第一章 图论的基本理论

1.1 图的基本概念

无向图 一个无向图 G 是由非空顶点集 V 和边集 E 按一定的对应关系构成的连接结构，记为：

$$G = (V, E).$$

非空集合 $V = \{v_1, v_2, \ldots, v_n\}$ 为 G 的顶点集，V 中的元素称为 G 的顶点，其元素的个数 $|V|$ 为顶点数。

非空集合 $E = \{e_1, e_2, \ldots, e_m\}$ 为 G 的边集，E 中的元素称为 G 的边，其元素的个数 $|E|$ 为图 G 边的条数。

图 G 的每一条边是由连接 G 中两个顶点而得的一条线，通常记作 $e_k = (v_i, v_j)$，其中，顶点 v_i, v_j 称为边 e_k 的两个端点，也称 e_k 与顶点 v_i, v_j 关联。

对无向图来说，对应一条边的顶点对表示是无序的，即 (v_i, v_j) 和 (v_j, v_i) 表示同一条边 e_k。有公共端点的两条边，或称邻边。同一条边 e_k 的两个端点称为相邻的顶点。

环 如果一条边的两个端点是同一个顶点，则称这条边为环。

重边 如果有两条边或多条边的端点是同一对顶点，则称这些边为重边或平行边。

孤立点 不与任何边相关联的顶点。

例题 1.1 设 $V = \{v_1, v_2, v_3, v_4, v_5\}$，$E = \{e_1, e_2, e_3, e_4, e_5\}$，$e_1 = (v_1, v_2)$，$e_2 = (v_2, v_3)$，$e_3 = (v_2, v_3)$，$e_4 = (v_3, v_4)$，$e_5 = (v_4, v_4)$。则 $G = (V, E)$ 是一个图，其图形如图1.1所示。边 e_2 和 e_3 为重边，e_5 为环，顶点 v_5 为孤立点。

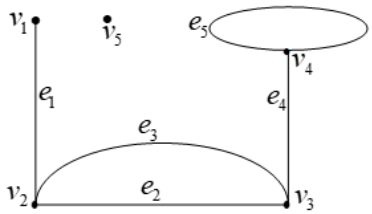


图 1.1: 非简单图例

有向图通常记为：

$$D = (V, A)$$

非空集合 $V = \{v_1, v_2, \ldots, v_n\}$ 为 D 的顶点集，$A = \{a_1, a_2, \ldots, a_n\}$ 为 D 的弧集合。每一条弧与一个有序的顶点对相对应，弧 $a_k = (v_i, v_j)$ 表示自顶点 v_i 指向 v_j 的弧，v_i 称为弧 a_k 的始端，v_j 称为弧 a_k 的末端或终端，其中 a_k 称为 v_i 的出弧，称为 v_j 的入弧。

在有向图中，(v_i, v_j) 与 (v_j, v_i) 表示不同的弧。

把有向图 $D = (V, A)$ 中所有弧的方向都去掉，得到的边集用 E 表示，就得到与有向图 D 对应的无向图 $G = (V, E)$，称 G 为有向图 D 的基本图，称 D 为 G 的定向图。

