1. 图论 Graph Theory

1.1. 定义与术语 Definition and Glossary

1.1.1. 图与网络 Graph and Network

二元组(V,E)称为**图**(graph)。V为**结点**(node)或**顶点**(vertex)集。E为V中结点之间的边的集合。

点对(u,v)称为**边**(edge)或称**弧**(arc)，其中$u,v\in V$，称u,v是**相邻**的(adjacent)，称 u,v 与边(u,v)相**关联**(incident)或相邻。

若边的点对(u,v)有序则称为**有向**(directed)边，其中 u 称为**头**(head)，v 称为**尾**(tail)。所形成的图称**有向图**(directed graph)。为对于 u 来说(u,v)是**出边**(outgoing arc)；对于 v 来说(u,v)是**入边**(incoming arc)。反之，若边的点对无序则称为**无向**(undirected)边，所形成的图称**无向图**(undirected graph)。

若图的边有一个**权值**(weight)，则称为赋权边，所形成的图称**赋权图**(weighted graph)或**网络**(network)。用三元组 G(V,E,W)表示网络。其中 W 表示权集，它的元素与边集 E 一一对应。

满足$|E|<|V|\log|V|$的图，称为稀疏(sparse)图；反之，称为稠密(dense)图。

1.1.2. 图的术语 Glossary of Graph

阶(order)：图 G 中顶点集 V 的大小称作图 G 的阶。
环(loop)：若一条边的两个顶点为同一顶点，则此边称作环。
简单图(simple graph)：没有环、且没有多重弧的图称作简单图。
定向图：对无向图 G 的每条无向边指定一个方向得到的有向图。
底图：把一个有向图的每一条有向边的方向都去掉得到的无向图。
逆图：把一个有向图的每条边都反向由此得到的有向图。
竞赛图(tournament)：有向图的底图是无向完全图，则此有向图是竞赛图。

邻域(neighborhood)：在图中与 u 相邻的点的集合$\{v\,|\,v\in V,(u,v)\in E\}$，称为 u 的邻域，记为$N(u)$。

度：
度(degree)：一个顶点的度是指与该边相关联的边的条数，顶点 v 的度记作 deg(v)。握手

定理：无向图：$\sum_{v\in V}\deg(v)=2|E|$；有向图：$\sum_{v\in V}\deg^+(v)=\sum_{v\in V}\deg^-(v)$。

入度(indegree)：在有向图中，一个顶点 v 的入度是指与该边相关联的入边(即边的尾是 v)的条数，记作 $\deg^+(v)$。

出度(outdegree)：在有向图中，一个顶点的出度是指与该边相关联的出边(即边的头是 v)的条数，记作 $\deg^-(v)$。

孤立点(isolated vertex)：度为 0 的点。**叶**(leaf)：度为 1 的点。

源(source)：有向图中，$\deg^+(v)=0$ 的点。**汇**(sink)：有向图中，$\deg^-(v)=0$ 的点。

奇点(odd vertex)：度为奇数的点。**偶点**(even vertex)：度为偶数的点。

子图：

子图(sub-graph)：G'称作图 G 的子图如果 $V(G')\subseteq V(G)$ 以及 $E(G')\subseteq E(G)$。

生成子图(spanning sub-graph)：即包含 G 的所有顶点的连通子图，即满足条件 $V(G')=V(G)$ 的 G 的子图 G'。

生成树(spanning tree)：设 T 是图 G 的一个子图，如果 T 是一棵树，且 $V(T)=V(G)$，则称 T 是 G 的一个生成树。即 G 的生成子图，且子图为树。

点导出子图(induced subgraph)：设 $V'\subseteq V(G)$，以 V' 为顶点集，以两端点均在 V' 中的边的全体为边集所组成的子图，称为 G 的由顶点集 V' 导出的子图，简称为 G 的点导出子图，记为 $G[V']$。

边导出子图(edge-induced subgraph)：设 $E'\subseteq E(G)$，以 E' 为顶点集，以两端点均在 E' 中的边的全体为边集所组成的子图，称为 G 的由边集 E' 导出的子图，简称为 G 的边导出子图，记为 $G[E']$。

图的补图(complement)：设 G 是一个图，以 $V(G)$ 为顶点集，以 $\{(u,v)|(u,v)\notin E(G)\}$ 为边集的图称为 G 的补图，记为 $\overline{G}$。

点集的补集：记 $\overline{V'}=V-V'$ 为点集 V' 的补集。

特殊的图：

零图(null graph)：$E=\varnothing$，即只有孤立点的图。n 阶零图记为 N_n。

平凡图(trivial graph)：一阶零图。

空图(empty graph)：$V=E=\varnothing$ 的图。

有向无环图(directed acyclic graph(DAG))：有向的无环的图。

完全图(complete graph)：完全图是指每一对不同顶点间都有边相连的的无向图，n 阶完全图常记作 K_n。

二分图（bipartite graph）：若图 G 的顶点集可划分为两个非空子集 X 和 Y，即 $V = X \cup Y$ 且 $X \cap Y = \varnothing$，且每一条边都有一个顶点在 X 中，而另一个顶点在 Y 中，那么这样的图称作二分图。

完全二分图(complete bipartite graph)：二分图 G 中若任意两个 X 和 Y 中的顶点都有边相连，则这样的图称作完全二分图。若 $|X| = m, |Y| = n$，则完全二分图 G 记作 $K_{m,n}$。

正则图(regular graph)：如果图中所有顶点的度皆相等，则此图称为正则图。

1.1.3. 路径与回路 Path and Cycle

途径(walk)：图 G 中一个点边交替出现的序列 $p = v_{i_0} e_{i_1} v_{i_1} e_{i_2} \cdots e_{i_k} v_{i_k}$，满足 $v_{i_j} \in V, e_{i_j} \in E$，$e_{i_j} = (v_{i_{j-1}}, v_{i_j})$。

迹(trail)：边不重复的途径。

路(path)：顶点不重复的迹。

简单图中的路可以完全用顶点来表示，$P = v_{i_0} v_{i_1} \cdots v_{i_k}$。

若 $p_1 = p_m$，称**闭**的(closed)；反之，称为**开**的(open)。

闭途径(closed walk)：起点和终点相同的途径。

闭迹(closed trail)：起点和终点相同的迹，也称为**回路**(circuit)。

圈(cycle)：起点和终点相同的路。

途径（闭途径）、迹（闭迹）、路（圈）上所含的边的个数称为它的**长度**(length)。
简单图 G 中长度为奇数和偶数的圈分别称为**奇圈**(odd cycle)和**偶圈**(even cycle)。

对任意 $u, v \in V(G)$，从 x 到 y 的具有最小长度的路称为 x 到 y 的**最短路**(shortest path)，其长度称为 x 到 y 的**距离**(distance)，记为 $d_G(u,v)$。

图 G 的**直径**(diameter)：$D = \max\{d_G(u,v) \mid \forall u, v \in V(G)\}$。

简单图 *G* 中最短圈的长度称为图 *G* 的围长(girth)，最长圈的长度称为图 *G* 的周长(perimeter)。

1.1.4. 连通性 Connectivity

连通(connected)：在图 G 中，两个顶点间，至少存在一条路径，称两个顶点连通的

(connected)；反之，称**非连通**(unconnected)。

强连通(strongly connected)：在有向图 G 中，两个顶点间，至少存在一条路径，称两个顶点强连通。

弱连通(weakly connected)：在有向图 G 中，两个顶点间，若不考虑 G 中边的方向的图才连通的，称原有向图为弱连通。

连通图(connected graph)：图 G 中任二顶点都连通。

连通分量或**连通分支**(connected branch, component)：非连通无向图的极大连通子图(maximally connected sub-graph)。具体说，若图 G 的顶点集 $V(G)$可划分为若干非空子集 $V_1,V_2,\cdots,V_\omega$，使得两顶点属于同一子集当且仅当它们在 G 中连通，则称每个子图 $G[V_i]$ 为图 G 的一个连通分支（$i=1,2,\cdots,\omega$）。图 G 的连通分支是 G 的一个极大连通子图。图 G 连通当且仅当ω=1。

强连通分量(strongly connected branch)：非强连通图有向图的极大强连通子图。

割(cut)：

点割集(vertex cut)：点集$V'\in V$，若 G 删除了V'后不连通，但删除了V'的任意真子集后 G 仍然连通。则称V'点割集。若某一结点就构成就了点割集，则称此结点**割点**(cut vertex)。点数最少的点割集称为**点连通度** k(G)。

边割集(edge cut set)：边集$E'\in E$，若 G 删除了E'后不连通，但删除了E'的任意真子集后 G 仍然连通。则称E'点割集。若某一边就构成就了边割集，则称此结点**割边**(cut edge)或**桥**(bridge)。边数最少的边割集称为**边连通度** k'(G)。

