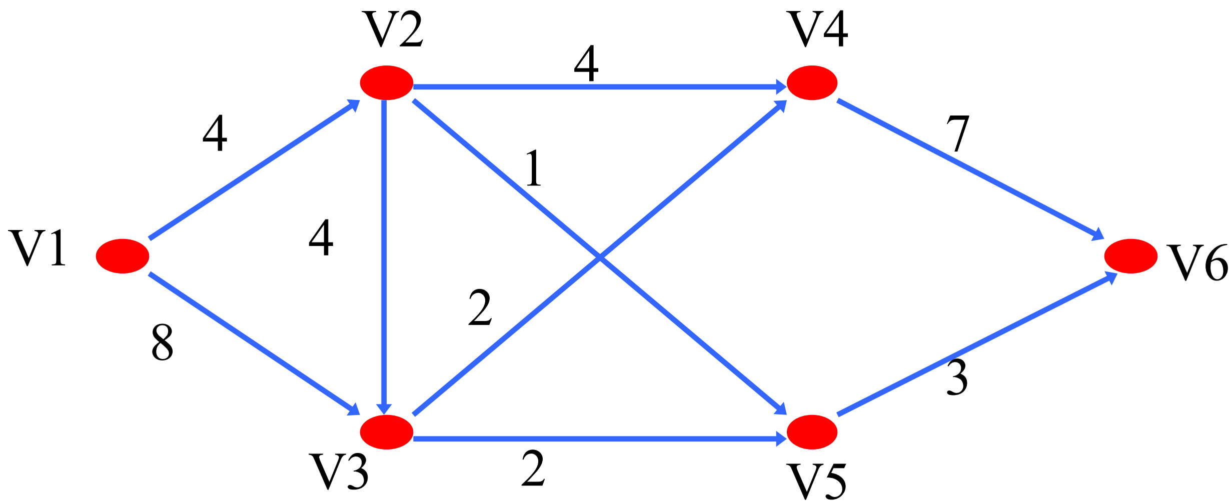
网络流
Network Flow

例0 运输方案

- 下图为连接产品产地V_1和销地V_6的交通网，每一边(V_i,V_j)代表从V_i到V_j的运输线，产品经这条边由V_i输送到V_j，边旁的数字表示这条运输线的最大通行能力（简称容量）。产品经过交通网从V_1输送到V_6，现要求制定一个输送方案，使V_1运到V_6的产品数量最多。

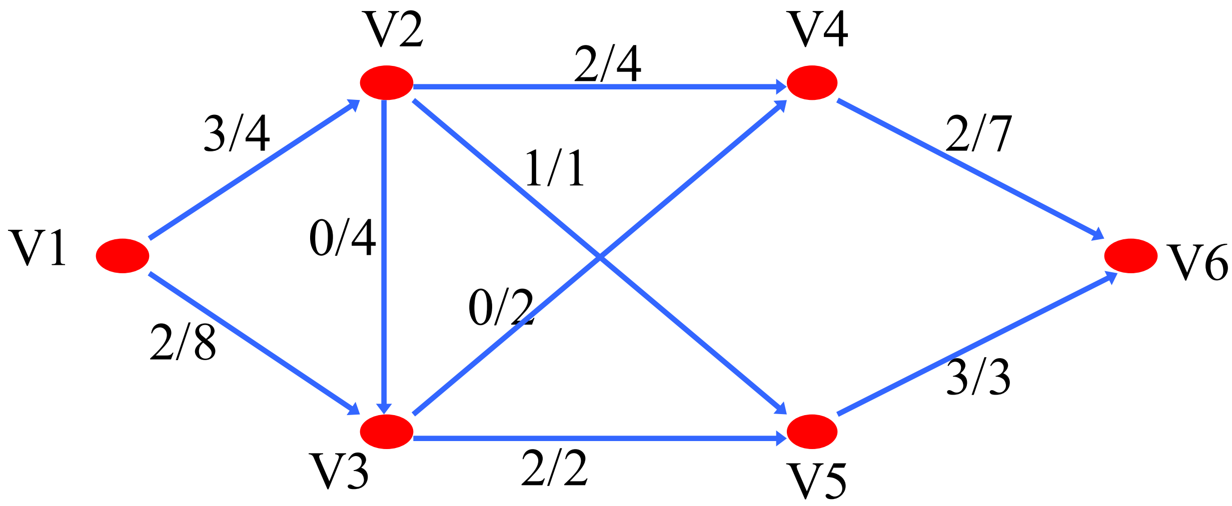


例0　　　　　　运输方案

- 运输方案的可行必须满足以下三个条件：
  - 实际运输量不能是负的；
  - 每条边的实际运输量不能大于该边的容量；
  - 除了起点V_1和终点V_6，对其他顶点（中间点）来说，不能囤积物资，即运到它那儿的物资是多少，从它那儿运走的物资也应该是多少。