简单图 无环且无重边的图称为简单图。

如果不特别申明，一般的图均指简单图。

完全图 任意两顶点均相邻的简单图称为完全图。含 n 个顶点的完全图记为 K_n。

赋权图 图 G 的每条边 e 都附有一个实数 $w(e)$，称 $w(e)$ 称为边 e 的权。

赋权图也称为网络。赋权图中的权可以是距离、费用、时间、效益、成本等。赋权图 G 一般记为：

$$G = (V, E, W)$$

其中，W 为权重的邻接矩阵。

如果有向图 D 的每条弧都被赋予了权，则称 D 为有向赋权图。对于无向图、有向图或网络都可以用 G 表示。

顶点的度

在无向图中，与顶点 v 关联的边的数目（环算两次）称为 v 的度，记为 $d(v)$。

在有向图中，从顶点 v 引出的弧的数目称为 v 的出度，记为 $d^+(v)$，从顶点 v 引入的弧的数目称为 v 的入度，记为 $d^-(v)$。

v 的度：$d(v) = d^+(v) + d^-(v)$，度为奇数的顶点称为奇顶点，度为偶数的顶点称为偶顶点。

结论 1. 给定图 $G = (V, E)$，所有顶点的度数之和是边数的 2 倍，即

$$\sum_{v \in V} d(v) = 2|E|$$

结论 2. 任何图中奇顶点的总数必为偶数。

1.2 图的矩阵表示

设图的顶点个数为 n，边（或弧）的条数为 m 。

对于无向图 $G=(V,E)$，其中 $V=\{v_1,v_2,\ldots,v_n\}$，$E=\{e_1,e_2,\ldots,e_m\}$，其关联矩阵为：$M=(m_{ij})_{n\times m}$，其中，

$$m_{ij}=\begin{cases}1, & \text{顶点}v_i\text{与边}e_j\text{关联}\\ 0, & \text{顶点}v_i\text{与边}e_j\text{不关联}\end{cases}$$

对于有向图 $D=(V,A)$，$V=\{v_1,v_2,\ldots,v_n\}$，$A=\{a_1,a_2,\ldots,a_m\}$，其关联矩阵为：$M=(m_{ij})_{n\times m}$，其中，

$$m_{ij}=\begin{cases}1, & \text{顶点}v_i\text{是弧}a_j\text{的始端}\\ -1, & \text{顶点}v_i\text{是弧}a_j\text{的末端}\\ 0, & \text{顶点}v_i\text{与弧}a_j\text{不关联}\end{cases}$$

邻接矩阵

对无向非赋权图 G，其邻接矩阵 $W=(w_{ij})_{n\times n}$，其中

$$w_{ij}=\begin{cases}1, & \text{顶点}v_i\text{与}v_j\text{相邻}\\ 0, & i=j\text{或顶点}v_i\text{与}v_j\text{不相邻}\end{cases}$$

对无向赋权图 G，其邻接矩阵 $W=(w_{ij})_{n\times n}$，其中

$$w_{ij}=\begin{cases}\text{顶点}v_i\text{与}v_j\text{之间边的权}, & (v_i,v_j)\in E\\ 0(\text{或}\infty), & v_i\text{与}v_j\text{之间无边}\end{cases}$$

对有向非赋权图 D，其邻接矩阵 $W=(w_{ij})_{n\times n}$，其中

$$w_{ij}=\begin{cases}1, & \text{弧}(v_i,v_j)\in A\\ 0, & i=j\text{或顶点}v_i\text{到}v_j\text{无弧}\end{cases}$$

例题 1.2 给出图1.2所示的无向赋权图的邻接矩阵。

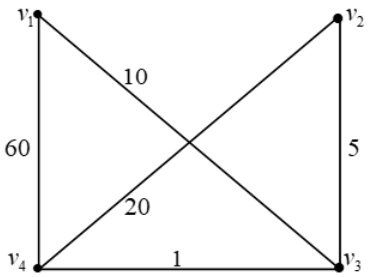


图 1.2: 无向赋权图

$$W=\begin{pmatrix}0 & 0 & 10 & 60\\ 0 & 0 & 5 & 20\\ 10 & 5 & 0 & 1\\ 60 & 20 & 1 & 0\end{pmatrix}$$

例题 1.3 给出图1.3所示的有向非赋权图的邻接矩阵。

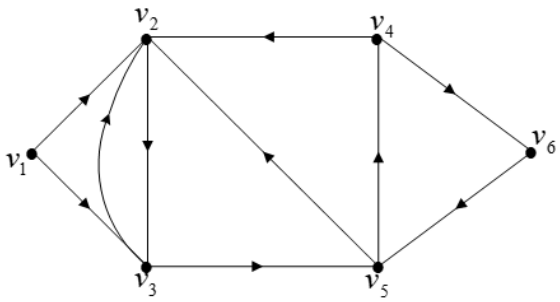


图 1.3: 有向非赋权图

$$W = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

子图

设 $G_1 = (V_1, E_1)$ 与 $G_2 = (V_2, E_2)$ 是两个图，并且满足 $V_1 \subset V_2$，$E_1 \subset E_2$，则称 G_1 是 G_2 的子图，G_2 称为 G_1 的母图。

