多有2个儿子节点.

每个点的出点$(i+n)$向汇点t连一条容量为1、费用为0的边,表示一个节点至多有1个父亲节点.

对满足$y_i > y_j$的节点i, j,节点i向节点$(j+n)$连一条容量为1、费用为$dis(i,j)$的边,其中$dis(i,j)$表示点i与点j的欧式距离.

问题转化为求该网的MCMF.

代码

```
namespace SPFA_Cost_Flow {
  static const int MAXN = 805, MAXM = 6.4e5 + 5;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量
  double cost[MAXM];  // cost[i]表示边i的费用
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  double dis[MAXN];  // dis[i]表示源点到节点i的最短路
  int pre[MAXN];  // pre[i]表示节点i的前驱边的编号
  bool state[MAXN];  // SPFA中记录每个节点是否在队列中

  void add(int a, int b, int c, double d) {  // 建边a->b,容量为c,费用为d
    edge[idx] = b, capa[idx] = c, cost[idx] = d, nxt[idx] = head[a], head[a] = idx++;  // 正向边
    edge[idx] = a, capa[idx] = 0, cost[idx] = -d, nxt[idx] = head[b], head[b] = idx++;  // 反向边,流量初始为0,费用为正向边的相反数
  }

  bool spfa() {  // 返回是否找到增广路
    memset(dis, 0x42, so(dis));  // 注意double数组的正无穷
    memset(min_capa, 0, so(min_capa));

    qi que;
    que.push(s);
    dis[s] = 0, min_capa[s] = INF;  // 源点处的流量无限制

    while (que.size()) {
      int u = que.front(); que.pop();
```

```
      state[u] = false;

      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (capa[i] && dis[v] > dis[u] + cost[i]) {  // 边还有容量
          dis[v] = dis[u] + cost[i];
          pre[v] = i;  // 记录前驱边
          min_capa[v] = min(min_capa[u], capa[i]);

          if (!state[v]) {
            que.push(v);
            state[v] = true;
          }
        }
      }
    }
    return min_capa[t];  // 汇点的流量非零即可以到达汇点,亦即存在增广路
  }

  pair<ll, double> EK() {  // first为最大流、second为最小费用
    pair<ll, double> res(0, 0);
    while (spfa()) {  // 当前还有增广路
      int tmp = min_capa[t];
      res.first += tmp, res.second += dis[t] * tmp;
      for (int i = t; i != s; i = edge[pre[i] ^ 1])
        capa[pre[i]] -= tmp, capa[pre[i] ^ 1] += tmp;  // 正向边减,反向边加
    }
    return res;
  }
}
using namespace SPFA_Cost_Flow;

#define x first
#define y second

double getDistance(pii a, pii b) {  // 求两点间的距离
  return hypot(a.x - b.x, a.y - b.y);
}

void solve() {
  memset(head, -1, so(head));

  int n; cin >> n;
  s = 0, t = 2 * n + 1;
  vii points(n + 1);
  for (int i = 1; i <= n; i++) {
    cin >> points[i].x >> points[i].y;
    add(s, i, 2, 0);  // 从原点向每个节点的入点连一条容量为2的边,表示每个节点至多有2个儿子节点
    add(i + n, t, 1, 0);  // 从每个节点的出点向汇点连一条容量为1的边,表示每个节点至多有1个父亲节点
  }
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (i != j && points[i].y > points[j].y)
        add(i, j + n, 1, getDistance(points[i], points[j]));
    }
  }

  auto [flow, cost] = EK();
```

```
  if (flow != n - 1) cout << -1;
  else cout << fixed << setprecision(12) << cost;
}

int main() {
  solve();
}
```

21.3.6 上下界可行流

21.3.6.1 志愿者招募

题意

思路

代码

```

```

21.3.7 建图

21.3.7.1 小 A 的卡牌游戏

原题指路:https://codeforces.com/gym/103186/problem/B

题意 (2 s)

一副n张卡的卡组恰包含a张A卡、b张B卡、c张C卡.现给出n次三选一的机会,三张卡分别来自三个种类,玩家需从三张卡中选一张加入自己的卡组,使得卡组强度尽量大.每张卡有一个强度值,卡组的强度是所有卡的强度之和.求卡组强度的最大值.

第一行输入四个整数n, a, b, c $(1 \leq a, b, c \leq n \leq 5000, a + b + c = n)$.接下来$n$行每行输入三个整数描述一个三选一的机会,其中第$i$行输入三个整数$a_i, b_i, c_i$ $(1 \leq a_i, b_i, c_i \leq 1e9)$,分别表示该次选择中A卡、B卡、C卡的强度.