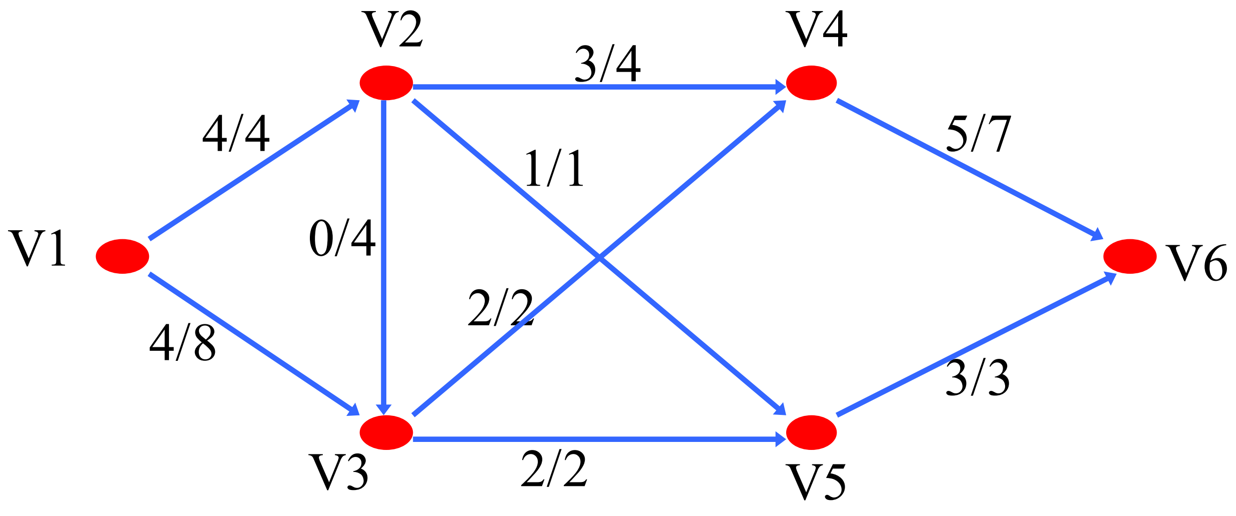
例0 运输方案

- 如下是一个可行的输送方案，边旁的数字为该运输线的实际运输量。
- 该运输方案表示：2（吨）产品沿有向路$P_1(V_1,V_2,V_4,V_6)$运到销地；1（吨）产品沿有向路$P_2(V_1,V_2,V_5,V_6)$运到销地；2（吨）产品沿有向路$P_3(V_1,V_3,V_5,V_6)$运到销地。总共有5（吨）从V_1运到V_6。



例0　　　　　运输方案

- 如下是最优的输送方案，总共能运输8（吨）。
- 该运输方案表示：3（吨）产品沿有向路$P_1(V_1,V_2,V_4,V_6)$运到销地；1（吨）产品沿有向路$P_2(V_1,V_2,V_5,V_6)$运到销地；2（吨）产品沿有向路$P_3(V_1,V_3,V_5,V_6)$运到销地； 2（吨）产品沿有向路$P_4(V_1,V_3,V_4,V_6)$运到销地。



网络流的定义

- **有向图**$G = (V, E)$中：
  - 有唯一的一个源点S（产地，出发点）
  - 有唯一的一个汇点T（销地，结束点）
  - 图中每条边(u, v)都有一非负容量$C_{u,v}$
- 满足上述条件的图G称为网络流图（又称网络），记为$G = (V, E, C)$。

网络的流

- 对于网络流图$G=(V,E,C)$的每一条边(u,v)，给定一个数$f_{u,v}$，且满足下列条件：
  - $0 \le f_{u,v} \le C_{u,v}$，运量不能超过容量。
  - 除源点和汇点外，其余顶点v恒有$\sum f_{i,v}=\sum f_{v,i}$，进货等于出货，不能囤货。
- 这时$G=(V,E,C)$中的流称为G的**可行流**f。
- 为了方便起见和出于**对称性**，我们认为$f_{u,v}=-f_{v,u}$。

网络的流

- 当所有$f_{u,v}=0$，称f为零流，零流一定是可行流。
- 对于源S和汇T有$\sum f_{S,i}-\sum f_{i,S}=\sum f_{i,T}-\sum f_{T,i}=\omega$，$\omega$叫做网络流的流量。
- 当$f_{u,v}=C_{u,v}$时，称边(u,v)饱和。
- ω达到最大值时的流f，称为$G=(V,E,C)$的**最大流**。