如 G_1 是 G_2 的子图，且 $V_1 = V_2$，则称 G_1 是 G_2 的生成子图。

1.3 图论中的 Matlab 函数命令

graph：无向图；

digraph：有向图；

G = graph(A)：使用邻接矩阵 A 创建赋权无向图。

G = graph(A, nodes)：使用邻接矩阵 A 和节点名称 nodes 创建赋权无向图。

G = graph(s,t)：使用节点对组 s,t 创建无向图。

G = graph(s, t, weights)：使用节点对组 s,t 和权重向量 weights 创建赋权无向图。

G = graph(s, t, weights, nodes)：使用字符向量元胞数组 nodes 指定节点名称。

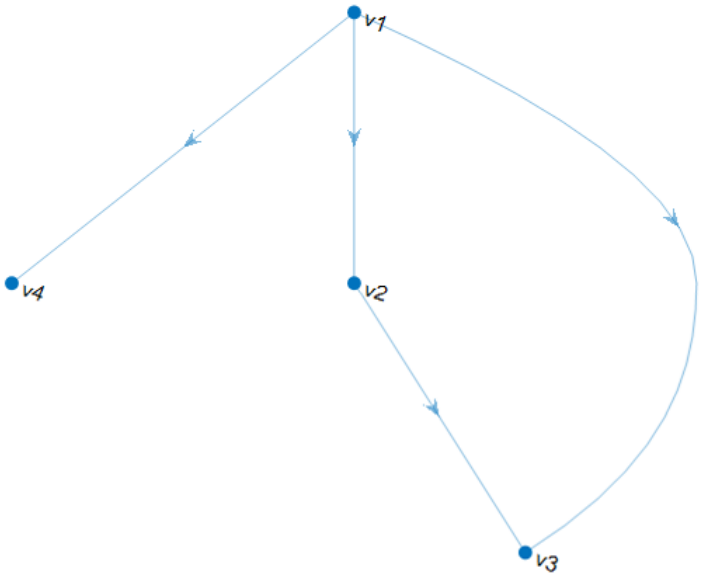
如下面是生成无向图以及增加、删去一些节点和边的代码。

```
clc
clear
s = [1 1 1 2];
t = [2 3 4 3];
G = graph(s, t)
plot(G)                    %绘制以s和t为节点的图
G1 = addnode(G,3)          %增加3个节点
figure(2);
plot(G1)
G2 = rmnode(G, [3 5 6])    %删除节点3、5、6
figure(3);
plot(G2)
G3 = addedge(G, [1 2], [3 4]) %增加两条边
figure(4);
plot(G3)
G4 = rmedge(G,1,3)          %删除由节点1和3连成的边
figure(5);
plot(G4)
```

如下面是生成赋权有向图的代码。

```
clc
clear
s = [1 1 1 2];
t = [2 3 4 3];
weights = [10 20 30 40];
names = \{'v1' 'v2' 'v3' 'v4'\};
G = digraph(s, t, weights, names);
plot(G)
```

生成的赋权有向图如下所示。



1.4 最短路问题

道路 设 $W = v_0e_1v_1e_2\cdots e_kv_k$ ，其中

$$e_i \in E,\ i = 1,2,\ldots,k;\ v_j \in V,\ j = 0,1,\ldots,k$$

e_i 与 v_{i-1} 和 v_i 关联，称 W 是图 G 的一条道路，简称路，k 为路长，v_0 为起点，v_k 为终点。

连通图

在无向图 G 中，如果从顶点 u 到顶点 v 存在道路，则称顶点 u 和 v 是连通的。如果图 G 中的任意两个顶点 u 和 v 都是连通的，则称图 G 是连通图，否则称为非连通图。

