数据结构与算法

Data Structures and Algorithms

谢昊

xiehao@cuz.edu.cn

非线性结构
Non-Linear Structures

大纲

1. 基本术语

2. 图的存储结构

3. 小结

引例：Königsberg 七桥问题

- 在 Königsberg 市有 7 座桥连通了 4 块区域
- 是否有算法实现
  - 从某处出发
  - 依次穿过所有桥仅 1 次
  - 回到原地

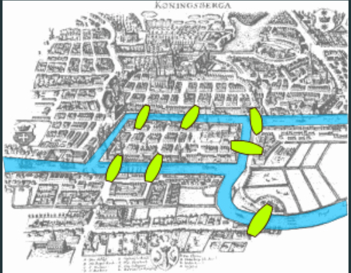


图 **1:** Königsberg 七桥问题

引例：Königsberg 七桥问题

- 在 Königsberg 市有 7 座桥连通了 4 块区域
- 是否有算法实现
  - 从某处出发
  - 依次穿过所有桥仅 1 次
  - 回到原地
- 可抽象为图的一笔画问题

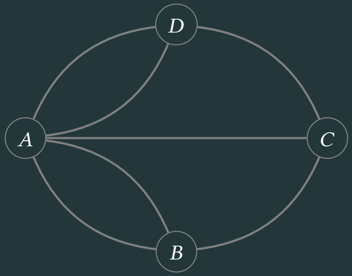


图 1: Königsberg 七桥问题

非线性结构

- 在半线性结构的基础上允许有环的存在
- 半线性结构的一种扩展
- 非线性结构主要指**图**结构
- 图与树之间可转换

基本术语

基本术语

图 (Graph)

- 可被定义为 $G = (V, E)$，其中[1]
  - 集合 V 中元素 v 为**顶点 (vertex)**
  - 集合 E 中元素 $e \in V \times V$ 为**边 (edge)**
- 只定义拓扑关系，与几何位置无关

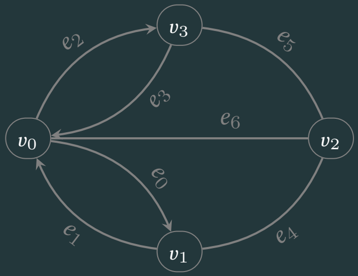


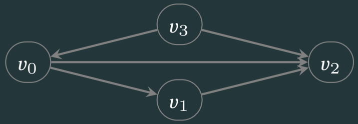
图 2: 图

[1]顶点又名**结点 (node)**，边又名**弧 (arc)**，且 V 与 E 均为有限集

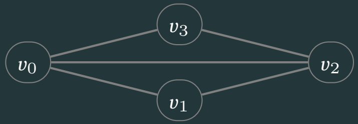
基本术语

有向图 (Undigraph) 与无向图 (Digraph)

- 按顶点是否有顺序可将边 e 分为
  - 无顺序的**无向边**，记作 (u,v)
  - 有顺序的**有向边**，记作 $\langle u,v \rangle$
- 只有无向边的图为**无向图**
- 只有有向边的图为**有向图**
- 二者均有的图为**混合图 (mixed graph)**
- 无向图与混合图均可转化为有向图
  - 将一条无向边拆成两条相反的有向边



(a) 有向图

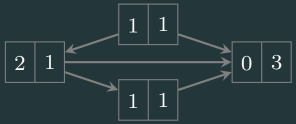


(b) 无向图

图 3: 有向图与无向图

顶点的度 (Degree)

- 若边 $e = \langle v_a, v_b \rangle$，则
  - 称 v_a 与 v_b **邻接 (adjacent)**
  - 称二者均与 e 彼此**关联 (incident)**
  - 称 e 为 v_a 的**出边 (outgoing edge)**
  - 称 e 为 v_b 的**入边 (incoming edge)**
- 在无向图中称与顶点 v 关联的边数为 v 的**度**
- 在有向图中称与顶点 v 关联的出入边数分别为 v 的**出度 (out-degree)** 与**入度 (in-degree)**