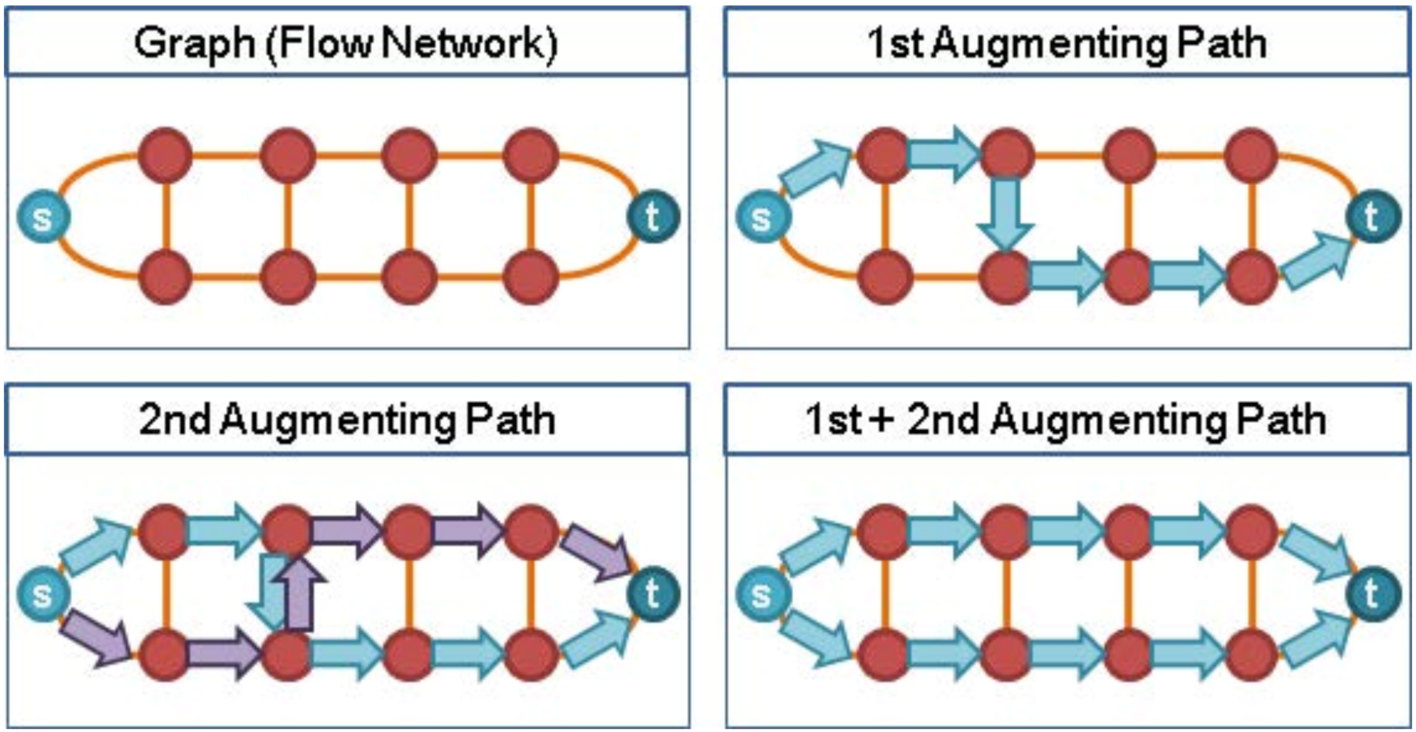
求解最大流

- 在例0中，我们把运输方案划分成了多条路线，这告诉我们流可以拆成多个小的流，我们将利用这个性质求解最经典的问题——最大流。
- 只要不断地在现有的流网络上累加小的流，当不能累加的时候就得到了最大流。

求解最大流

- 如果现有的流并不是最大流的一部分，是否还能累加形成最大流呢？
- 实际上，这是保证可行的！由于流具有方向性，并且相反方向的流互相抵消，我们可以发现，通过不同的流之间的叠加，我们可以通过回溯之前流的部分路段，从而修正之前的错误选择。
- 之前的**对称性**得到了应用！
- 接下来会给出详细证明。

求解最大流

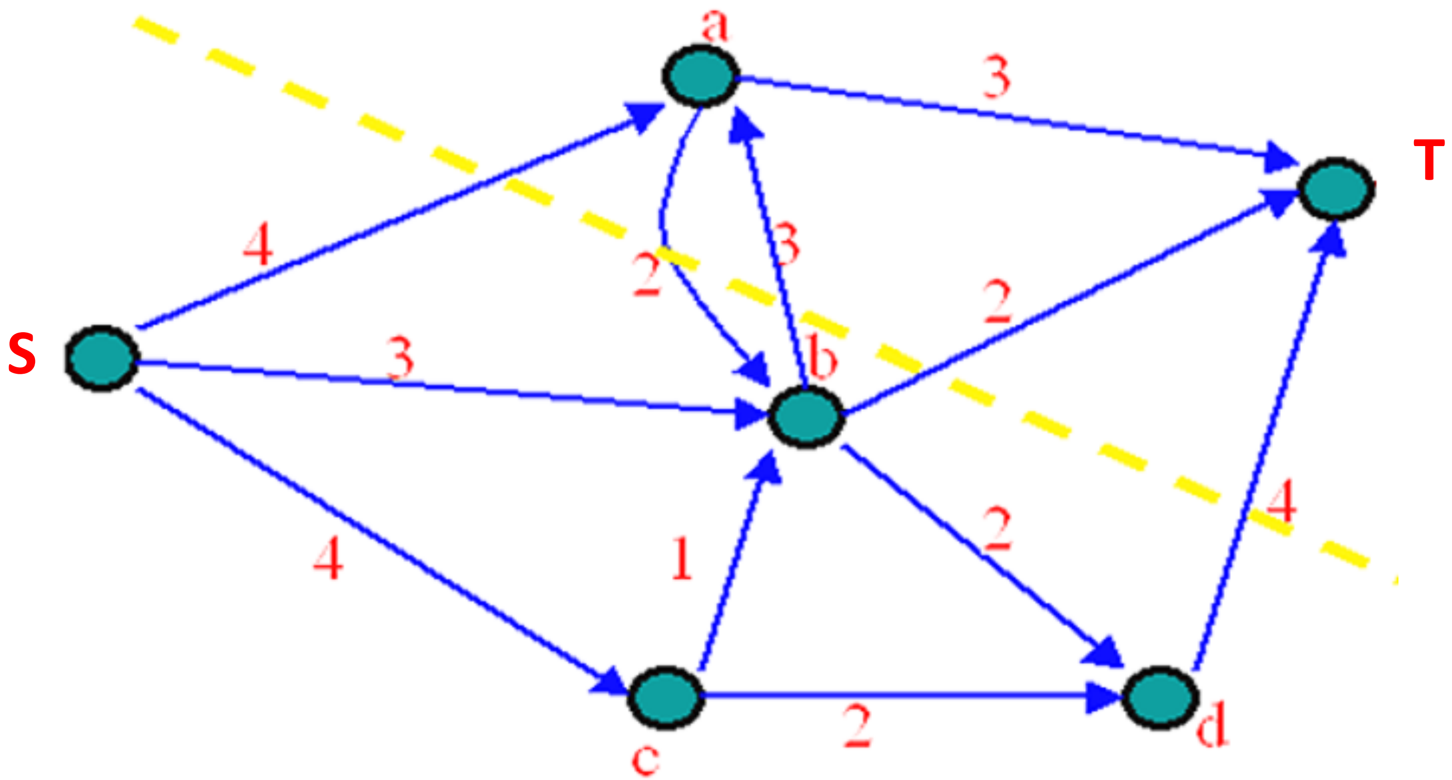


Graph (Flow Network)
s
t
1st Augmenting Path
s
t
2nd Augmenting Path
s
t
1st + 2nd Augmenting Path
s
t

切割

- 切割的定义
  - 设$G=(V,E,C)$是已知的网络流图，假定s是V的一个子集，s'为s的补集，这样把顶点集V分成s和s'两个部分，满足$S\in s,T\in s'$。
  - 对于一个端点在s、另一个端点在s'的所有**边的集合**，叫做网络流图G的一个切割，用(s,s')表示。下页图用虚线划去的边表示一个切割，其中$s=\{S,b,c,d\},s'=\{a,T\}$。

切割



切割

- 切割容量$C(s, s')$：在切割(s, s')中，把所有边容量和叫做这个切割的容量。

$$C(s, s') = \sum_{u \in s, v \in s'} C(u, v)$$

- 净流量 $f(s, s')$：横跨切割 (s, s') 流量的代数和。
  - 净流量的值=当前的流量ω。

$$f(s, s') = \sum_{u \in s, v \in s'} f(u, v) - \sum_{u \in s, v \in s'} f(v, u)$$