连通分支

非连通图中的连通子图，称为连通分支。

强连通图 在有向图 D 中，如果对于任意两个顶点 u 和 v，从 u 到 v 和从 v 到 u 都存在道路。

路的长度 设图 G 是赋权图，Γ 为 G 中的一条路。称 Γ 的各边权之和为路 Γ 的长度。

最短路 对于连通的赋权图 G 的两个顶点 u_0 和 v_0 ，从 u_0 到 v_0 的路一般不止一条，其中长度最小的一条称为从 u_0 到 v_0 的最短路。

最短路问题是图论应用的基本问题，很多实际问题，如线路的布设、运输安排、运输网络最小费用等问题，都可通过建立最短路问题模型来求解。哥本哈根大学计算机科学家 Mikkel Thorup 米克尔·索鲁普表示：“最短路径是个绝妙的好问题，全世界人都能感同身受。”

求解最短路问题的两种方法：Dijkstra 算法和 Floyd 算法。

1.5 Dijkstra 算法

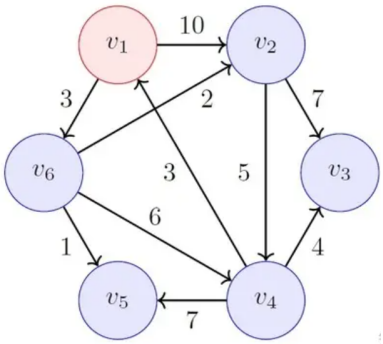
求赋权图中从给定点到其余顶点的最短路用 Dijkstra 算法，该算法由荷兰计算机科学家狄克斯特拉于 1959 年所提出。

Dijkstra 算法的基本思想是：采用贪心算法的策略，从起点开始，逐步扩展到周围的点，直到到达目标点。每次扩展的点都是当前距离起点最近的点。Dijkstra 算法描述如下表所示。

集合 S：存储已经找到的最短路径。

Step 1. 将图上的初始点 v1 加至最短路径集合 S，其它点看作另一个集合。
Step 2. 根据初始点，求出其它点到初始点的距离，若无直接相连的边，则定义距离为无穷大。
Step 3. 选取最小的距离，并将此边所对应的点（记为 x）加入集合 S。
Step 4. 更新与 x 相邻点的距离值。
Step 5. 重复步骤 3 和 4，直到所有点都加至集合 S。注意步骤 3 是从 S 的所有路径中寻找距离。

例题 1.4 用 Dijkstra 算法计算 v_1 到各节点的最小路径。



寻找最优路径的步骤如下。

步骤 1. v_1 到自身的路径最短，先添加进集合 S。

步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	3

步骤 2. 从 v_1 出发，计算 v_1 到其它节点的距离，若无法连接则用 ∞。找到最小值 v_6，加至集合 S 中。后面不再考虑 v_6。

步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	$\times$
Step 2.	$v_1 \to v_6$	5	∞	9	4	

步骤 3. 从 $v_1 \to v_6$ 出发，计算到其它节点的距离。找到最小值 v_5，加至集合 S 中。后面不再考虑 v_5。

步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	$\times$
Step 2.	$v_1 \to v_6$	5	∞	9	4	
Step 3.	$v_1 \to v_6 \to v_5$	∞	∞	∞	$\times$	

步骤 4. 从 $v_1 \to v_6 \to v_5$ 出发，计算到其它节点的距离。除去 v_5 和 v_6，找到其它列的最小值 5，对应的路径为：$v_1 \to v_6 \to v_2$，添加最短路径 $v_1 \to v_6 \to v_2$。后面不再考虑 v_2 列。

步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	$\times$
Step 2.	$v_1 \to v_6$	5	∞	9	4	
Step 3.	$v_1 \to v_6 \to v_5$	∞	∞	∞	$\times$	
Step 4.	$v_1 \to v_6 \to v_2$	$\times$	12	10		

步骤 5. 从 $v_1 \to v_6 \to v_2$ 出发，计算 v_1 通过 v_6 及 v_2 到其它节点的距离。除去 v_5、v_6 和 v_2，找到其它列的最小值 9，对应的路径为：$v_1 \to v_6 \to v_4$。添加最短路径 $v_1 \to v_6 \to v_4$。后面不再考虑 v_4 列。