记号$[S,S']$表示一端在 S 中另一端在S'中的所有边的集合。

块(block)是指没有割点的极大连通子图。

1.1.5. 图论中特殊的集合 Sets in graph

点覆盖(集)(vertex covering (set))：$V'\in V$，满足对于$\forall e\in E$，有$\exists v\in V'$，v关联e。即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。**极小点覆盖**(minimal vertex covering)：本身为点覆盖，其真子集都不是。**最小点覆盖**(minimum vertex covering)：点最少的点覆盖。**点覆盖数**(vertex covering number)：最小点覆盖的点数，记为$\alpha_V(G)$

一般说覆盖集就是指点覆盖集。

边覆盖(集)(edge covering (set))：$E'\in E$，满足对于$\forall v\in V$，有$\exists e\in E'$，e关联v。即一个边集，使得所有点都与集合里的边邻接。或者说是“边” 覆盖了所有“点”。 **极小边覆盖**(minimal edge covering)：本身是边覆盖，其真子集都不是。**最小边覆盖**(minimum edge covering)：边最少的边覆盖。**边覆盖数**(edge covering number)：最小边覆盖的边数，记为$\alpha_E(G)$。

独立集(independent set)：$V'\in V$，满足对于$\forall u,v\in V'$，有$(u,v)\notin E$。即一个点集，集合中任两个结点不相邻，则称V'为独立集。或者说是导出的子图是零图（没有边）的点集。**极大独立集**(maximal independent set)：本身为独立集，再加入任何点都不是。**最大独立集**(maximum independent set)：点最多的独立集。**独立数**(independent number)：最大独立集的点

数，记为 $\beta_V(G)$。

团(clique)：$V'\in V$，满足对于 $\forall u,v\in V'$，有 $(u,v)\in E$。即一个点集，集合中任两个结点相邻。或者说是导出的子图是完全图的点集。**极大团**(maximal clique)：本身为团，再加入任何点都不是。**最大团**(maximum clique)：点最多的团。**团数**(clique number)：最大团的点数，记为 $\omega(G)$。

边独立集(edge independent set)：$E'\in E$，满足对于 $\forall e,f\in E'$，有 e,f 不邻接。即一个边集，满足边集中的任两边不邻接。**极大边独立集**(maximal edge independent set)：本身为边独立集，再加入任何边都不是。**最大边独立集**(maximum edge independent set)：边最多的边独立集。**边独立数**(edge independent number)：最大边独立集的边数，记为 $\beta_E(G)$。

边独立集又称匹配(matching)，相应的有**极大匹配**(maximal matching)，**最大匹配**(maximum matching)，**匹配数**(matching number)。

支配集(dominating set)：$V'\in V$，满足对于 $\forall u\in V-V'$，有 $\exists v\in V'$，$(u,v)\in E$。即一个点集，使得所有其他点至少有一个相邻点在集合里。或者说是一部分的“点”支配了所有“点”。**极小支配集**(minimal dominating set)：本身为支配集，其真子集都不是。**最小支配集**(minimum dominating set)：点最少的支配集。**支配数**(dominating number)：最小支配集的点数，记为 $\gamma_V(G)$。

边支配集(edge dominating set)：$E'\in E$，满足对于 $\forall e\in E-E'$，有 $\exists f\in E'$，e,f 邻接。即一个边集，使得所有边至少有一条邻接边在集合里。或者说是一部分的“边”支配了所有“边”。**极小边支配集**(minimal edge dominating set)：本身是边支配集，其真子集都不是。**最小边支配集**(minimum edge dominating set)：边最少的边支配集。**边支配数**(edge dominating number)：最小边支配集的边数，记为 $\gamma_E(G)$。

//**********

（？）定理：若 G 中无孤立点，D 为支配集，则 D=V(G)-D 也是一个支配集。

定理：一个独立集是极大独立集，当且仅当它是支配集。

关系：

定理：无向图 G 无孤立点，V_1 是极小支配集，则存在 V_2 是极小支配集，且 $V_1\cap V_2=\varnothing$。

定理：无向图 G 无孤立点，V' 是极大独立集，则 V' 是极小支配集。逆命题不成立。

$\beta_V(G)\geq\gamma_V(G)$。

定理：连通图中，V' 是点覆盖，则 V' 是支配集。极小点覆盖不一定是极小支配集。支配集不一定是点覆盖。

定理：无向图 G 无孤立点，V' 是(极，最小)点覆盖，充要于 $V-V'$ 是(极，最大)独立集。

$\alpha_V(G)+\beta_V(G)=|V|$。

定理：无向图 G，V'是 G 的(极，最大)团，充要于V'是$\overline{G}$的(极，最大)独立集。$\omega(G)=\beta_V(\overline{G})$。

由上述定理知，$\alpha_V(G)$，$\beta_V(G)$，$\omega(G)$三者互相确定，但都是 NPC 的。但是二分图中，点覆盖数是匹配数。

M 是匹配，W 是边覆盖，N 是点覆盖，Y 是点独立集。

定理：无向图 G 无孤立点，|M|<=|N|,|Y|<=|W|

定理：无向图 G 无孤立点，$\alpha_E(G)+\beta_E(G)=|V|$。先取一个最大匹配 M，1 条边盖两个点；剩下的一个未盖点要用一条边来覆盖，边覆盖数=|M| +(|V|-2|M|)= |V|-|M|。

定理：无向图 G 无孤立点，$\beta_E(G)=\alpha_V(G)$，$\beta_V(G)=\alpha_E(G)$。

定理：无向图 G 无孤立点，$\omega(G)=\beta_V(G)$。

求匹配数是 P 的，所以边覆盖和匹配都是易求的。

**********//

最小路径覆盖(path covering)：是“路径” 覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图 G 的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为 0(单个点)。

最小路径覆盖数＝G 的点数－最小路径覆盖中的边数。应该使得最小路径覆盖中的边数尽量多，但是又不能让两条边在同一个顶点相交。拆点：将每一个顶点 i 拆成两个顶点 Xi 和 Yi。然后根据原图中边的信息，从 X 部往 Y 部引边。所有边的方向都是由 X 部到 Y 部。因此，所转化出的二分图的最大匹配数则是原图 G 中最小路径覆盖上的边数。因此由最小路径覆盖数＝原图 G 的顶点数－二分图的最大匹配数便可以得解。

1.1.6. 匹配 Matching

匹配(matching)是一个边集，满足边集中的边两两不邻接。匹配又称**边独立集**(edge independent set)。

在匹配中的点称为**匹配点**(matched vertex)或饱和点；反之，称为**未匹配点**(unmatched vertex)或未饱和点。

交错轨(alternating path)是图的一条简单路径，满足任意相邻的两条边，一条在匹配内，一条不在匹配内。

增广轨(augmenting path)：是一个始点与终点都为未匹配点的交错轨。

最大匹配(maximum matching)是具有最多边的匹配。

匹配数(matching number)是最大匹配的大小。

完美匹配(perfect matching)是匹配了所有点的匹配。

完备匹配(complete matching)是匹配了二分图较小部份的所有点的匹配。

增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。

综上，**在二分图中，最小覆盖数=最大匹配数。**边覆盖数=最大独立数=|V|-最大匹配数。

1.1.7. 树 Tree

G=(V, E)为一个图，则下列命题等价：(1)G 是一棵树；(2)G 连通，且|E|=|V|-1；(3)G 无圈，且|E|=|V|-1；(4)G 的任何两个顶点之间存在唯一的一条路；(5)G 连通，且将 G 的任何一条弧删去之后，该图成为非连通图；(6)G 无圈，且在 G 的任何两个不相邻顶点之间加入一条弧之后，该图正好含有一个圈。

Cayley 公式：在 n 阶完全图 K_n 中，不同生成树的个数为 n^{n-2} 。

1.1.8. 组合优化 Combinatorial optimization

从若干可能的安排或方案中寻求某种意义下的最优安排或方案，数学上把这种问题称为**(最)优化**(optimization)问题。

所谓**组合(最)优化**(combinatorial optimization)又称**离散优化**(discrete optimization)，它是通过数学方法去寻找离散事件的最优编排、分组、次序或筛选等. 这类问题可用数学模型描述为：

$$\min f(x)$$
$$s.t.\ g(x) \geq 0,\ x \in D$$

其中 D 表示有限个点组成的集合(定义域)，f 为目标函数，$F = \{x \mid x \in D, g(x) \geq 0\}$ 为可行域。

网络优化(network optimization)就是研究与(赋权)图有关的组合优化问题。

常见的 P 类网络优化问题：最小生成树，最短路，最大流，最小费用最大流，最大匹配，中国邮路问题。

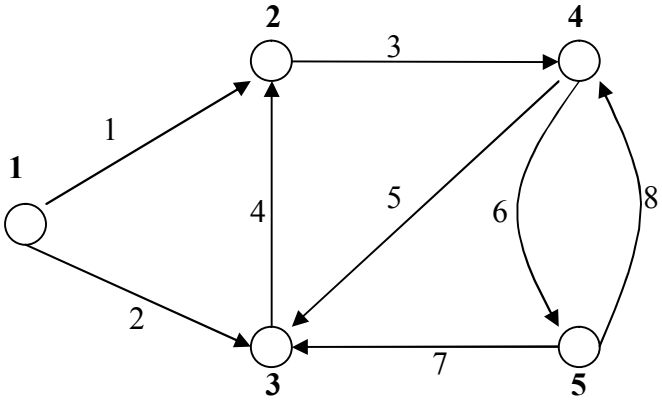
常见的 NP 类网络优化问题：旅行商问题。

参考文献：

[1]Dictionary of Algorithms and Data Structures NIST，http://www.nist.gov/dads/
[2]Wikipedia，http://en.wikipedia.org/wiki/Graph_theory
[3]谢金星，清华大学数学科学系<<网络优化>>讲义
http://faculty.math.tsinghua.edu.cn/~jxie/courses/netopt