最大流-最小割定理

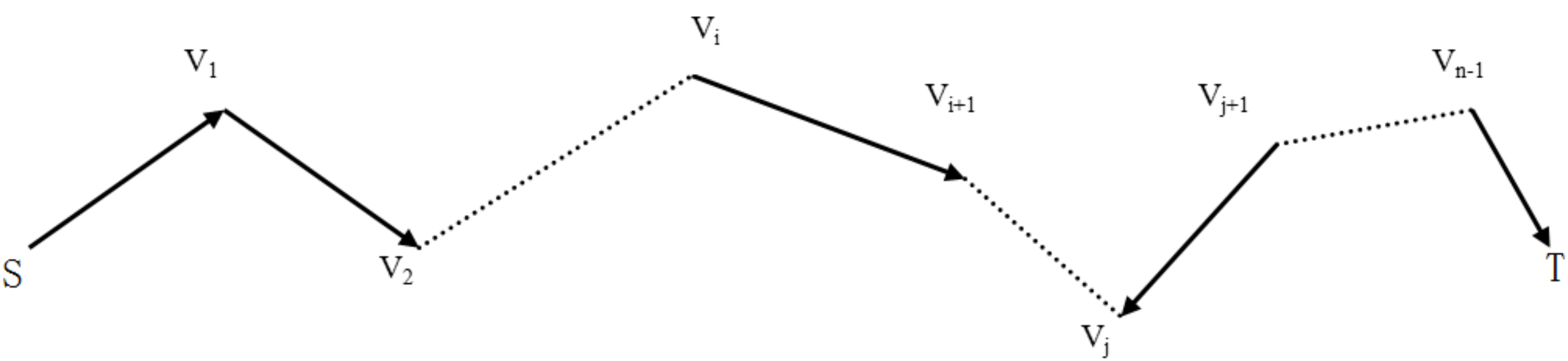
- 引理
  - 对于已知的网络流图，从源点S到汇点T的流量ω的最大值小于等于任何一个切割的容量，即$\omega_{\max} \le C(s,s')_{\min}$。
- 最大流-最小割定理（Ford-Fulkerson定理）
  - 在一个给定的网络流图上，流的最大值等于切割容量的最小值，即$\omega_{\max} = C(s,s')_{\min}$。

最大流-最小割定理的证明

- 在网络流图$G = (V, E, C)$中，定义从源点S到汇点T的道路：设$(V_0 = S, V_1, V_2, \cdots, V_n = T)$为$G$上的顶点序列，对于$i = 0,1, \ldots, n-1$，都有$(V_i, V_{i+1})$或$(V_{i+1}, V_i)$属于$E$，则称$(V_0, V_1, V_2, \cdots, V_n)$是一条从$S$到$T$的道路。
- 由于G是**有向图**，道路上边的方向与道路方向一致的边称为前向边P^+，反之称为后向边P^-。
- 道路上边的**饱和**：前向边若$f_{u,v} = C_{u,v}$，后向边若$f_{u,v} = 0$（**对称性**），则称边(u, v)为关于该道路上的饱和边。
- 若从S到T的道路上所有的边**均不饱和**，即对于P^+有$f_{u,v} < C_{u,v}$，P^-有$f_{u,v} > 0$，则称这条路为**可增广道路**。

最大流-最小割定理的证明

- 修改可增广道路上每条边的流量，同时保持网络流的可行性，达到流量的增加，其增量的确定方法如下：
- 令$\delta(u,v)=\begin{cases}C_{u,v}-f_{u,v}, & P^+\\ f_{u,v}, & P^-\end{cases}$，取$\delta=\min\{\delta(u,v)\}$为增量。
- 然后对可增广道路的每一条前向边流量增加δ，每一条后向边流量减少δ，从而使得整个网络流的流量获得增加。



最大流-最小割定理的证明

- 设网络流图G的流量f达到最大，我们构造一个点的集合s如下：
  - $S \in s$
  - 若$x \in s$，且$f_{x,y} < C_{x,y}$，则$y \in s$
  - 若$x \in S$，且$f_{y,x} > 0$，则$y \in s$
- 由此可见s就是从S出发、有空余流量的边所关联的点集，因此$T \notin s$，否则S到T不饱和，f不是最大流。
- 设s'是s的补集，那么$T \in s'$，我们构造出了一个切割(s, s')。
- 按照s的定义，若$x \in s, y \in s'$，则$f_{x,y} = C_{x,y}, f_{y,x} = 0$。
- 所以$\omega_{\max} = \sum(f_{x,y} - f_{y,x}) = \sum f_{x,y} = C(s, s')$。再结合引理，可知我们证明了最大流-最小割定理。 ■

求解最大流

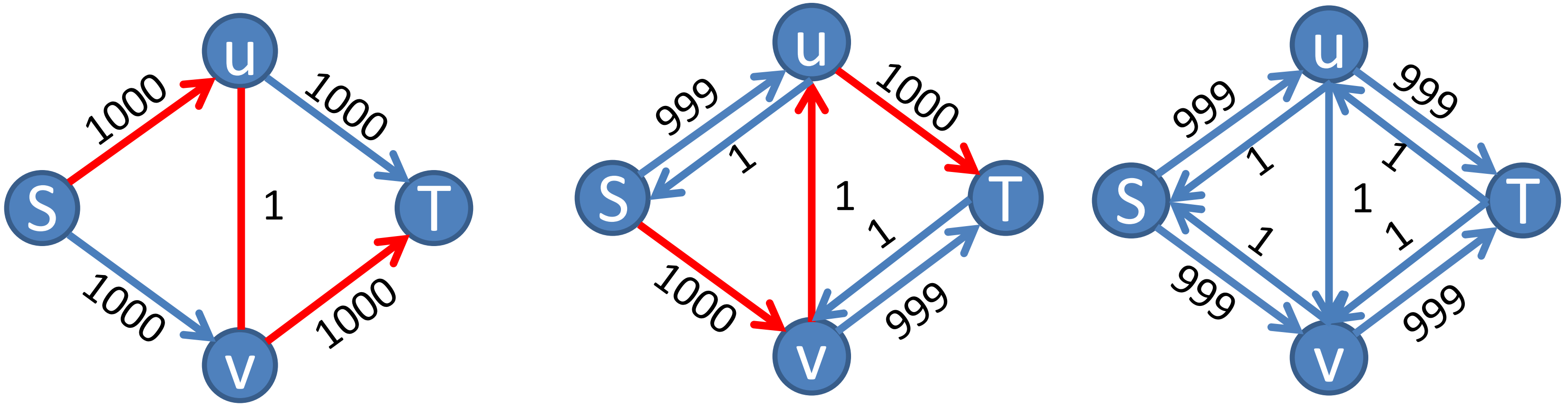
- 同时我们也证明了之前算法的正确性。
- 每次新增一条流的时候，我们都要满足每条边的容量限制。即当前这条边的流量加上新的流量，不能超过边的（**剩余**）容量限制。
- 一个便捷的处理方式是：记录这条边还能容纳的流量，即边的容量加上能够被回溯（对称边）的流量，称为“残存容量”。将所有的残存容量看成一个残存网络，也就是一个新的网络流问题，我们在新的图上继续寻找**不饱和路径**。

