Graph and Network Algorithm

Mathematical Modeling

Prof. Dr. Jingzhi Li

Department of Mathematics,
Southern University of Science and Technology

2025 Spring

1. 图
2. 图与网络的表示方法
3. 最短路径问题
4. MATLAB 的其他图论函数

基本概念

在平面上 n 个点，把其中的一些点对用曲线或直线连接起来，不考虑点的位置与连线长短，这样形成的一个关系结构就是一个图（Graph）。记为

$$G = (V, E)$$

其中 V 是以上述点为元素的点集，E 是以上述连线为集合的边集。

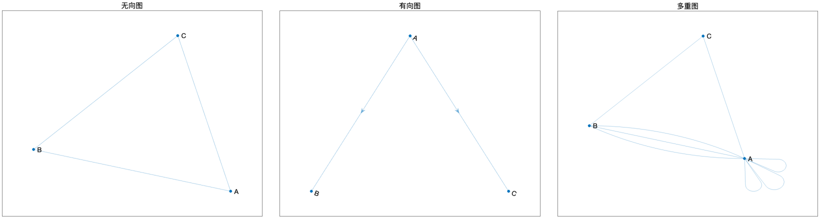
图的类型

- 有向图：边有方向
- 无向图：边无方向
- 混合图：有的边有方向，有的边无方向
- 简单图：任意两点间最多一条边，且无自环
- 多重图：允许多条边连接同一对点，或存在自环
- 权重图：边带有权重，例如距离、成本

创建有向图和无向图

在 MATLAB 中，graph 和 digraph 函数用于构建表示无向图和有向图的对象。

- 使用 ismultigraph 函数判断是否为多重图



图对象

`graph` 函数会返回无向图对象，`digraph` 函数会返回有向图对象

- G.Edges：图的边，以表的形式返回
  - G.Edges.EndNodes：每条边的两个端点
  - G.Edges.Weight：边的权重的数值向量
- G.Nodes：返回一个表，其中列出图的节点属性。默认情况下此表为空

修改图的节点和边

节点

- addnode：添加新节点
- rmnode：从图中删去节点
- numnodes：返回节点数量

边

- addedge：添加新边
- rmedge：从图中删去边
- numedges：返回边数量
- flipedge：反转边的方向

邻接矩阵表示法

假设 $G=(V,E)$ 是一个简单的无向图，顶点集合 $V=\{v_1,\cdots,v_n\}$，边集 $E=\{e_1,\cdots,e_m\}$。邻接矩阵记为

$$W=(w_{ij})_{n\times n}$$

当 G 为赋权图时，有：

$$w_{ij}=\begin{cases}\text{权值} & \text{当 } v_i \text{ 与 } v_j \text{ 之间有边时}\\ 0 \text{ 或} \infty & \text{当 } v_i \text{ 与 } v_j \text{ 之间无边时}\end{cases}$$

当 G 为非赋权图时，有：

$$w_{ij}=\begin{cases}1 & \text{当 } v_i \text{ 与 } v_j \text{ 之间有边时}\\ 0 & \text{当 } v_i \text{ 与 } v_j \text{ 之间无边时}\end{cases}$$

稀疏矩阵表示法

当图的边数 m 远小于顶点数 n 时，邻接矩阵大多数元素都是 0，会造成很大的空间浪费。所以我们使用稀疏矩阵表示，只记录非零元素位置和数值，节省空间并提高计算效率。

- `sparse` 函数能将普通邻接矩阵转为稀疏邻接矩阵
- `full` 函数将稀疏邻接矩阵转为普通邻接矩阵
- `adjacency` 函数能返回图的稀疏邻接矩阵
- `adjacency(G,'weighted')` 返回图的加权稀疏矩阵

1. 图
2. 图与网络的表示方法
3. 最短路径问题
4. MATLAB 的其他图论函数

最短路问题是重要的最优化问题之一，也是图论研究中的一个经典算法问题。最短路问题一般被归为两类：

- 求从某个顶点到其他顶点的最短路径
- 求图中每一个顶点间的最短路径

最短路算法主要以 Dijkstra 算法和 Floyd 算法为基础。

Dijkstra 算法

使用 Dijkstra 算法，可以寻找图中节点之间的最短路径。特别是，可以在图中寻找一个节点到所有其它节点的最短路径，生成一个最短路径树。

- Dijkstra 只能用在权重为正的图中，如果图中有负权重的边，可能出现多绕路反而路线更短的情况，不合实际。

算法步骤：

1. 初始化
   - 将起点到自身距离设为 0，其他节点设为 ∞
   - 所有节点标记为“未访问”
2. 重复以下步骤直到所有节点被访问：
   - 选出当前距离起点最近的未访问节点 u
   - 遍历其所有邻居 v，更新路径：