1.2. 图的表示 Expressions of graph

下面介绍几种表示图的数据结构。并统一用下图做例子：



1.2.1. 邻接矩阵 Adjacency matrix

用二元数组 $A(u,v)$，来表示图。**这种表示法一般用于稠密图。当图不是简单图，邻接矩阵法不能用。**

在无权图中，若边 (u,v) 存在， $A(u,v)=1$；否则 $A(u,v)=0$。

$$A=(a_{u,v})_{n\times n}\in\{0,1\}^{n\times n},\quad a_{u,v}=\begin{cases}0, & (u,v)\notin E\\ 1, & (u,v)\in E\end{cases}$$

无权图的例子：

$$\begin{bmatrix}0&1&1&0&0\\0&0&0&1&0\\0&1&0&0&0\\0&0&1&0&1\\0&0&1&1&0\end{bmatrix}$$

在有权图中，若边 (u,v) 存在,则 $A(u,v)$ 为它的权值；否则人为的规定 $A(u,v)=\infty$ 或 $-\infty$。∞ 是一个足够大的数。

$$A=(a_{u,v})_{n\times n},\quad a_{u,v}=\begin{cases}w(u,v), & (u,v)\notin E\\ \infty, & (u,v)\in E\end{cases}$$

无向图中，邻接矩阵是按矩阵副对角线对称的。

1.2.2. 关联矩阵 Incidence matrix

用二元数组 $B(u,k)$，来**表示无权有向图。一般不用这种表示法。**

若边 k 与点 u 关联，若 k 是 u 的出边，则 $B(u,k)=1$；若 k 是 u 的入边 $B(u,k)=-1$；否则

$B(u,k)=0$。

$$B=(b_{u,k})_{n\times m}\in\{-1,0,1\}^{n\times m},\quad b_{u,k}=\begin{cases}1, & \exists v\in V,k=(u,v)\in E,\\ -1, & \exists v\in V,k=(v,u)\in E,\\ 0, & else\end{cases}$$

无权图的例子：

$$\begin{bmatrix}1&1&0&0&0&0&0&0\\-1&0&1&-1&0&0&0&0\\0&-1&0&1&-1&0&-1&0\\0&0&-1&0&1&1&0&-1\\0&0&0&0&0&-1&1&1\end{bmatrix}$$

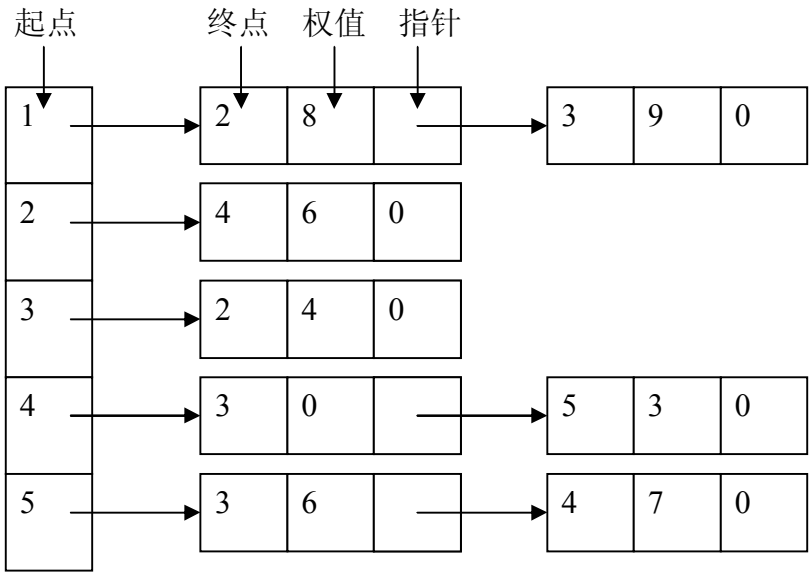
1.2.3. 邻接表 Adjacency list

图的邻接表是图的所有节点的邻接表的集合；而对每个节点，它的邻接表就是它的所有出弧的集合，含有终点，权值等信息。

对于有向图 G=(V,E)，一般用 A(v)表示节点 v 的邻接表，即节点 v 的所有出弧构成的集合或链表(实际上只需要列出弧的另一个端点，即弧的尾)。

一般图都适用。邻接表方法增加或删除一条弧所需的计算工作量很少。

有权图的例子：A(1)={2,3}，A(2)={4}，A(3)={2}，A(4)={3,5}，A(5)={3,4}



1.2.4. 弧表 Arc list

所谓图的弧表，也就是图的弧集合中的所有有序对以表格的方式来表示。弧表表示法直接列出所有弧的起点和终点，以及权值。一般用于稀疏图。缺点是无法通过一些信息(起点，终点)定位一条边。

用 S(i),F(i),W(i)分别表示起点，终点，权值。

有权图的例子：

起点	1	3	4	5	5	4	2	1
终点	2	2	5	4	3	3	4	3
权值	8	4	3	7	6	0	6	9

1.2.5. 星形表示 Star

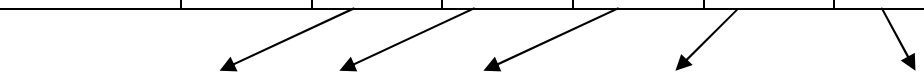
星形表示法就是对弧表的缺点的改进，使之可以通过起点或终点定位边。由于很多时候，算法只需事先知道起点，通过枚举边扩展，而不需要事先知道终点；如图的遍历，松弛操作。

按定位方式，又分前向星形(forwards star)与反向星形(reverse star)。前向星形：通过起点定位边。反向星形：通过终点定位边。实际上，反向星形几乎没用。故本文只讨论前向星形。

通常有两种方法实现这种对弧表改进：边排序法，链表法。

边排序法：把弧表按起点为第一关键字，终点为第二关键字来排序。排序用不用额外空间的快速排序 O(mlogm)或用额外空间 O(m)的计数排序 O(m)均可。之后用数组 last(u)记录以结点 u 为起点的最后一条边的编号，并规定 last(0)=0。这样以结点 u 为起点的边编号就是 last(u-1)+1 到 last(u)。有权图的例子：

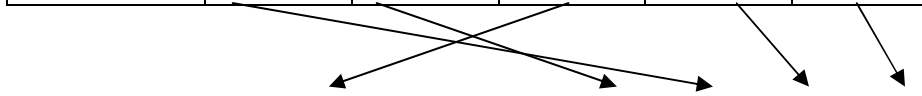
作为起点的点	0	1	2	3	4	5
最后边的编号	0	2	3	4	6	8



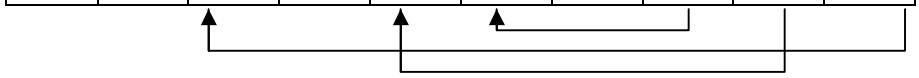
编号	1	2	3	4	5	6	7	8
起点	1	1	2	3	4	4	5	5
终点	2	3	4	2	3	5	3	4
权值	8	9	6	4	0	3	6	7

链表法：给每条边(u,v)加一个前趋，表示以 u 为起点的边链表的前一条边。直观的讲，就是将相同结点的边用链表串起来。last(u)存以 u 为起点的最后一条边的编号。有权图的例子：

作为起点的点	1	2	3	4	5
最后边的编号	6	5	2	7	8



编号	0	1	2	3	4	5	6	7	8
起点	nil	5	3	4	1	2	1	4	5
终点	nil	4	2	5	2	4	3	3	3
权值	nil	7	4	3	8	6	9	0	6
前趋	nil	0	0	0	0	0	4	3	1



星形表示法的优点是占用的存贮空间较少。一般图都适用。边排序法的优点是已知起点和终点的情况下可以精确定位边，容易在起点定位的范围内二分查找终点，在反向边的定位中常用；缺点是代码麻烦，时间抑或空间上都有额外开销。**链表法的优点很多，不仅代码简单，而且没有太多的时空开销，对于反向边的定位只要多加一个数据项纪录下反向边即可；除了终点定位性，几乎没缺点。（就是所谓的“池子法”）**

参考文献：
[1]谢金星，清华大学数学科学系<<网络优化>>讲义
http://faculty.math.tsinghua.edu.cn/~jxie/courses/netopt
[2]刘汝佳，黄亮，<<算法艺术与信息学竞赛>>，P60

1.3. 图的遍历 Traveling in graph

1.3.1. 深度优先搜索 Depth first search (DFS)