求解最大流

- 这就是Ford-Fulkerson方法，前面的定理也是他提出的。
- 我们所需要做的就是记录每条边和它的**反向边**（对称性）以及每条边的剩余容量，写一个dfs每次寻找不饱和路径并修改容量，找不到时退出。此时我们得到了最大流。
- 使用最普通的深度优先搜索寻找增广路，每次时间复杂度为$O(E)$。设最大流的流量为ω，由于每次流量至少增加1，时间复杂度为 $O(E\omega)$ 。
- 实际上，对于绝大多数的网络，该算法的运行时间往往远远小于它的实际复杂度。

求解最大流

- Ford-Fulkerson方法每次只增广一条路径，缺点也显而易见，如下图所示。当最大流量ω较大时，寻找的路径不对很可能使运行时间变长，例如反复增广$(S,u,v,T),(S,v,u,T)$。



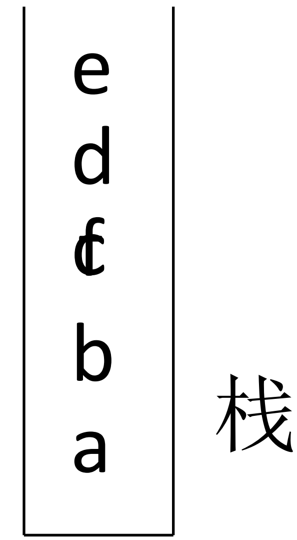
求解最大流

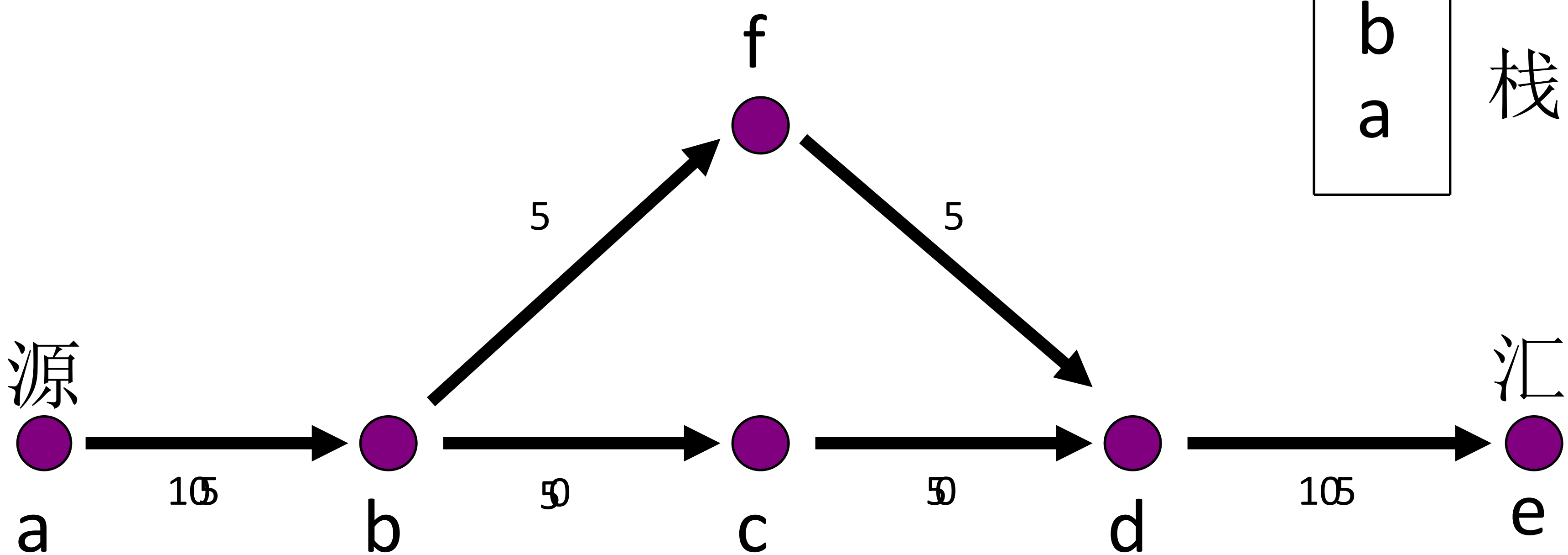
- Edmonds-Karp算法如下：
- 我们可以通过**广度优先搜索**改善Ford-Fulkerson方法，每次寻找残存网络上S到T的**最短路径**（每条边的长度为1）。
- 可以证明，该算法的时间复杂度是 $O(VE^2)$ 。

求解最大流

- 为了加快算法的运行时间，我们需要一次增广多条路径，也就是Dinic算法。
- 每次对**残存网络**bfs标记每个点距S的深度（注意只有还有容量剩余的边才会出现在残量网络上），dfs的时候只对与当前点有边且深度大1的点继续增广。