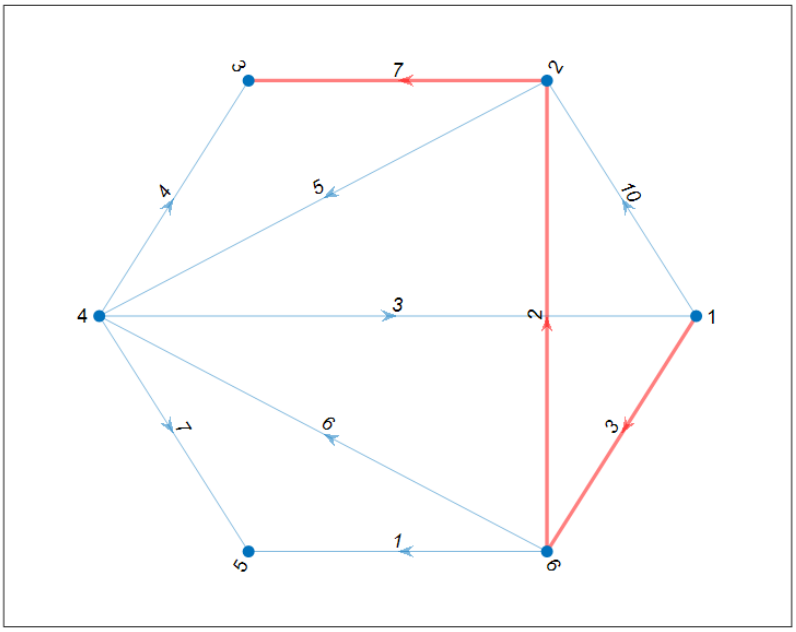
步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	$\times$
Step 2.	$v_1 \to v_6$	5	∞	9	4	
Step 3.	$v_1 \to v_6 \to v_5$	∞	∞	∞	$\times$	
Step 4.	$v_1 \to v_6 \to v_2$	$\times$	12	10		
Step 5.	$v_1 \to v_6 \to v_4$		13	$\times$		

步骤 6. 从 $v_1 \to v_6 \to v_4$ 出发，计算 v_1 通过 v_6 及 v_4 到其它节点的距离。除去 v_5、v_6、v_2 和 v_4 所在的列，找到最小的值 12，对应 $v_1 \to v_6 \to v_2 \to v_3$ 路径最短，添加至 S，后面不再考虑 v_3 列。

步骤	S	v_2	v_3	v_4	v_5	v_6
Step 1.	v_1	10	∞	∞	∞	$\times$
Step 2.	$v_1 \to v_6$	5	∞	9	4	
Step 3.	$v_1 \to v_6 \to v_5$	∞	∞	∞	$\times$	
Step 4.	$v_1 \to v_6 \to v_2$	$\times$	12	10		
Step 5.	$v_1 \to v_6 \to v_4$		13	$\times$		
Step 6.	$v_1 \to v_6 \to v_2 \to v_3$		$\times$			

在利用 MATLAB 工具箱计算时采用稀疏矩阵，如果两个顶点之间没有边，对应的邻接矩阵元素为 0，而不是像数学理论上对应的邻接矩阵元素为 inf 或 0。工具箱为 Dijkstra 标号算法。下面的代码给出了寻找从 v_1 到 v_3 的最优路径命令。

```
clc
clear
clf
E = [1,2,10; 1,6,3; 2,3,7; 2,4,5; 4,1,3; 4,3,4; 4,5,7; 6,2,2; 6,4,6; 6,5,1];
G = digraph(E(:,1), E(:,2), E(:,3));
[path, d] = shortestpath(G, 1, 3, 'method','positive')
p = plot(G,'EdgeLabel',G.Edges.Weight,'Layout', 'circle');
highlight(p,path,'EdgeColor','r','LineWidth',1.5)
```