1.3.1.1. 概念

1.3.1.2. 求无向连通图中的桥 Finding bridges in undirected graph

在无向连通的条件下，边是桥的充要条件是：1.桥一定是 DFS 树中的边；2.桥一定不在圈中。

圈是由一条后向边(u,v)与 DFS 树中 u 到 v 的路径组成。也就是说 u 到 v 的路径上的边都不可能是桥，应该给以标记。记 f(x)为 **x 与其子孙的后向边所连到的最老祖先**(深度最浅)，表示 x 到 f(x) 上的边均不为桥。然而维护 f(x)比较麻烦，其实只要知道 f(x)的**拓扑序数**就可以了。所谓拓扑序数就是**满足儿子的序数总比父亲大**的一个编号方式。这个拓扑序，常用使用深度 d，或者使用时间戳(TimeStamp) DFN（DFS 访问的次序）。下面以深度为例：

记 $g(x)=d(f(x))$，在 DFS 树中从 x 开始通过前向弧和后向弧所能到达的最小的 d。有以下的动态规划：

$$g(x)=\min\begin{cases} d(x) & \\ d(y) & (x,y)\ is\ backforward\ edge,\ y\neq father \\ g(c) & c\ is\ x's\ child. \end{cases}$$

这里：
1.第一次访问 x 时，记录 d(x)
2. d(y)自己发出去的后向边所达到的深度。
3. g(c)就是其子孙中的 g 最小值。

最后，若 g(x)=d(x)，即(father,x)不在圈中，则(father, x)就是桥。

1.3.2. 广度优先搜索 Breadth first search (BFS)

1.4. 拓扑排序 Topological sort

拓扑排序是对有向无圈图(DAG)顶点的一种排序，它使得如果存在 u,v 的有向路径，那么满足序中 u 在 v 前。拓扑排序就是由一种偏序(partical order)得到一个全序(称为拓扑有序(topological order))。偏序是满足自反性，反对称性，传递性的序。

拓补排序的思路很简单，就是每次任意找一个入度为 0 的点输出，并把这个点以及与这个点相关的边删除。实际算法中，用一个队列实现。

算法：

1. 把所有入度=0 的点入队 Q。
2. 若队 Q 非空，则点 u 出队，输出 u；否则转 4。
3. 把所有与点 u 相关的边(u,v)删除，若此过程中有点 v 的入度变为 0，则把 v 入队 Q，转 2。
4. 若出队点数<N，则有圈；否则输出结果。

算法复杂度: O(m)。

习题：MIPT 012 Correct dictionary

设 R 为非空集合 A 上的二元关系，如果 R 满足自反性(对于每一个 x∈A，(x,x)∈R)，反对称性((x,y)∈R∧(y,x)∈R→x=y)和传递性((x,y)∈R∧(y,x)∈R→(x,z)∈R)，则称 R 为 A 上的偏序关系，记作≤。如果(x,y)∈R，则记作 x≤y，读作“x 小于等于 y”。存在偏序关系的集合 A 称为**偏序集**(partical order)。

拓扑排序的计数。

？？？？？？？？？？？？？？？？？？？？？？？？？？？？？？

AOV(activity on vertex network)

AOE(activity on edge network)，其中顶点表示事件 event，权表示时间。

1.5. 路径与回路 Paths and circuits

1.5.1. 欧拉路径或回路 Eulerian path

对于连通的可重边的图 G，若存在一回路，它通过 G 的所有边一次且仅一次，则这回路称为欧拉路径或回路。

著名的问题：The Königsberg Bridges

下面讨论有向性：

1.5.1.1. 无向图

习题：Ural 1124 Mosaic

1.5.1.2. 有向图

习题：CEOI 2005 Depot Rearrangement

1.5.1.3. 混合图

混合图是指有的边是有向的，有的边是无向的图。

由于无向边只能经过一次，所以不能拆成两条方向相反的有向边，只能给无向边定向，使得定向后的图满足“入度等于出度”。
见 LRJ P324。

下面讨论在有权性上的扩展：

1.5.1.4. 无权图 Unweighted

//**********

1.5.1.5. 有权图 Weighed — 中国邮路问题 The Chinese post problem

这就是一个经典的问题中国邮路问题：给出一个连通的无向的可重边的有权图 G，求最短的回路，使得每边至少遍历 1 次。

由于每边至少遍历一次，所以最短路的瓶颈就在于重复遍历。由于图一直保持连通性，所以两两奇点之间都存在欧拉路；又两两奇点之间的最短路可求；奇点个数为偶数。所以问题就等价于找一个奇点构成的完全图 G’(V,E)的最小权匹配(Perfect Matching in General Graph)。V(G’)为原图 G 中的奇点，每条边为两奇点对应原图的最短路长度。

算法：

1. 确定 G 中的奇点，构成 G’。
2. 确定 G’两两结点在 G 中的最短路作为它们在 G’中的边权。
3. 对 G’进行最小权匹配。
4. 最小权匹配里的各匹配边所对应的路径在 G 中被重复遍历一次,得到欧拉图 G’’。
5. 对 G’’找一条欧拉路即可。

有向的中国邮路问题，比较复杂。
**********//

1.5.2. Hamiltonian Cycle 哈氏路径与回路

分支限界搜索
模拟退火

1.5.2.1. 无权图 Unweighted

1.5.2.2. 有权图 Weighed — 旅行商问题 The travelling salesman problem

动态规划

1.6. 网络优化 Network optimization

1.6.1. 最小生成树 Minimum spanning trees

最小生成树是指连通图中所有生成树中边权和最小的一个。即：求 G 的一棵生成树 T，使得

$$w(T) = \min_{T} \sum_{e \in T} w(e)$$

1.6.1.1. 基本算法 Basic algorithms

1.6.1.1.1. Prim

基本思想：不断扩展一棵子树 $T = (S, E'), E' \in E$，直到 S 包括原图的全部顶点，得到最小生成树 T。每次增加一条边，使得这条边是由当前子树结点集 S 及其补集 $\overline{S}$ 所形成的边割集的最小边。

令 d(v)为 v 到结点集 S 的最小距离。

算法：

1. $S = \{v\}, E' = \varnothing$,d(v_0)=0 (这里 v_0 是任意一个结点), $d(v) = \infty, (v \neq v_0)$
2. 若 $|S| = N$，则结束；否则，转 3。
3. 找补集 $\overline{S}$ 中的 d 最小的节点 v，加入 S。更新与 v 相邻的结点 w 的 d 值，即若 $d(w) > W(v, w)$，

则 $d(w)=W(v,w)$，转 2。

这里的 d 可以用优先队列实现，需用到删除最小(DeleteMin)与减值(DecreaseKey)的操作。假设用 Fibonacci Heap 实现(删除最小 O(logn)，减值 O(1))，算法复杂度：$O(n\log n+m)$。

1.6.1.1.2. Kruskal

基本思想：就是维护一个生成森林。每次将一条权最小的边加入子图 T 中，并保证不形成圈。如果当前弧加入后不形成圈，则加入这条弧，如果当前弧加入后会形成圈，则不加入这条弧，并考虑下一条弧。

算法：

1. $T=\varnothing, i=0$，将 E 中的边按权从小到大排序，$W(e_1)\le W(e_2)\le\cdots\le W(e_m)$。
2. i=i+1，若 i>m，结束，此时 G 没有生成树；否则判断 $T\cup e_i$ 是否含圈，是则转 2，否则转 3。
3. $T=T\cup e_i$。若|T|=N，结束，此时 T 为 G 的最小生成树。

分离集合(disjoint set)，可用并查集实现。由于排序是 $O(m\log m)$ 的。所以复杂度为 $O(m\log m+ma(n))$。

1.6.1.1.3. Sollin（Boruvka）

基本思想：前面介绍的两种算法的综合。每次迭代同时扩展多棵子树，直到得到最小生成树 T。

算法：

1. 对于所有 $v\in V, G_v=\{v\}$。$T=\varnothing$。
2. 若|T|=N，结束，此时 T 为 G 的最小生成树；否则，对于 T 中所有的子树集合 G_v，计算它的边割 $\left[G_v,\overline{G_v}\right]$ 中的最小弧 e_v^*（有的书称连接两个连通分量的最小弧“安全边”）。
3. 对 T 中所有子树集合 G_v 及其边割最小弧 $e_v^*=(p,q)$，将 G_v 与 q 所在的子树集合合并。$T=T\cup e_v^*$。转 2。

由于每次循环迭代时，每棵树都会合并成一棵较大的子树，因此每次循环迭代都会使子树的数量至少减少一半，或者说第 i 次迭代每个分量大小至少为 2^i。所以，循环迭代的总次数为 O(logn)。每次循环迭代所需要的计算时间：对于第 2 步，每次检查所有边 O(m)，去更