Dinic算法演示





后退到原路径中从源点能够到达的最远点（dfs过程）

求解最大流

- 我们需要记录当前顶点u，以及到目前为止剩余能分配的容量$limit$，具体实现见下一页。

求解最大流

- **function** dinic(*u, limit*)
  - **if** u为终点T **return** *limit*
  - *used* ← 0
  - **for all** u能连接到未被使用且深度大1的边edge(u,v,w) **do**
    - *flow* ← *dinic*(*v,* min(*w, limit - used*))
    - *w* ← *w - flow*
    - *edge*的反向边*(w)* ← *edge*的反向边*(w)* + *flow*
    - *used* ← *used* + *flow*
    - **if** *used* = *limit* 退出
    - 标记*edge*为已用，在这次dfs 过程中不再访问。
  - **end for**
  - **return** *used*
- **end function**

求解最大流

- 每次更新完之后，残量网络上所有源点到汇点的最短路径都被阻塞。所以，最多只要V次更新就可以找到最大流。
- 每次更新的复杂度为$O(VE)$，总复杂度为$O(V^2E)$。实际更快。

例1　　最长递增子序列问题

- 给定正整数序列$x_1, x_2, \cdots, x_n (n \le 500)$。
  - 计算其最长递增子序列的长度s。
  - 问从给定的序列中最多可取出多少个（互不共用同一个元素的）长度为s的递增子序列。
- 来源：网络流与线性规划24题。

例1　　最长递增子序列问题

- 先dp求出$f[i]$，表示以x_i为结尾的最长递增子序列长度。
- 每个数只能被取一次，在网络流意义下启示我们把一个点i拆成i_1和i_2，中间连一条容量为1的边。
- 把递增子序列转化为网络流图上的路径，那么S向$f[i] = 1$的i_1连边；$f[i] = s$的i_2向T连边；$i < j$，$x_i < x_j$且$f[i] + 1 = f[j]$的i_2向j_1连边。容量均为1。

例2 星际转移问题

- 现有$-1,0,1,\cdots,n$共$n+2$个太空站，m艘太空船在其间来回穿梭，共k个人需要进行星际转移。每个太空站可容纳无限多的人，而每艘太空船i只可容纳$H[i]$个人。每艘太空船将周期性地停靠一系列的太空站。
- 例如：(1，3，4)表示该太空船将周期性地停靠太空站134134134…每一艘太空船从一个太空站驶往任一太空站耗时均为1。人们只能在太空船停靠太空站时上、下船，上下船不需要花费时间。
- 初始时所有人全在0号点上，太空船全在初始站，问最少要多少时间把所有人从0号送到-1号太空站。
- $n \le 20, m \le 13, k \le 50$。
- 来源：网络流与线性规划24题。

例2　星际转移问题

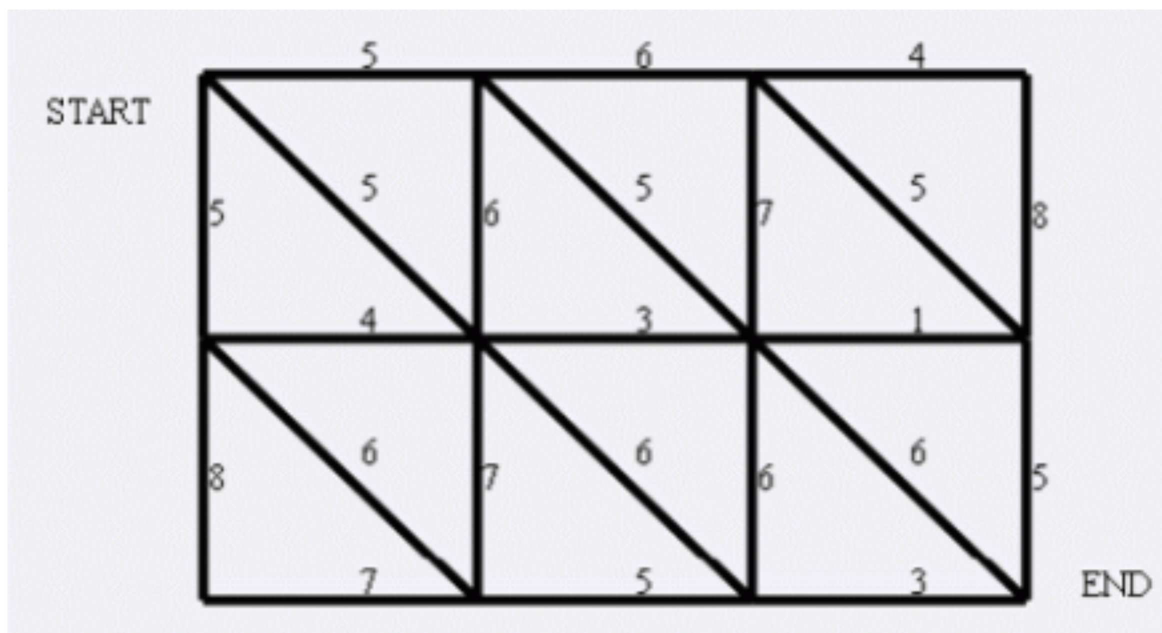
- 追踪人的行动：乘坐太空船经过一系列太空站可以看作先下船再上船，形成一条路径。
- 如果已知天数，只要判断限定天数内是否能够让所有人到终点站就可以了。
- 再次采用拆点的思想：对于同一个空间站每个时间对应一个点。
- 枚举天数，由于只需要添加点和边，利用增广路的性质，直接在残量网络上添加就可以了。

例2　　　　星际转移问题

- 假设当前需要判定时间t的可行性。对于空间站i，拆成点$i_1, i_2, \cdots, i_t$。对于线路(a, b, c)，a_1向连边b_2，　b_2向c_3连边，　c_3向a_4连边，向连边……容量是H。
- 由于一个人也可以在空间站上等下一艘船，因此i_k要向i_{k+1}连容量为无限大的边。