1.5.1 Dijkstra 算法的发展

排序瓶颈设定了算法速度的根本限制。1984 年，Tarjan 和另一位研究者在学术论文 “Optimization Algorithms Fibonacci Hleaps and ”heir Uses in lmproved Network” 中，改进了迪杰斯特拉的原始算法，使其达到了这个速度极限。任何进一步的改进都必须来自一个避免排序的算法。

在 20 世纪 90 年代末和 21 世纪初，Thorup 和其他研究者设计出了打破排序瓶颈的算法。

2022 年，清华大学毕业的学者段然在其论文 “A Randomized Algorithm for Single-Source Shortest Path on Undirected Real-Weighted Graphs” 中，提出了一种算法，打破了任意权重下的排序瓶颈，但仅适用于所谓无向图。在无向图中，每条连接线都可以双向通行。计算机科学家通常更关注包含单向路径的更广义的图类，但这些「有向图」往往更难处理。

2025 年，段然突破了 Dijkstra 算法瓶颈。在其论文 “Breaking the Sorting Barrier for Directed Single-Source Shortest Paths” 中提出了将图分层处理，像 Dijkstra 算法那样从源头向外推进的方法。该方法结合了另一经典的算法：贝尔曼-福特算法。其精妙之处在于定位关键节点后优先探索，稍后回溯处理其他边界节点。由于不严格按距离顺序探索每层节点，排序障碍自然失效。若采用恰当的分层策略，其速度甚至略超优化版迪杰斯特拉算法。随着排序障碍的攻克，新算法的运行效率已远超现有理论极限。

Dijkstra 算法能否求任意两个顶点对之间的最短路？

具体方法如下。每次以不同的顶点作为起点，用 Dijkstra 算法求出从该起点到其余顶点的最短路径，反复执行 $n-1$（n 为顶点个数）次这样的操作，就可得到每对顶点之间的最短路。

缺点：重复计算效率低，为此，引入下面求解所有顶点对之间最短路的 Floyd 算法。

Floyd 算法

求赋权图中任意两点间的最短路用 Floyd 算法，也称为 Floyd-Warshall 算法、Roy-Warshall 算法、Roy-Floyd 算法或 WFI 算法。该算法由 1978 年图灵奖获得者、斯坦福大学计算机科学系教授罗伯特·弗洛伊德命名，是一种利用动态规划的思想寻找给定的加权图中多源点之间最短路径的算法。Floyd 算法允许赋权图中包含负权的边或弧，但对于赋权图中的每个圈 C，要求圈 C 上所有弧的权总和为非负。

Dijkstra 算法通过选定的被访问顶点，求出从出发访问顶点到其他顶点的最短路径；Floyd 算法中每一个顶点都是出发访问点，所以需要将每一个顶点看做被访问顶点，求出从每一个顶点到其他顶点的最短路径。

Floyd 算法包含三个关键算法：求距离矩阵、求路径矩阵、最短路查找算法。

设所考虑的赋权图 $G = (V, E, A)$，其中顶点集 $V = \{v_1, v_2, \ldots, v_n\}$，邻接矩阵

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

其中，

$$a_{ij} = \begin{cases} v_i\text{与}v_j\text{间边的权值}, & \text{当}v_i\text{与}v_j\text{之间有边} \\ \infty, & \text{当}v_i\text{与}v_j\text{之间无边时} \\ 0, & i = j \end{cases}$$

对于无向图，A 为对称矩阵，$a_{ij} = a_{ji}, i, j = 1, 2, \cdots, n$。

1.5.2 求距离矩阵

Floyd 算法基本思想：结合了 “贪心” 和 “动态规划” 的思想，通过通过遍历图中的每个点，以每个点为 “中转站”，不断更新所有其他点到所有点的最短距离。具体地，递推产生一个矩阵序列：

$$A_1, A_2, \ldots, A_n,$$

其中，矩阵 $A_k = (a_k(i,j))_{n\times n}$ 的第 i 行第 j 列元素 $a_k(i,j)$ 表示从顶点 v_i 到 v_j 的路径上所经过的顶点序号不大于 k 的最短路径长度，k 表示迭代次数。