思路

各边容量都为1,费用为卡牌强度的负值,转化为求最小费用流,最终答案为最小费用的负值.

代码

```
namespace SPFA_Cost_Flow {
  static const int MAXN = 5005, MAXM = 1e5 + 10;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], cost[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量,cost[i]表示边i的费用
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  int dis[MAXN];  // dis[i]表示源点到节点i的最短路
  int pre[MAXN];  // pre[i]表示节点i的前驱边的编号
  bool state[MAXN];  // SPFA中记录每个节点是否在队列中
```

```
9
10    void add(int a, int b, int c, int d) {  // 建边a->b,容量为c,费用为d
11      edge[idx] = b, capa[idx] = c, cost[idx] = d, nxt[idx] = head[a], head[a] = idx++;  //
      正向边
12      edge[idx] = a, capa[idx] = 0, cost[idx] = -d, nxt[idx] = head[b], head[b] = idx++;  //
      反向边,流量初始为0,费用为正向边的相反数
13    }
14
15    bool spfa() {  // 返回是否找到增广路
16      memset(dis, INF, so(dis));
17      memset(min_capa, 0, so(min_capa));
18
19      qi que;
20      que.push(s);
21      dis[s] = 0, min_capa[s] = INF;  // 源点处的流量无限制
22
23      while (que.size()) {
24        int u = que.front(); que.pop();
25        state[u] = false;
26
27        for (int i = head[u]; ~i; i = nxt[i]) {
28          int v = edge[i];
29          if (capa[i] && dis[v] > dis[u] + cost[i]) {  // 边还有容量
30            dis[v] = dis[u] + cost[i];
31            pre[v] = i;  // 记录前驱边
32            min_capa[v] = min(min_capa[u], capa[i]);
33
34            if (!state[v]) {
35              que.push(v);
36              state[v] = true;
37            }
38          }
39        }
40      }
41      return min_capa[t];  // 汇点的流量非零即可以到达汇点,亦即存在增广路
42    }
43
44    pll EK() {  // first为最大流、second为最小费用
45      pll res(0, 0);
46      while (spfa()) {  // 当前还有增广路
47        int tmp = min_capa[t];
48        res.first += tmp, res.second += (ll)tmp * dis[t];
49        for (int i = t; i != s; i = edge[pre[i] ^ 1])
50          capa[pre[i]] -= tmp, capa[pre[i] ^ 1] += tmp;  // 正向边减,反向边加
51      }
52      return res;
53    }
54  }
55  using namespace SPFA_Cost_Flow;
56
57  int A, B, C;
58
59  void solve() {
60    memset(head, -1, so(head));
61    s = 0, t = MAXN - 1;  // 超级源点、超级汇点
62
63    cin >> n >> A >> B >> C;
64
```

```
  // 所有汇点向超级汇点连边,容量为每种卡的数量,费用为0
  add(n + 1, t, A, 0), add(n + 2, t, B, 0), add(n + 3, t, C, 0);

  for (int i = 1; i <= n; i++) {
    int a, b, c; cin >> a >> b >> c;
    add(s, i, 1, 0);  // 超级源点向源点连边,容量为1,费用为0
    // 各源点向对应的汇点连边,容量为1,费用为强度的负值
    add(i, n + 1, 1, -a), add(i, n + 2, 1, -b), add(i, n + 3, 1, -c);
  }

  cout << -EK().second;
}

int main() {
  solve();
}
```

21.3.7.2 Direction Setting

原题指路:https://codeforces.com/gym/103117/problem/F

题意

给定一张包含编号$1\sim n$的n个节点和编号$1\sim m$的m条边的无向图,其中节点i $(1\le i\le n)$有一个限制a_i.为每条边规定一个方向,使得该图变为有向图,同时$D=\sum_{i=1}^{n}\max\{0,d_i-a_i\}$最小,其中$d_i$表示节点$i$的入度.

有t组测试数据.每组测试数据第一行输入两个整数n,m $(2\le n\le 300,1\le m\le 300)$.第二行输入$n$个整数$a_1,\cdots,a_n$ $(0\le a_i\le 1e4)$.接下来m行每行输入两个整数u_i,v_i $(1\le u_i,v_i\le n)$,表示第i条边连接节点u与v.注意可能存在重边和自环.数据保证所有测试数据的n之和、m之和不超过3000.

对每组测试数据,第一行输出一个整数,表示D的最小值.第二行输出一个长度为m的$0-1$串$s_1\cdots,s_m$,表示边i $(1\le i\le m)$的方向,其中$s_i=0$表示边i从节点u_i指向v_i,$s_i=1$表示边i从节点v_i指向u_i.若有多组解,输出任一组.