新每个连通分量的最小弧；对于第 3 步，合并 $O(n/2^i)$ 个子树。所以总的复杂度为 $O(m\log n)$。

```
BORŮVKA(V, E):
  F = (V, ∅)
  while F has more than one component
      choose leaders using DFS
      FINDSAFEEDGES(V, E)
      for each leader v̄
          add safe(v̄) to F
```

```
FINDSAFEEDGES(V, E):
  for each leader v̄
      safe(v̄) ← ∞
  for each edge (u, v) ∈ E
      ū ← leader(u)
      v̄ ← leader(v)
      if ū ≠ v̄
          if w(u, v) < w(safe(ū))
              safe(ū) ← (u, v)
          if w(u, v) < w(safe(v̄))
              safe(v̄) ← (u, v)
```

1.6.1.2. 扩展模型 Extended models

1.6.1.2.1. 度限制生成树 Minimum degree-bounded spanning trees

由于这个问题是 NP-Hard 的，一般用搜索算法解决。本文就只讨论一种特殊多项式情况：单点度限制(one node degree bounded)。

把度限制的点记为 v_0，满足度限制 $\deg(v_0)\le k$。一种贪心的思路：在最小生成树 T 的基础上，通过添删边来改造树，使之逐渐满足度限制条件。

算法：

1. 求图的最小生成树 T。
2. 若 v_0 已满足度限制，结束；否则转 3。
3. 对于删去 v_0 后的每一个连通分支 T_i(为一棵树)，求添加边割 $[T_i,\overline{T_i}-\{v_0\}]$ 中的最小弧，删除边割 $[T_i,\{v0\}]$ 里的最大弧后的生成树中的边权和最小的一个。更新最小生成树 T。转 2。

第 3 步，v_0 的度少 1。

习题：NOI 2005 小 H 的聚会

1.6.1.2.2. k 小生成树 The k minimum spanning tree problem(k-MST)

生成树 T 删除一条边 f 并加入一条新边 e 的操作称为**交换**。若交换后的图仍是一颗树，则

此交换称为**可行交换**。若生成树 T 可通过一次交换成为生成树 T'，则称它们互为**邻树**。对于生成树集合 S，生成树 T，若 T 不在 S 中，且在 S 中存在某生成树是 T 的邻树，称为 T 为 S 的邻树。

定理：设 $T_1,T_2,\cdots,T_k$ 为图的前 k 小生成树，则生成图集合 $\{T_1,T_2,\cdots,T_k\}$ 的邻树中的边权和最小者可作为第 k+1 小生成树(可能有边权和相同的情况)。

按这个定理设计算法，很难得到有满意的时间复杂度的算法。

下面讨论一个特例：次小生成树(The second MST, 2-MST)

基本思想：枚举最小生成树 T 的每一个邻树，即枚举被添加与被删除的边。由于在树中添加一条边 (u,v) ($(u,v)\notin T$)，一定形成了一个环，环是由 (u,v) 与从 u 到 v 的生成树上的唯一路径(记为 $P(u,v)$)组成的。所以被删除的边一定在 $P(u,v)$ 上。由于要求最小边权和，所以被删除的边一定是 $P(u,v)$ 上最小者，其权值记为 $f(u,v)$ ： $f(u,v)=\min\{w(e)\mid e\in P(u,v)\}$ 。

算法：

1. 求图的最小生成树 T。次小生成树 T'的权值的最小值 $w(T')=\infty$ 。
2. 以每个结点为根 r，DFS 遍历树。在遍历过程中求出 $f(r,v)\mid v\in V$ ，用 $w(T)+w(r,v)-f(r,v)$ 更新次小生成树的权值的最小值 $w(T')$ 。
3. 输出 $w(T')$ 。

算法复杂度的瓶颈在第 2 步 $O(n^2)$ ，故总算法复杂度为 $O(n^2)$ 。

习题：Ural 1416 Confidential

1.6.2. 最短路 Shortest paths

1.6.2.1. 单源最短路 Single-source shortest paths

令 s 为起点，t 为终点。单源最短路问题定义为：对于网络 G=(V,E,W)，寻找 s 到 t 的一条简单路径，使得路径上的所有边权和最少。令 d(v)为 s 到 v 的最短路长度上界。单源最短路问题的规划模型如下：

$$
\begin{aligned}
&(1)\ min \quad d(t)\\
&s.t.\\
&(2)\ d(v)\le d(u)+w(u,v) \quad (u,v)\in E\\
&(3)\ d(s)=0
\end{aligned}
$$

对于只含正有向圈的连通有向网络，从起点 s 到任一顶点 j 都存在最短路，它们构成以起点 s 为根的树形图（称为最短路树（Tree of Shortest Paths））。当某弧(u,v)位于最短路上时，

一定有 $d(v)-d(u)=w(u,v)$。所以最短路的长度可以由 Bellman 方程（最短路方程）唯一确定：

$$(1)\ d(s)=0$$
$$(2)\ d(v)=\min_{u\neq v}\{d(u)+w(u,v)\}$$

在规划模型与最短路方程中都出现了 $d(v)\le d(u)+w(u,v)$ 形式的式子，下面引入松弛操作：松弛操作是对于边(u,v)进行的改进操作：若 $d(v)>d(u)+w(u,v)$，则改进 $d(v)=d(u)+w(u,v)$。直观的讲，就是路径最后通过(u,v)，使得 s 到 v 的距离比原来 s 到 v 的方案的距离短。松弛操作是最短路算法求解的基本方式。

最短路算法求解过程中的标号规定：对于 V 中每一个顶点 v，设置一个标号：距离标号 d(v)，记录的是从起点到该顶点的最短路长度的上界；再设置一个是前趋 pred(v)，记录的是当起点 s 到该顶点 v 的一条路长取到该上界时，该条路中顶点 v 前面的那个直接前趋。算法通过不断修改这些标号，进行迭代计算。当算法结束时，距离标号表示的是从起点到该顶点的最短路长度。

标号设定算法(Label-Setting)：在通过迭代过程对标号进行逐步修正的过程中，**每次迭代将一个顶点从临时标号集合中移入永久标号集合中**。

标号修正算法(Label-Correcting)：每次迭代时并不一定将任何顶点标号从临时标号转变为永久标号，只是对临时标号进行一次修正，所有顶点标号仍然都是临时标号；**只有在所有迭代终止时，所有顶点标号同时转变为永久标号**。

最长路问题可以转化为最短路问题，把弧上的费用反号即可。

1.6.2.1.1. 基本算法 Basic algorithms

1.6.2.1.1.1. Dijkstra

采用了标号设定算法(Label-Setting)。在迭代进行计算的过程中，所有顶点实际上被分成了两类：一类是离起点较近的顶点，它们的距离标号表示的是从点 s 到该顶点的最短路长度，因此其标号不会在以后的迭代中再被改变（称为永久标号）；一类是离起点较远的顶点，它们的距离标号表示的只是从点到该顶点的最短路长度的上界，因此其标号还可能会在以后的迭代中再被改变（称为临时标号）。下文称永久标号为已检查。

算法：

1. $d(s)=0$，$d(v)=\infty,(v\neq s)$，已检查 $U=\varnothing$

2. 取未检查的 u，即 $u\notin U$，使得 $d(u)$ 最小。若 u 取不到，即 d(u)=∞则结束；否则标记为已检查，即 $U=U\cup\{u\}$。

3. 枚举所有的 u 的临边 (u,v)，满足 v 未检查，即 $v\notin U$。松弛 (u,v)，即若 $d(v)>d(u)+w(u,v)$，

则改进 $d(v)=d(u)+w(u,v)$，pred(v)=u。转 2。

这里的 d 可以用优先队列实现，需用到删除最小(DeleteMin)与减值(DecreaseKey)的操作。假设用 Fibonacci Heap 实现(删除最小 O(logn)，减值 O(1))，则算法复杂度：$O(n\log n+m)$。

若用 Binary Heap 则 $O((n+m)\log n)$。

适用范围：非负权图。

1.6.2.1.1.2. Bellman-Ford

采用了标号修正算法(Label-Correcting)。本质就是用迭代法（动态规划）解 Bellman-Ford 方程：

$$d^{(1)}(s)=0,$$
$$d^{(1)}(v)=w(s,v),\quad v\neq s$$
$$d^{(k+1)}(v)=\min\{d^{(k)}(v),\min_{u\neq v}\{d^{(k)}(u)+w(u,v)\}\},\quad u,v\in V,k=1,2,\cdots,n-2$$

$d^{(k)}(v)$ 表示 s 到 v 的且边数不超过 k 条时的最短路路长。下面用归纳法证明：k=1 显然成立。假设对 k 成立，下面考虑 k+1 的情况.从起点 s 到顶点 v 且所经过的弧数不超过 k+1 条时的最短路有两种可能：含有不超过 k 条弧，即 $d^{(k)}(v)$；含有 k+1 条弧，即 $\min_{u\neq v}\{d^{(k)}(u)+w(u,v)\}$。

由于第 k+1 次迭代过程中，不会影响 k 次的迭代结果，d 用 O(n)的空间即可。

算法：

1. $d(s)=0$，$d(v)=\infty,(v\neq s)$
2. 松弛所有边(u,v)，即若 $d(v)>d(u)+w(u,v)$，则改进 $d(v)=d(u)+w(u,v)$，pred(v)=u。若没有边被松弛，则算法结束。
3. 若 2 的迭代次数超过 n-1，则有负权圈，结束；否则转 2 继续迭代。