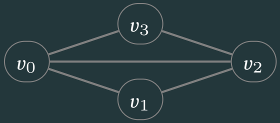
(a) 有向图的出度（左）与入度（右）

(b) 无向图的度

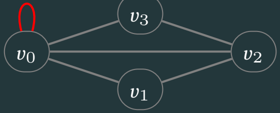
图 4: 顶点的度

简单图 (Simple Graph)

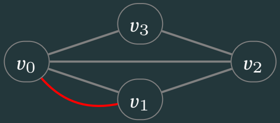
- 称起点与终点相同的边为**自环 (self-loop)**
- 称不含自环且所有边均唯一的图为**简单图**[2]



(a) 简单图



(b) 带自环的非简单图



(c) 有重复边的非简单图

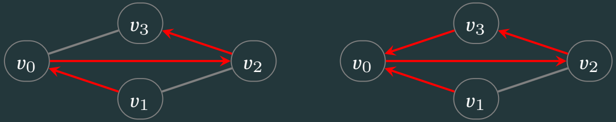
图 5: 简单图与非简单图

[2] 本课程只讨论简单图

基本术语

路径 (Path)

- 若序列 $\pi = \{v_k\}_{k=0}^{n}$ 满足 v_k 与 v_{k+1} 邻接[3]，则称之为自 v_0 至 v_n 的一条**路径**
  - 称经过的总边数 $|\pi|$ 为路径**长度 (length)**
  - 称无重复顶点的路径为**简单 (simple) 路径**
- 两顶点间的路径一般不唯一



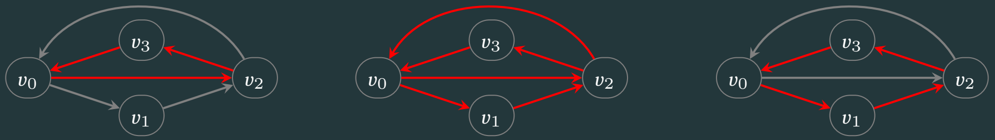
(a) 简单路径：(v_1, v_0, v_2, v_3) **(b)** 非简单路径：$(v_1, v_0, v_2, v_3, v_0)$

图 6: 图的路径

[3]指存在边 e_k 满足 $e_k = (v_k, v_{k+1})$ 或 $e_k = \langle v_k, v_{k+1} \rangle$， $k \in \mathbb{Z} \cap [0, n)$

环路 (Cycle)

- 若路径 $\pi = \{v_k\}_{k=0}^{n}$ 中起止顶点相同，即 $v_0 = v_n$，则称其为**环路**
  - 若除起止结点相同外无任何其他结点两两相同，则称其为 **简单环路**
  - 称经过图中各边一次且仅一次的环路为**欧拉环路 (Eulerian tour)**
  - 称经过图中各顶点一次且仅一次的环路为 **哈米尔顿环路 (Hamiltonian tour)**

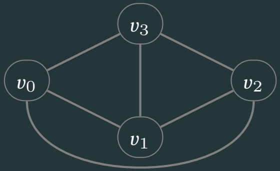


(a) 简单环路：(v_0, v_2, v_3) **(b)** 欧拉环路：$(v_0, v_1, v_2, v_0, v_2, v_3, v_0)$ **(c)** 哈米尔顿环路：$(v_0, v_1, v_2, v_3, v_0)$

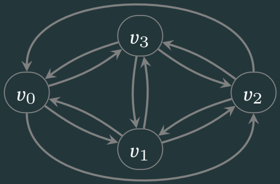
图 7: 图的环路

完全图 (Complete Graph)

- 图中任意两顶点均邻接
- 若顶点数为 n 则无向与有向边数分别为 $\frac{n(n-1)}{2}$ 与 $n(n-1)$



(a) 完全无向图

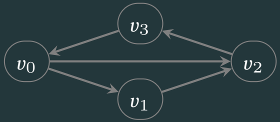


(b) 完全有向图

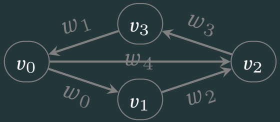
图 8: 完全图

带权图 (Weighted Graph)