例3　Catch the Thieves/狼抓兔子

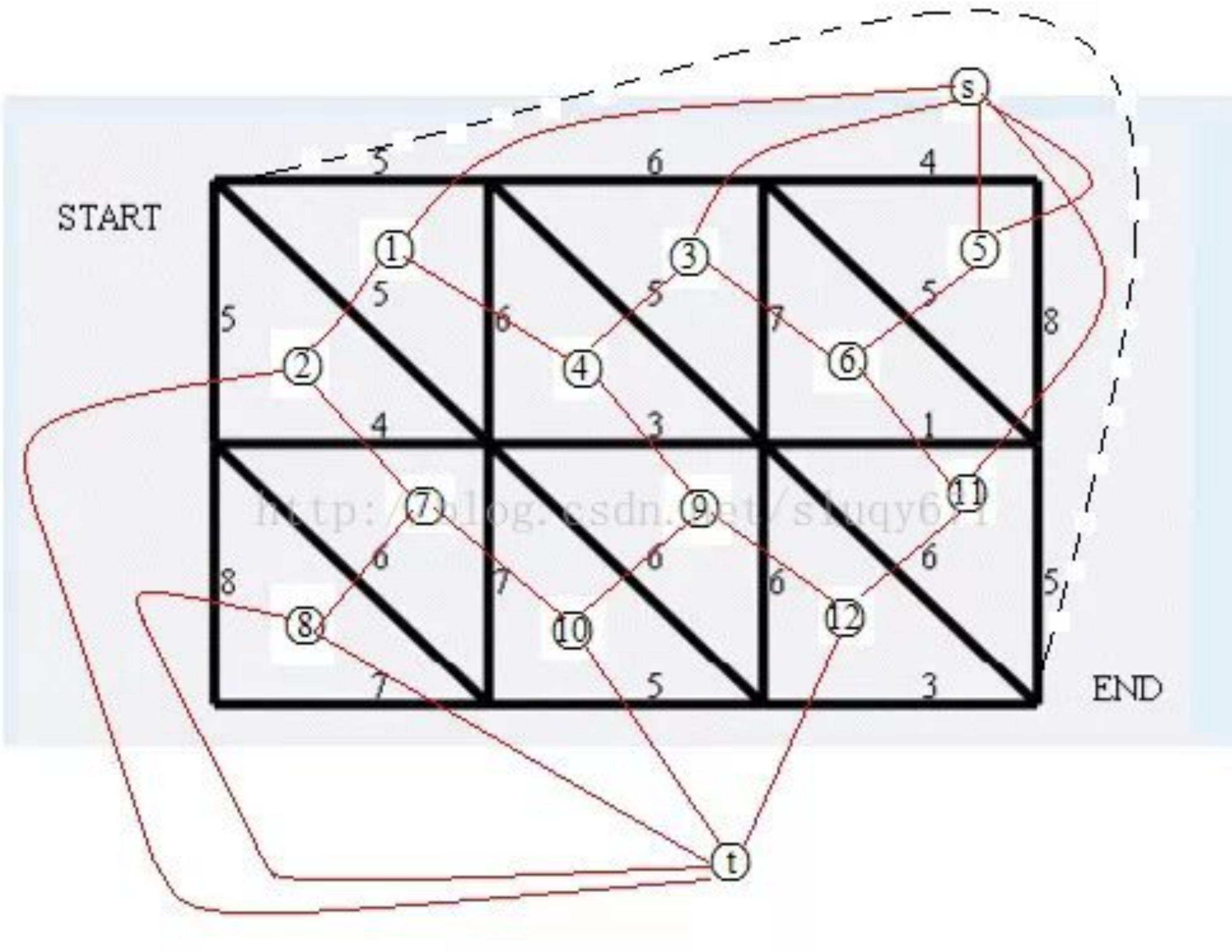
- 给定一个$n \times m$的网格图（还存在主对角线），每条边有一个容量，求$(1,1)$到(n,m)的最大流/最小割。
- $n, m \leq 1000$。
- 来源：BZOJ。



例3 Catch the Thieves/狼抓兔子

- 两道题是分别从两个方面问的问题，一般最大流转最小割很少见。
- 直接跑最大流是不能通过全部测试点的，考虑转化成最小割。
- 割边的代价越小越好，我们形象地把它转化成“割开”，割开一条边的代价就是这条边的容量，割开的方法是在两侧区域内新建两个点，连上一条边。
- 原图的割被转换成新图的一条最短路。

例3 Catch the Thieves/狼抓兔子



二分图匹配

- 二分图
  - 如果图$G = (V, E)$，存在一种对V的划分(s, s')（和切割不一样），使得对于所有的边(a, b)，a和b不属于同一个集合，那么G是二分图。
  - 经典的二分图：棋盘格，坐标，奇偶，男女，树。
  - 另一种描述：不存在奇环的图是二分图。

二分图匹配

- 匹配
  - 图$G = (V, E)$，选取边集$e \subseteq E$，使得每个点与0或1条边连接，那么e被称作G的一个匹配。
- 二分图匹配有一些专门的算法，如匈牙利算法，网络流能不能解决呢？
- 答案是肯定的。

二分图匹配

- 不妨设原图的点集为黑和白。构建网络流图G：
  - S向所有黑点连容量为1的边
  - 所有白点向T连容量为1的边
  - 原图所有的边都保留，容量为1
- 此时S到T的最大流就是最大匹配，完美地利用了容量的限制。

例4　有向无环图最小路径覆盖

- 有向无环图中，使用最少数量的路径覆盖整张图上的所有点。每个点只能被路径经过一次。
- 来源：经典问题。

例4　有向无环图最小路径覆盖

- 路径：点的集合，除最后一个点外，每个点都能连接到下一个点。
- 每个点只属于一条路径。怎样转化？
- 拆点思想：将一个点拆成入点和出点两个点。
- 一个出点匹配一个入点，表示路径上的一条边。
- 剩下来未匹配的就是路径头/尾。