可以证明算法一定在 n-1 步迭代后收敛；否则一定有负权圈。所以算法复杂度为 O(nm)。

适用范围：一般图。

1.6.2.1.1.2.1.Shortest path faster algorithm(SPFA)

SPFA 其实就是 Bellman-Ford 的一种队列实现，减少了冗余，即松驰的边至少不会以一个 d 为∞的点为起点。

算法：

1. 队列 Q={s}，$d(s)=0$，$d(v)=\infty,(v\neq s)$
2. 取出队头 u，枚举所有的 u 的临边 (u,v)。若 $d(v)>d(u)+w(u,v)$，则改进

$d(v)=d(u)+w(u,v)$，pred(v)=u，由于$d(v)$减少了，v 可能在以后改进其他的点，所以若 v 不在 Q 中，则将 v 入队。

3. 一直迭代 2，直到队列 Q 为空(正常结束)，或有的点的入队次数>=n（含有负圈）。

由于点可能多次入队，但队列中同时不会超过 n 个点。所以用一个长度为 n 的循环队列来实现这个队。

SPFA 在形式上和宽度优先搜索非常类似，不同的是宽度优先搜索中一个点出了队列就不可能重新进入队列，但是 SPFA 中一个点可能在出队列之后再次被放入队列，也就是一个点改进过其它的点之后，过了一段时间可能本身被改进，于是再次用来改进其它的点，这样反复迭代下去。设一个点用来作为迭代点对其它点进行改进的平均次数为 k，有办法证明对于通常的（不含负圈，较稀疏）情况，k 在 2 左右。算法复杂度理论上同 Bellman-Ford，O(nm)，但实际上却是 O(km)。

一般用于找负圈(效率高于 Bellman-Ford)，稀疏图的最短路。

习题：Ural 1254 Die Hard （可斜走的网格最短路）。

1.6.2.1.2. 应用 Applications

1.6.2.1.2.1. 差分约束系统 System of difference constraints

差分约束系统：求解 n 个未知数x_i，满足 m 个不等式：

$$x_j - x_i \le b_k,\quad 1 \le i,j \le n, 1 \le k \le m$$

若描述为线形规划模型：$Ax \le B$。$m \times n$矩阵 A 每行含一个 1 一个-1，其他都是 0。

如线形规划模型

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \le \begin{pmatrix} -2 \\ 4 \\ 5 \\ -3 \\ 2 \\ 3 \end{pmatrix}$$

等价于

$$\begin{cases} x_1 - x_3 \le -2 \\ x_2 - x_4 \le 4 \\ x_1 - x_4 \le 5 \\ x_2 - x_3 \le -3 \\ x_4 - x_1 \le 2 \\ x_1 - x_2 \le 3 \end{cases}$$

注意到单源最短路模型中的。$d(v) \le d(u) + w(u,v)$，即 $d(v) - d(u) \le w(u,v)$。很像差分约束系统模型。于是可以构造一个有向网络 G=(V,E,W)，称为约束图(constraints graph)。

$V = \{v_0, v_1, v_2, \cdots, v_n\}$。 $E = \{(v_i, v_j) \mid x_j - x_i \le b_k\} + \{(v_0, v_i) \mid 1 \le i \le n\}$

W： $w(i,j) = b_k \mid x_j - x_i \le b_k$， $w(v_0, v_i) = 0$

对 v_0 进行 Bellman-Ford 算法，可以得到 $d(v_i)$，令 $x_i = d(v_i) + C$，即为所求，其中 C 是任意一个常数，均满足原不等式组。当有 x_i 为负数且题目要求 x_i 非负时，常常令 C 为最小的 x_i 的相反数。

习题：NOI 1999 01 串

1.6.2.1.2.2. 有向无环图上的最短路 Shortest paths in DAG

算法：

1. $d(s) = 0$， $d(v) = \infty, (v \ne s)$。对有向无环图 G，拓扑排序 O(m)。

2.按拓扑序枚举 u。若枚举完，则结束。

3.枚举所有 u 的临边(u,v)，对(u,v)进行松弛操作，即若 $d(v) > d(u) + w(u,v)$，则改进 $d(v) = d(u) + w(u,v)$，pred(v)=u。转 2。

复杂度： $O(m)$

1.6.2.2. 所有顶点对间最短路 All-pairs shortest paths

1.6.2.2.1. 基本算法 Basic algorithms

1.6.2.2.1.1. Floyd-Warshall

本质就是用迭代法（动态规划）：

$$d^{(1)}(u,u) = 0, \quad u \in V$$
$$d^{(1)}(u,v) = w(u,v), \quad u,v \in V$$
$$d^{(k+1)}(u,v) = \min\{d^{(k)}(u,v), d^{(k)}(u,k) + d^{(k)}(k,v)\}, \quad u,v \in V, k = 1,2,\cdots,n-1$$

$d^{(k)}(u,v)$ 表示 u 到 v 的不经过 k,k+1,…,n 结点(除 u,v 外)时，从 u,v 的最短路长。下面用归纳法证明：k=1 显然成立。假设对 k 成立,下面考虑 k+1 的情况；从 u 到 v 且不通过 k+1,…, n 节点的最短路有两种可能：(1)不经过 k 节点，即为 $d^{(k)}(u,v)$；(2)经过 k 节点，即为

$d^{(k)}(u,k)+d^{(k)}(k,v)$。

由于第 k+1 次迭代过程中，不会影响 k 次的迭代结果，d 用 $O(n^2)$ 的空间即可。二维数组 $p(u,v)$，记录 u 到 v，最后经过哪个 k 的迭代。

算法：

1. k=1，对于所有 u,v ， $d(u,v)=w(u,v), p(u,v)=v$
2. k=k+1，对于所有 u,v，若 $d(u,v)>d(u,k)+d(k,v)$，则 $d(u,v)=d(u,k)+d(k,v)$， $p(u,v)=k$。

若发现某个节点 u 使得 $d^{(k)}(u,u)<0$, 则说明网络本来就含有负有向圈，结束。

3. 若 k=n，结束；否则转 2。

算法复杂度： $O(n^3)$

1.6.2.2.1.2. Johnson

本算法适用于稀疏图。它是 Dijkstra 和 Bellman-Ford 算法的综合应用。

本算法用了权值改造(reweighting)技术。若图的权值非负，则对于每个结点用一次 Dijkstra 算法，就可以求出所有顶点对间最短路。若图中有负权，但没有负圈，则可以把权值 W 改造成 W'， W'非负，使得能够使用 Dijkstra 算法。

算法：

1. 给图 G，加一个新结点 v_o。对 v_o 用 Bellman-Ford，若有负圈，则结束；否则可以得到 $h(v)$，即 v_o 到 v 的最短路长。O(nm)
2. 对于所有边 (u,v)，权值改造，即令 $w'(u,v)=w(u,v)+h(u)-h(v)$。O(m)
3. 对于每个结点 v，用一次以 Fibonacci Heap 为优先队列的 Dijkstra 算法，可求出 v 与其他顶点的最短路。O(nlogn+m)*O(n)

算法总复杂度 $O(n^2\log n+nm)$。

权值改造满足两个原则：(1)W' 非负；(2)对于任意顶点 u, v，p 是在 W 上的 u 到 v 的最短路当且仅当 p 是在 W'上的 u 到 v 的最短路。下证 $w'(u,v)=w(u,v)+h(u)-h(v)$，满足以上两个原则：

(1)由于 h 是 v0 到 v 的最短路长，$h(u)+w(u,v)\geq h(v)$，所以 $w'(u,v)=w(u,v)+h(u)-h(v)\geq 0$。

(2)令路径 $p=(v_1,v_2,\cdots,v_n)$， 则

$$w'(p)=\sum_{i=2}^{k}w'(v_{i-1},v_i)=\sum_{i=2}^{k}\left[w(v_{i-1},v_i)+h(v_{i-1})-h(v_i)\right]$$
$$=w(p)+h(v_1)-h(v_k)$$

$w'(p)$ 只与 $w(p)$ 以及首尾的标号 h 有关，不影响 $w'(p)$ 的决策。所以 $w(p)$ 是最短路长当且仅当 $w'(p)$ 是最短路长，且不影响最短路 p。

参考文献：

[1] 25.3 Johnson's algorithm for sparse graphs，Introduction to Algorithms，MIT Press

1.6.3. 网络流 Flow network

1.6.3.1. 最大流 Maximum flow

在有向无权图 G(V,E,C)中，其中 C 为每条边的容量(capability) $c(u,v)$，再给每条边赋予一个流值(flow) $f(u,v)$，并规定源 s 和汇 t。最大流问题的数学模型描述如下：

$$(1)\ \max \quad \sum_{v\in V} f(s,v)$$

s.t.

$$(2)\ f(u,v)\le c(u,v) \quad u,v\in V$$
$$(3)\ f(u,v)=-f(v,u) \quad u,v\in V$$
$$(4)\ \sum_{v\in V} f(u,v)=0 \quad u\in V-\{s,t\}$$