- 为每条边指定权重，又名**带权网络 (network)**
- 用于表示顶点关系的细节，如长度、流量、成本等
- 普通图可看作所有边权重均为 1 的带权图



(a) 普通图



(b) 带权图

图 9: 普通图与带权图

图的存储结构

图的抽象数据类型

ADT Graph {

数据:

数据对象: $\mathcal{D} = \{v_k | v_k \in$ 顶点集合$, k \in \mathbb{Z} \cap [1, n]\}$

逻辑关系: $\mathcal{R} = \{\langle v_x, v_y \rangle | \exists e_x \in$ 边集合$, s.t.\ e_x = \langle v_x, v_y \rangle\}$

操作:

`create_graph(), destroy_graph(g)`

构造与销毁一个图g

`get_vertex(g, v), get_first_neighbor(g, v), get_next_neighbor(g, v, w)`

返回顶点v的信息，获取顶点v的第一个邻接顶点与相对于w的下一个邻接顶点

`insert_vertex(g, v), remove_vertex(g, v)`

插入与删除顶点v

`insert_edge(g, va, vb), remove_edge(g, va, vb)`

在顶点v_a与v_b间插入与删除一条边

`depth_first_search(g, x), breadth_first_search(g, x)`

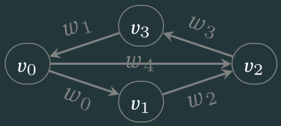
对图g进行深度与广度优先搜索值x

}

图的存储结构

邻接矩阵 (Adjacency Matrix)

- 图抽象数据类型的一种基本实现
- 用稠密方阵表示，其元素描述一对顶点间可能的邻接关系
    - 若有边相连，则元素为该边权重；否则可为 ∞ 或 0



(a) 带权图

$$
\begin{array}{c}
\begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\
\begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \end{array}
\begin{pmatrix}
\infty & w_0 & w_4 & \infty \\
\infty & \infty & w_2 & \infty \\
\infty & \infty & \infty & w_3 \\
w_1 & \infty & \infty & \infty
\end{pmatrix}
\end{array}
$$

(b) 带权图的邻接矩阵

图 10: 邻接矩阵

图的存储结构

邻接矩阵特点

- 无向图邻接矩阵必对称，故可只存储上三角部分
- 有向图邻接矩阵第 k行/列非零元素个数为对应顶点的 出/入度
- 增删边只需修改矩阵特定元素值
- 增删顶点需调整矩阵维度，导致大量元素移动

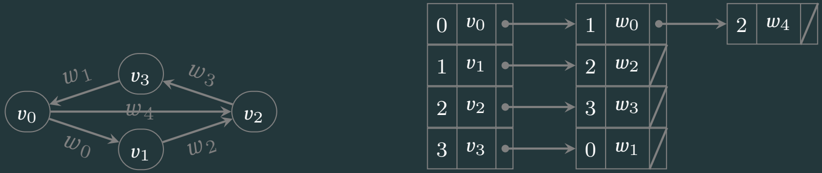
复杂度

- 空间复杂度：$O(n^2)$，不利于表达稀疏图 (sparse graph)[4]
- 查找特定边的时间复杂度：$O(1)$

[4]指边数远小于完全图边数的图

邻接表 (Adjacency List)

- 每个顶点以链表存储其邻接顶点集合
  - 链表结点存储信息包括：顶点序号、边权重、下一个邻接顶点
- 相当于只存储邻接矩阵中的有效元素



(a) 带权图　　**(b)** 带权图的邻接表

图 11: 邻接矩阵

邻接表特点

- 只存储邻接顶点信息，可大幅节省空间
- 增删边或顶点只需修改极少量数据

复杂度

- 空间复杂度：$O(n+e)$[5]
- 查找特定边的时间复杂度：$O(n)$

[5] n 与 e 分别为顶点数与边数

小结

小结

-

问与答