对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 和 w 再到 v，比已知的路径更短。如果是更新它，即第 k 次的值为：

$$a_k(i,j) = \min\{a_{k-1}(i,j), a_{k-1}(i,k) + a_{k-1}(k,j)\}.$$

当 $k = n$ 时，A_n 即是各顶点之间的最短距离值。

1.5.3 求路径矩阵

在距离矩阵 A_k 的迭代过程中，引入一个路由矩阵 $R_k = (r_k(i,j))_{n\times n}$ 来记录两点间路径的前驱后继关系，其中 $r_k(i,j)$ 表示从顶点 v_i 到顶点 v_j 的路径经过编号为 $r_k(i,j)$ 的顶点。

路径矩阵的迭代过程如下：

（1）初始时

$$R_0 = 0_{n\times n}$$

（2）迭代公式为

$$R_k = (r_k(i,j))_{n\times n}$$

其中

$$r_k(i,j) = \begin{cases} k, & a_{k-1}(i,j) > a_{k-1}(i,k) + a_{k-1}(k,j), \\ r_{k-1}(i,j), & \text{其他} \end{cases}.$$

迭代到 $k = n$，算法终止。

1.5.4 最短路的路径查找

查找 v_i 到 v_j 最短路径的方法如下：

若 $r_n(i,j) = p_1$，则点 v_{p_1} 是顶点 v_i 到 v_j 的最短路的中间点，然后用同样的方法再分头查找。

（1）向顶点 v_i 反向追踪得：

$$r_n(i,p_1) = p_2,\ r_n(i,p_2) = p_3,\ \ldots,\ r_n(i,p_s) = 0.$$

（2）向顶点 v_j 正向追踪得：

$$r_n(p_1,j) = q_1,\ r_n(q_1,j) = q_2,\ \ldots, r_n(q_t,j) = 0.$$

则由点 v_i 到 v_j 的最短路径为：

$$v_i, v_{p_s}, \ldots, v_{p_2}, v_{p_1}, v_{q_1}, v_{q_2}, \ldots, v_{q_t}, v_j$$

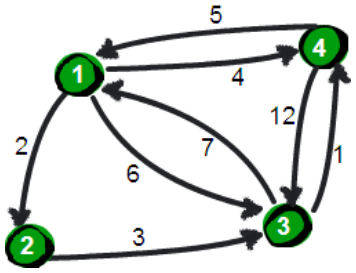
1.5.5 Floyd 算法

求距离矩阵 $D = (d_{ij})_{n\times n}$ 和路径矩阵 $R = (r_{ij})_{n\times n}$ 的 Floyd 算法如下。

Step 1. 初始化：
$k = 0$，令 $d_{ij} = a_{ij}$, $r_{ij} = 0$, $i, j = 1, 2, \ldots, n$。
Step 2. 迭代：
$k = k + 1$，若 $d_{ij} > d_{ik} + d_{kj}$，令
$d_{ij} = d_{ik} + d_{kj}$, $r_{ij} = k$, $i, j = 1, 2, \ldots, n$. 否则，d_{ij} 和 r_{ij} 不更新。
Step 3. 算法终止：
条件：$k = n$；否则，转 Step 2。

时间复杂度：$O(n^3)$，三重循环。
空间复杂度：$O(n^2)$，存储两个 n 阶矩阵。

例题 1.5 求如下图中任意一点到另一点的最短路径。



创建距离矩阵：

$$A=\begin{pmatrix}0&2&6&4\\ \infty&0&3&\infty\\ 7&\infty&0&1\\ 5&\infty&12&0\end{pmatrix}$$