其中(2) $f(u,v)\le c(u,v)$，为容量限制条件：边的流量不超过边上的容量。

其中(3)规定反向边的流量为正向边的流量的相反数。

其中(4)可以改写成：

$$\sum_{(u,w)} f(u,w)=\sum_{(w,v)\in E} f(w,v), \quad w\in V-\{s,t\}$$

称为流量平衡条件。它表示除了源 s 和汇 t 外的结点流入等于流出。

其中(1)表示要求从源流出的流量要最大。

最小切割最大流定理：s-t 最大流流值=s-t 最小切割容量。

习题：Ural 1277 Cops and Thieves (最小切割最大流)

1.6.3.1.1. 基本算法 Basic algorithms

1.6.3.1.1.1. Ford-Fulkerson method

增广轨定理：

1.6.3.1.1.1.1. Edmonds-Karp algorithm

1.6.3.1.1.1.1.1. Minimum length path

1.6.3.1.1.1.1.2. Maximum capability path

1.6.3.1.1.2. 预流推进算法 Preflow push method

1.6.3.1.1.2.1. Push-relabel

1.6.3.1.1.2.2. Relabel-to-front

1.6.3.1.1.3. Dinic method

DINIC & MPM
==========

我们能够做的更快一些么？？

前面的算法需要 O(mn)次迭代，而每次需要 O(m)的时间，因此总共需要 O(m^2 n)。下面我们试着降到 O(n^3)。

想法：假设 d=d(t)，我们将试着一次性的找到很多到 t 的长度为 d 的路径。为了做到这个，我们看前面得到的层次图（我们不考虑所有的后向边以及两个端点在同一层的边）。我们现在尝试着在这个图中执行尽量多的流量，然后我们才重新计算剩余图。

术语：一个充满了这个剩余图的流称之为块流。

因此我们的目标就是在这个剩余图中在 O(n^2)的时间内找到一个块流。

我们定义 c(v)=顶点 v 的容量：也就 c_in[v],c_out[v]的最小值，这里 c_in[v]是所有 v 的入边的容量的和，而 c_out[v]类似。

算法：

-找到具有最小容量 c 的顶点 v。如果他的容量是 0，oh yeah--我们就可以把他和与他相关联的边都删除掉，并且更新他的邻居的容量值。

-如果 c 不为 0，那么我们将从 s 运送 c 个单位流到 v，然后从 v 运送到 t。你可能觉得这不又回到原来的问题了么？但是，这里有点值得注意：因为 v 具有最小的容量，我们可以使用任何的贪心策略来从其他顶点流入流出 c 的流量而不会被堵住。这一点意味着，我们可以在 O(m)的时间内充满 v。

（* daizi
nnd，想了好久才明白具体怎么做的
首先，为了从 s 运送 c 到 v，我们从 v 开始向上考虑，把任意一条从 v 向上的边给充满，如果不能够充满，说明还没有安置的流量就这些了，全部扔在这条边上就是了，为什么一定能够找到这样一条边呢，：）

然后假设我们刚才填充的这条边是 u->v，那么我们就有一些流量需要在 u 继续往上进行处理，使用刚才在 v 点同样的办法，直到一直处理到了 s，一路上肯定是不会出现什么障碍的。

然后我们将 c 从 v 运送到 t，我们在构造这个层次图的时候很容易就可以保证所有从 s 出发的路径都会在 t 终结，这样就好办了，使用上面同样的办法，只是处理的方向是从 v 到 t 了。

这样，我们刚才处理的过程就容易得到下面的下面所谓的性质了。
*）

上面的算法给了我们一个可以在 O(mn)的时间内充满这个层次图的算法，下面进一步分析。

为了得到我们需要的界，我们需要这样一个性质：当我们在上面的对每个新的顶点 v 执行算法的步骤中，保证对每条边我们只检查一次，对其做完所有需要的操作之后才会去做下一条边。这点意味着，我们可以把运行时间分成两个部分来考虑：a）花在被充满的边上的时间 b）花在没有被充满的边上的时间。对于 a），因为我们每次充满一条边就会把它从图中删除，所以 a）的总时间是 O(m)。

因此我们的时间主要由 b）来决定，而 b）对应的边在对每个新的顶点 v 的执行过程中至多出现 O(n)次，因此总的时间是 O(n^2)。而我们可以在 O(m)的时间内重新计算新的层次图，所以我们就可以在 O(n m + n n^2)时间完成整个算法，也就是 O(n^3)。

1.6.3.1.2. 扩展模型 Extended models

1.6.3.1.2.1. 有上下界的流问题

问题模型：

给定一个加权的有向图 $G=(V,E,B,C)$，满足：

(1)容量限制条件：$b(u,v) \le f(u,v) \le c(u,v)$

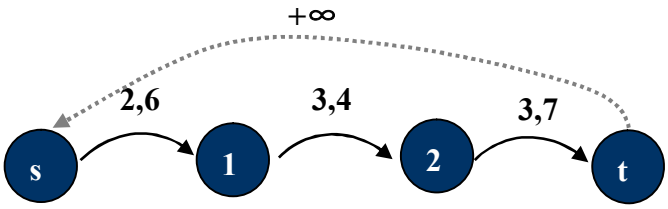
(2)流量平衡条件：$\sum_{(u,w)} f(u,w) = \sum_{(w,v)\in E} f(w,v)$

(2)中的$w \in V - \{s,t\}$，即除了源汇外，所有点都满足流量平衡条件，则称 G 为**有源汇**网络；

否则$w \in V$，即不存在源汇，所有点都满足流量平衡条件，则称 G 为**无源汇**网络。

将这类问题由易到难一一解决：

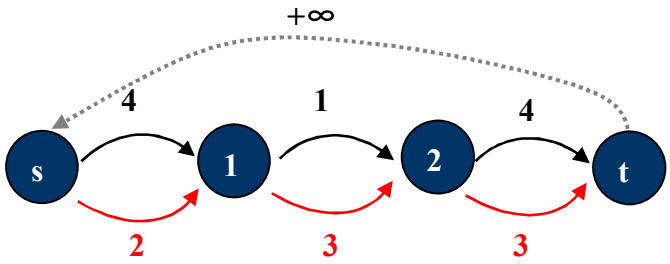
问题[1] 求无源汇的网络有上下界的可行流



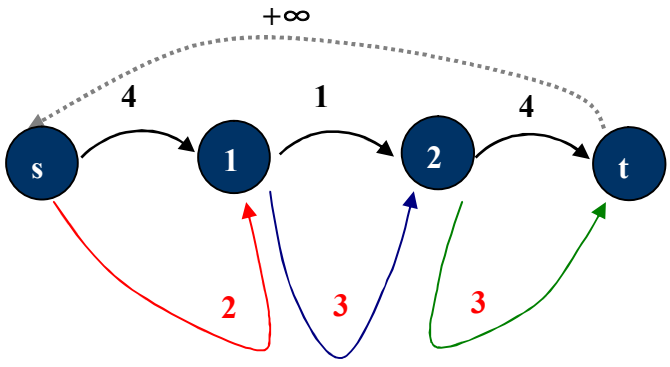
由于下界是一条弧上的流必需要满足的**确定值**。下面引入**必要弧**的概念：**必要弧**是一定流要满的弧。必要弧的构造，将容量下界的限制分离开了，从而构造了一个没有下界的网络 G'：

1．将原弧(u,v)分离出一条必要弧：$c'(u,v)_{nes} = b(u,v)$。（红色表示）

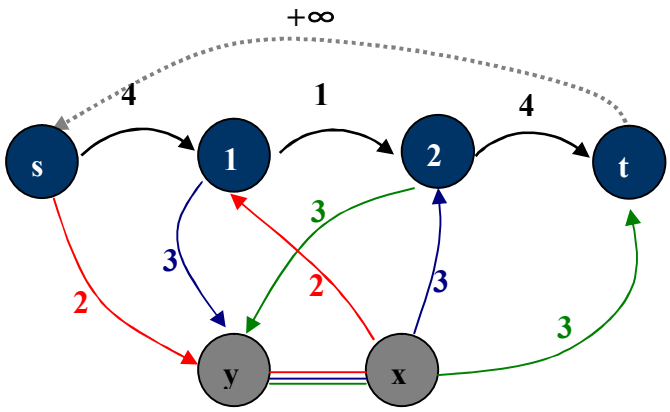
2．原弧：$c'(u,v) = c(u,v) - b(u,v)$。



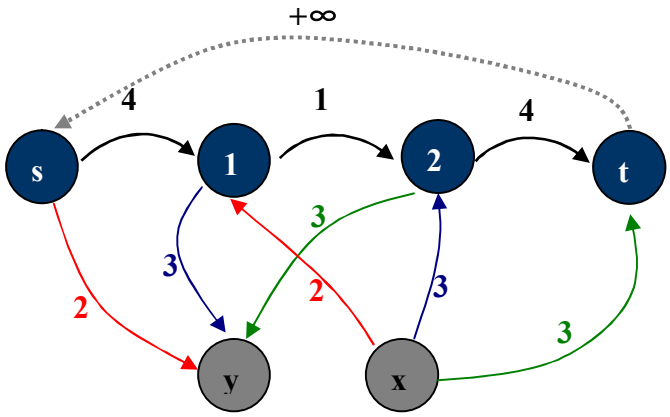
由于必要弧的有一定要满的限制，将必要弧“拉”出来集中考虑：



添加附加源 x，附加汇 y。想像一条不限上界的(y, x)，用必要弧将它们“串”起来，即对于有向必要弧(u, v)，添加(u, y)，(x, v)，容量为必要弧容量。这样就建立了一个等价的网络。