二分图匹配

- 二分图匹配问题通常要输出方案，从左点集指向右点集的流满的边就是匹配方案。

例5　　　　　魔术球问题

- 有n根柱子，现要按下述规则在这n根柱子中依次放入编号为1，2，3，……的球。
  - 每次只能在某根柱子的最上面放球。
  - 在同一根柱子中，任何2个相邻球的编号之和为完全平方数。
- 试设计一个算法，计算出在n根柱子上最多能放多少个球。例如，在4根柱子上最多可放11个球。**输出方案。**
- $n \leq 55$。
- 来源：网络流与线性规划24题。

例5 魔术球问题

- 考虑把柱子转化成路径。
- 枚举答案A，在图中建立节点$1,2,\cdots,A$。如果对于$i<j$有$i+j$为一个完全平方数，连接一条有向边(i,j)。该图是有向无环图，求最小路径覆盖。
- 如果满足最小路径覆盖数小于等于等于n，那么A是一个可行解，在所有可行解中找到最大的A，即为最优解。
- 具体方法可以顺序枚举A的值，当最小路径覆盖数刚好大于n时终止，$A-1$就是最优解。
- 之所以不二分答案是因为直接在残量网络上做复杂度低。

例6 圆桌问题

- 有来自n个不同单位的代表参加一次国际会议。每个单位的代表数为r_i。会议餐厅共有m张餐桌，每张餐桌可容纳c_i个代表就餐。
- 为了使代表们充分交流，希望从同一个单位来的代表不在同一个餐桌就餐。试设计一个算法，**给出满足要求的代表就餐方案。**
- $n \le 270, m \le 150$。
- 来源：网络流与线性规划24题。

例6 圆桌问题

- 建立二分图：
  - S向每个单位顶点连接一条容量为该单位人数的有向边
  - 每个餐桌顶点向T连接一条容量为该餐桌容量的有向边
  - 每个单位顶点向每个餐桌顶点连一条容量为1的有向边
- 求网络最大流，如果最大流量等于所有单位人数之和，则存在解，否则无解。
- 其实本题可以贪心解决。

例7 矩阵游戏

- 给一个$n \times n$的黑白矩阵，你可以交换任意两行或交换任意两列，问通过一系列交换后，能否使主对角线$(1,1)-(n,n)$上全是黑色。
- $n \leq 200$。
- 来源：BZOJ。

例7 矩阵游戏

- 经过观察发现，同行（同列）的黑点在经过交换后仍然同行（同列）。那么问题就变成寻找n个不同行不同列的黑点。
- 按行列建二分图，对于每个黑点，它所在行和列连边，如果最大匹配是n就有解。

例8　　方格取数问题

- 在一个有$m \times n$个方格的棋盘中，每个方格中有一个正整数。现要从方格中取数，使任意2个数所在方格没有公共边，且取出的数的总和最大。试设计一个满足要求的取数算法。
- $n, m \le 30$。
- 来源：网络流与线性规划24题。

例8　　方格取数问题

- 这是一个二分图最大点权独立集问题，就是找出图中一些点，使得这些点之间没有边相连，这些点的权值之和最大。独立集与覆盖集是互补的，求最大点权独立集可以转化为求最小点权覆盖集。最小点权覆盖集问题可以转化为最小割问题解决。
- 结论：最大点权独立集=所有点权-最小点权覆盖集=所有点权-最小割集=所有点权-网络最大流。

例8 方格取数问题

- 首先把棋盘黑白染色，使相邻格子颜色不同，所有黑色格子看做二分图X集合中顶点，白色格子看做Y集合顶点，建立源S汇T。
  - 从S向X集合中每个顶点连接一条容量为格子中数值的有向边。
  - 从Y集合中每个顶点向T连接一条容量为格子中数值的有向边。
  - 相邻黑白格子X_i, Y_j之间从X_i向Y_j连接一条容量为无穷大的有向边。
- 求出网络最大流，要求的结果就是所有格子中数值之和减去最大流量。

最小费用最大流

- 有时，一条边上不仅有容量限制，对单位流量还需要收取一定的费用，每条边费用可能不同。
- 在**流量最大**的前提下**费用最少**，这就是最小费用最大流问题。
- 记为$G=(V,E,C,W)$，W为每条边的单位流量费用，在ω最大的情况下，最小化$\sum f_{u,v}w_{u,v}$。

求解最小费用最大流

- 每次把残量网络上每条边的**费用**看做距离，求S到T的最短路，把（其中一条）最短路径流满，计算费用，直到S和T不连通。
- 由最大流-最小割定理，此时得出的一定是最大流，由于每次都是找费用最小的路径增广，最后得出的一定是最小费用最大流。
- 由于会出现**负权边**，要使用SPFA算法求最短路。

例9　　餐巾计划问题

- 一个餐厅在 N 天内需要用餐巾，第 i 天需要 r_i 块餐巾。餐厅可以购买新的餐巾，每块餐巾的费用为p；或者把旧餐巾送到快洗部，洗一块需m天，其费用为f；或者送到慢洗部，洗一块需$n(n > m)$天，其费用为$s(s < f)$。
- 每天结束时，餐厅必须决定将多少块脏的餐巾送到快洗部，多少块餐巾送到慢洗部，以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和，要满足当天的需求量。最小化总的花费。
- $N \leq 800, r_i \leq 400$。
- 来源：网络流与线性规划24题。

例9　　　餐巾计划问题

- 这个问题的主要约束条件是每天的餐巾够用，而餐巾的来源可能是最新购买，也可能是前几天送洗，今天刚刚洗好的餐巾。每天用完的餐巾可以选择送到快洗部或慢洗部，或者留到下一天再处理。
- 经过分析可以把每天要用的和用完的分离开处理，建模后就是二分图。二分图X集合中顶点X_i表示第i天用完的餐巾，从S向X_i连接容量为r_i的边作为限制。Y集合中每个点Y_i则是第i天需要的餐巾，数量为r_i，与T连接。每天用完的餐巾可以选择留到下一天洗（X_i连X_{i+1}），不需要花费；送到快洗部（X_i连Y_{i+m}），费用为f，送到慢洗部（X_i连Y_{i+n}），费用为s。每天需要的餐巾除了刚刚洗好的餐巾，还可能是新购买的（S连Y_i），费用为p。