创建路径矩阵：v_i 到 v_j 的最短路径中，v_i 的下一个点，默认为-1。

$$R=\begin{pmatrix}1&2&3&4\\ -1&2&3&-1\\ 1&-1&3&4\\ 1&-1&3&4\end{pmatrix}$$

步骤 1. 顶点 1 为中间点

2→1→3 距离 $\infty+6>3$，不更新表格。

2→1→4 距离 $\infty+4=\infty$，不更新表格。

3→1→2 距离 $7+2=9<\infty$，更新表格。

3→1→4 距离 $7+4=11>1$，不更新表格。

4→1→2 距离 $5+2=7<\infty$，更新表格。

4→1→3 距离 $5+6=11<12$，更新表格。

$$A=\begin{pmatrix}0&2&6&4\\ \infty&0&3&\infty\\ 7&9&0&1\\ 5&7&11&0\end{pmatrix},\ R=\begin{pmatrix}1&2&3&4\\ -1&2&3&-1\\ 1&1&3&4\\ 1&1&1&4\end{pmatrix}$$

步骤 2. 顶点 2 为中间点

1→2→3 距离 $2+3=5<6$，更新表格。

1→2→4 距离 $2+\infty=\infty>4$，不更新表格。

3→2→1 距离 $9+\infty=\infty>7$，不更新表格。

3→2→4 距离 $9+\infty=\infty>1$，不更新表格。

4→2→1 距离 $7+\infty=\infty>5$，不更新表格。

4→2→3 距离 $7+3=10<11$，更新表格。

$$A=\begin{pmatrix}0&2&5&4\\ \infty&0&3&\infty\\ 7&9&0&1\\ 5&7&10&0\end{pmatrix},\ R=\begin{pmatrix}1&2&2&4\\ -1&2&3&-1\\ 1&1&3&4\\ 1&1&2&4\end{pmatrix}$$

步骤 3. 顶点 3 为中间点

1→3→2 距离 $5+9=14>2$，不更新表格。
1→3→4 距离 $5+1=6>4$，不更新表格。
2→3→1 距离 $3+7=10<\infty$，更新表格。
2→3→4 距离 $3+1=4<\infty$，更新表格。
4→3→1 距离 $10+7=17>5$，不更新表格。
4→3→2 距离 $10+9=19>7$，不更新表格。

$$A=\begin{pmatrix}0&2&5&4\\10&0&3&4\\7&9&0&1\\5&7&10&0\end{pmatrix},\ R=\begin{pmatrix}1&2&2&4\\3&2&3&3\\1&1&3&4\\1&1&2&4\end{pmatrix}$$

步骤 4. 顶点 4 为中间点
1→4→2 距离 $4+7=11>2$，不更新表格。
1→4→3 距离 $4+10=14>5$，不更新表格。
2→4→1 距离 $4+5=9<10$，更新表格。
2→4→3 距离 $4+10=14>3$，不更新表格。
3→4→1 距离 $1+5=6<7$，更新表格。
3→4→2 距离 $1+7=8<9$，更新表格。

$$A=\begin{pmatrix}0&2&5&4\\9&0&3&4\\6&8&0&1\\5&7&10&0\end{pmatrix},\ R=\begin{pmatrix}1&2&2&4\\4&2&3&3\\4&4&3&4\\1&1&2&4\end{pmatrix}$$

运行代码如下。

```
A = [0 2 6 4; inf 0 3 inf; 7 inf 0 1; 5 inf 12 0];
[D, path] = floyd(a)

% floyd函数文件
function [D,path,min1,path1]=floyd(a,start,terminal)
D=a;n=size(D,1);path=zeros(n,n);
for i=1:n
   for j=1:n
      if D(i,j)~=inf
         path(i,j)=j;
      end
   end
end

for k=1:n
   for i=1:n
      for j=1:n
         if D(i,k)+D(k,j)<D(i,j)
            D(i,j)=D(i,k)+D(k,j);
            path(i,j)=path(i,k);
         end
      end
   end
end
if nargin==3
   min1=D(start,terminal);
   m(1)=start;
   i=1;
   path1=[ ];
   while  path(m(i),terminal)~=terminal
      k=i+1;
      m(k)=path(m(i),terminal);
      i=i+1;
   end
   m(i+1)=terminal;
   path1=m;
end
```

1.6 模型的应用

参见优秀论文 2020-B-03。