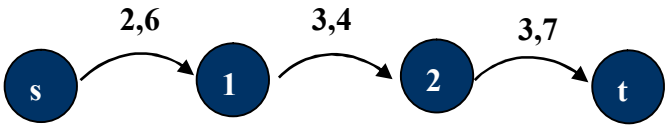
一个无源汇网络的可行流的方案一定是必要弧是满的。若去掉(y, x)后，附加源 x 到附加汇 y 的最大流，能使得 x 的出弧或者 y 的入弧都满，充要于原图有可行流。



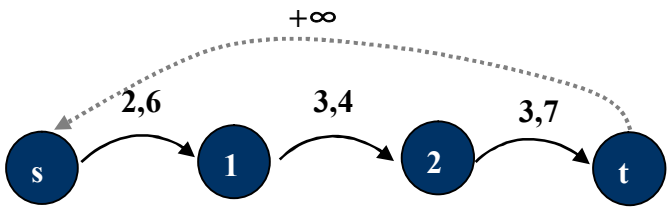
算法：

1. 按上述方法构造新网络(分离必要弧，附加源汇)
2. 求附加源 x 到附加汇 y 的最大流
3. 若 x 的出弧或 y 的入弧都满，则有解，将必要弧合并回原图；否则，无解。

问题[2] 求有源汇的网络有上下界的可行流



加入边(t, s)，下界为 0(保证不会连上附加源汇 x, y)，不限上界，将问题[2]转化为问题[1]来求解。



问题[3]求有源汇的网络有上下界的最大流

算法：

1. 先转化为问题[2]来求解一个可行流。若可行无解，则退出。由于必要弧是分离出来的，所以就可以把必要弧（附加源汇及其临边）及其上的流，暂时删去。再将(T,S)删去，恢复源汇。
2. 再次，从 S 到 T 找增广轨，求最大流。
3. 最后将暂时删去的下界信息恢复，合并到当前图中。输出解。

这样既不破坏下界(分离出来)也不超出上界(第 2 步满足容量限制)，问题解决。

问题[4]求有源汇的网络有上下界的最小流

算法：

1. 同问题[3]。
2. 从 T 到 S 找增广轨，不断反着改进。
3. 同问题[3]。

问题[3]与问题[4]的另一种简易求法：

注意问题[2]中，构造出的(t, s)，上下界几乎没什么限制。下面看看它的性质：

定理：如果从 s 到 t 有一个流量为 a 的可行流 f，那么从 t 到 s 连一条弧(t, s)，其流量下界 $b(t, s) = a$，则这个图一定有一个无源汇的可行流：除了弧(t, s)的容量为 a 外，其余边的容量与 f 相同。

证明：如果从 s 到 t 的最大流量为 a_{max}，那么从 t 到 s 连一条下界 $b(t, s) = a' > a_{max}$ 的弧(t, s)，则从在这个改造后的图中一定没有无源汇的可行流：否则将这个可行流中的弧(t, s)除去，就得到了原图中 s 到 t 的流量为 a'的流，大于最大流量 a_{max}，产生矛盾。

可以二分枚举这个参数 a，即下界 $b(t, s)$，每次用问题[1]判断是否有可行流。这样就可以求出最大流。

同理，问题[4]要求最小流，只要二分枚举上界 $c(t, s)$即可。

因为朴素的预流推进算法 $O(N^3)$，总复杂度为 $O(N^3 \log_2 流量)$ 。

思路：

无源汇 (附加源汇+最大解决)

有源汇 (附加(T,S)->无源汇)

习题：SGU194 Reactor Cooling

SGU176 Flow Construction

参考文献：

[1]安徽 周源，《一种简易的方法求解流量有上下界的网络中网络流问题》，IOI2004 国家集训队作业

[2]江涛，《最大流在信息学竞赛中应用的一个模型》，NOI2006 冬令营

1.6.3.2. 最小费用流 Minimum cost flow

对于有向带权图 G=(V,E,C,W)，求一个图上的流 f，使得每条边的流量与费用的乘积加起来的总和最小，且总流量等于 d。下面是最小费用流问题的数学模型的表述：

$$(1)\ \min \sum_{(u,v)\in E} w(u,v)f(u,v)$$

s.t.

$$(2)\ f(u,v)\le c(u,v) \quad u,v\in V$$

$$(3)\ f(u,v)=-f(v,u) \quad u,v\in V$$

$$(4)\ \sum_{v\in V} f(u,v)=0 \quad u\in V-\{s,t\}$$

$$(5)\ \sum_{v\in V} f(s,v)=d$$

最小费用最大流问题：总流量 d 为网络的最大流的最小费用流问题。

1.6.3.2.1. 找最小费用路 Finding minimum cost path

每次在残量网络中找最小费用路并沿着它增广，直到不存这样增广路。

算法：

1. 用 Bellman-Ford 在残量网络中求 s-t 的最小费用路，若不存在，结束；否则转 2。
2. 沿着找到的最小费用路改进残量网络。转 2。

设 v 是最大流量，则算法最多迭代 v 次，故复杂度为 $O(nmv)$。

习题：FJOI 2002 蚯蚓问题

1.6.3.2.2. 找负权圈 Finding negative circle

在最大流的残量网络中不断的找负费用圈并沿着它增广，直到不存这样的负圈。易知，每次沿负费用圈增广，总费用总是不断的减小。

算法：

1. 用最大流算法求出网络的任一个最大流的残量网络。
2. 用 Bellman-Ford 算法找残量网络的负费用圈，若不存在，结束；否则转 3。
3. 沿着找到的负费用圈改进残量网络。转 2。

设 c 为容量最大值，w 为费用最大值。因为每次消圈，费用至少少 1，而初始费用不超过 mcw，所以复杂度上限：$O(nm^2cw)$，是伪多项式算法。但实际上，会比这个快多了。还有一个按照特定次序消圈的算法，$O(nm^2\log n)$。所以最小费用流问题是 P 类的。

习题：Ural 1237 Evacuation Plan

1.6.3.2.3. 网络单纯形 Network simplex algorithm

我不会阿！！！

1.6.4. 匹配 Matching

所有的匹配算法都基于增广轨定理：一个匹配是最大匹配当且仅当没有增广轨。这个定理适用于任意图。

Kuhn-Munkres
Edmonds
Gabow

1.6.4.1. 二分图 Bipartite Graph

由于二分图（又称二部图）的特殊性，使得它可以转化成流问题来解决。

$O(m\sqrt{n})$

习题：Ural 1203 Conference (最小点覆盖)

1.6.4.1.1. 无权图-匈牙利算法 Unweighted - Hopcroft and Karp algorithm

匈牙利算法，匈牙利数学家 Edmonds 于 1965 年提出。

1.6.4.1.2. 带权图-KM 算法 Weighted –Kuhn-Munkres(KM) algorithm

构造**可行顶标**：结点函数 l，对任意弧 (u,v) 满足：$l(u)+l(v)\geq w(u,v)$ 。

引入**相等子图**：G 的生成子图，包含所有点，但只包含满足 $l(u)+l(v)=w(u,v)$ 的所有弧 (u,v) 。

下证相等子图的一个性质定理：
如果相等子图有完美匹配，则该匹配是原图的最大权匹配。
证明：设 M*是相等子图的完美匹配，M 是原图的任意完美匹配。

$$\begin{aligned} w(M^*) &= \sum_{e\in M^*} w(e) \\ &= \sum_{v\in X\cup Y} l(v) \qquad (1) \\ &\geq \sum_{e\in M} w(e) = w(M) \qquad (2) \end{aligned}$$

(1)是根据相等子图定义;(2)是根据可行顶标定义。

所以关键是寻找好的可行顶标，使相等子图有完美匹配。

$\sum_{v \in V} \deg^+(v) =$

算法思想：随便构造一个可行顶标（例如 Y 结点顶标为 0，X 结点的顶标为它出发所有弧的最大权值，然后求相等子图的最大匹配）

-若存在完美匹配，算法终止

-否则修改顶标使得相等子图的边变多，有更大机会存在完美匹配

Kuhn 和 Munkras

1.6.4.2. 一般图 General Graph

1.6.4.2.1. 无权图-带花树算法 Unweighted - Blossom (Edmonds)

朱(永津)-刘(振宏)算法（1965）
Edmons 算法（1968）

Dictionary of Algorithms and Data Structures NIST
Wikipedia
Introduction to Algorithm