思路

由节点的限制和数据范围易想到网络流,考虑如何建图.因限制在节点上而不在边上,而连接节点u与v的边$<u,v>$可选择为节点u贡献1点入度还是为节点v贡献1点入度,考虑拆边,即将边$<u,v>$变成一个虚节点P,分别向节点u和v连边,容量为1、费用为0.

对每个节点i,显然可向汇点连一条容量为$a[i]$、费用为0的边,它表示$a[i]$是节点i的免费流量.实际流量可超过$a[i]$,此时从节点i向汇点连一条容量为INF、费用为1的边,表示超出$a[i]$的流量需1点花费,最终答案即达到最大流的最小花费.

对输出方案,只需考虑残量网络中容量减为0的边的方向与输入数据中边的方向是否相同即可.

代码

```
namespace SPFA_Cost_Flow {
  static const int MAXN = 3005, MAXM = 3e5 + 10;  // 边开两倍
  int n, m, s, t;  // 点数、边数、源点、汇点
  int head[MAXN], edge[MAXM], capa[MAXM], cost[MAXM], nxt[MAXM], idx;  // capa[i]表示边i的容量,cost[i]表示边i的费用
  int min_capa[MAXN];  // min_capa[i]表示到节点i的所有边的容量的最小值
  int dis[MAXN];  // dis[i]表示源点到节点i的最短路
  int pre[MAXN];  // pre[i]表示节点i的前驱边的编号
  bool state[MAXN];  // SPFA中记录每个节点是否在队列中
```

```
  void add(int a, int b, int c, int d) {  // 建边a->b,容量为c,费用为d
    edge[idx] = b, capa[idx] = c, cost[idx] = d, nxt[idx] = head[a], head[a] = idx++;  // 正向边
    edge[idx] = a, capa[idx] = 0, cost[idx] = -d, nxt[idx] = head[b], head[b] = idx++;  // 反向边,流量初始为0,费用为正向边的相反数
  }

  bool spfa() {  // 返回是否找到增广路
    memset(dis, INF, so(dis));
    memset(min_capa, 0, so(min_capa));

    qi que;
    que.push(s);
    dis[s] = 0, min_capa[s] = INF;  // 源点处的流量无限制

    while (que.size()) {
      int u = que.front(); que.pop();
      state[u] = false;

      for (int i = head[u]; ~i; i = nxt[i]) {
        int v = edge[i];
        if (capa[i] && dis[v] > dis[u] + cost[i]) {  // 边还有容量
          dis[v] = dis[u] + cost[i];
          pre[v] = i;  // 记录前驱边
          min_capa[v] = min(min_capa[u], capa[i]);

          if (!state[v]) {
            que.push(v);
            state[v] = true;
          }
        }
      }
    }
    return min_capa[t];  // 汇点的流量非零即可以到达汇点,亦即存在增广路
  }

  pll EK() {  // first为最大流、second为最小费用
    pll res(0, 0);
    while (spfa()) {  // 当前还有增广路
      int tmp = min_capa[t];
      res.first += tmp, res.second += (ll)tmp * dis[t];
      for (int i = t; i != s; i = edge[pre[i] ^ 1])
        capa[pre[i]] -= tmp, capa[pre[i] ^ 1] += tmp;  // 正向边减,反向边加
    }
    return res;
  }
}
using namespace SPFA_Cost_Flow;

pii edges[MAXM];

void solve() {
  memset(head, -1, so(head));

  cin >> n >> m;

  s = n + m + 1, t = n + m + 2;  // 源点、汇点
```

```
  for (int i = 1; i <= n; i++) {
    int a; cin >> a;
    add(i, t, a, 0), add(i, t, INF, 1);  // 免费流量、额外流量
  }
  for (int i = 1; i <= m; i++) {
    cin >> edges[i].first >> edges[i].second;
    add(s, n + i, 1, 0);  // 拆边的虚节点
    add(n + i, edges[i].first, 1, 0), add(n + i, edges[i].second, 1, 0);  // 虚点向u,v连边
  }

  cout << EK().second << endl;
  for (int u = n + 1; u <= n + m; u++) {  // 枚举拆边的虚节点
    for (int i = head[u]; ~i; i = nxt[i]) {
      int v = edge[i];
      if (v > n + m) continue;  // 不是原图中的节点

      if (!capa[i]) {
        if (v == edges[u - n].second) cout << 0;
        else cout << 1;
        break;
      }
    }
  }
  cout << endl;
}

int main() {
  CaseT  // 单测时注释掉该行
  solve();
}
```