$$\text{If } d[v] > d[u] + w(u,v), \text{ then } d[v] = d[u] + w(u,v)$$

   - 将 u 标记为已访问

Dijkstra 算法的 MATLAB 实现

某公司在六个城市 $c_1, c_2, \ldots, c_6$ 中有分公司，从 c_i 到 c_j 的直接航程票价记在下述矩阵的 (i, j) 位置上（∞ 表示无直接航路）。请帮助该公司设计一张从城市 c_1 到其他城市间票价最低的路线图。

$$\begin{bmatrix} 0 & 50 & \infty & 40 & 25 & 10 \\ 50 & 0 & 15 & 20 & \infty & 25 \\ \infty & 15 & 0 & 10 & 20 & \infty \\ 40 & 20 & 10 & 0 & 10 & 25 \\ 25 & \infty & 20 & 10 & 0 & 55 \\ 10 & 25 & \infty & 25 & 55 & 0 \end{bmatrix}$$

通过 dijkstra 函数得到的结果为

d = [0,35,45,35,25,10]

index1 = [1,6,5,2,4,3]

index2 = [1,6,5,6,1,1]

通过 shortestpath 函数得到的结果为

P = [1,6,2]

Floyd 算法

对于任意一个顶点 $v_k \in V$，顶点 v_i 到顶点 v_j 的最短路径经过顶点 v_k，或者不经过顶点 v_k。

比较 d_{ij} 与 $d_{ik} + d_{kj}$ 的值。若 $d_{ij} > d_{ik} + d_{kj}$，则令

$$d_{ij} = d_{ik} + d_{kj}$$

并保持 d_{ij} 是当前搜索的顶点 v_i 到顶点 v_j 的最短距离。

重复这个过程，最后当搜索完所有顶点 v_k 时，d_{ij} 就是顶点 v_i 到顶点 v_j 的最短距离。

Floyd 算法基本步骤

令 d_{ij} 是顶点 v_i 到顶点 v_j 的最短距离，w_{ij} 是顶点 v_i 到 v_j 的权。Floyd 算法的步骤是：

1. 输入图 G 的权矩阵 W。对所有 i,j，有 $d_{ij}=w_{ij}$，$k=1$。
2. 更新 d_{ij}。对所有 i,j，若 $d_{ik}+d_{kj}<d_{ij}$，则令

$$d_{ij}=d_{ik}+d_{kj}$$

3. 若 $d_{ii}<0$，则存在一条含有顶点 v_i 的负回路，停止；或 $k=n$ 停止，否则转到步骤 2。

shortestpathtree

`shortestpathtree` 函数用于生成从节点的最短路径树

- `TR = shortestpathtree(G,s)`：返回有向图 TR，包含了从源节点 s 到图中所有其他节点的最短路径树
- `TR = shortestpathtree(G,s,t)`：计算多个源或目标节点之间的最短路径树
- `[TR,D,E] = shortestpathtree()`
  - D 返回各节点之间的最短距离
  - E 返回逻辑向量 E，指示图中的每条边是否在 TR 中

distances

`distances` 函数返回所有节点对组的最短路径距离

- `d = distances(G)`: 返回矩阵 d，其中 d(i,j) 是节点 i 和节点 j 之间的最短路径的长度
- `d = distances(G,s,t)`： d(i,j) 是从节点 s(i) 到节点 t(j) 的距离

allpaths

`allpaths` 函数查找两个节点之间的所有路径

- `[paths,edgepaths] = allpaths(G,s,t)`：返回从 `s` 到 `t` 的每条路径中的边。输出 edgepaths 是元胞数组，其中 edgepathsk 给出沿对应路径 pathsk 的边。

课堂练习

在许多实际问题中，不仅需要求出图的最短路，而且还要找到它的次短路，顾名思义，就是相对于最短路而言，“退而求其次”的路。显然，相同始点、终点的相同距离的次短路不止一条，而在实际中，往往与最短路至少有一条边或弧不同的“次短路”非常重要。

1. 方法一
    - 先求出最短路径
    - 枚举删除路径上的每条边
    - 修改图后重新求最短路径
    - 取长度最小的作为次短路径
2. 方法二；改进 Dijkstra 算法