数据结构与算法期末复习
Homework

尹超

中国科学院大学，北京 100049

Carter Yin

University of Chinese Academy of Sciences, Beijing 100049, China

2024.8 – 2025.1

序言

本文为笔者数据结构与算法的期末复习笔记。
望老师批评指正。

目录

题目 第九章图

151

> 任何两个顶点间都有一条（无向）边的图称为完全图，包含 n 个顶点的完全图用 Kn 表示。下列哪个图一定不是平面图？A. K2 B. K3 C. K4 D. K5

Solution 1. 正确答案是 D。

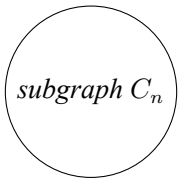
详细分析：

一个图是**平面图**（*Planar Graph*），如果它可以在一个平面上被画出来，且没有任何两条边相互交叉（除了在顶点处相交）。

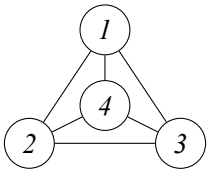
- ***A. K2***: 包含 *2* 个顶点和 *1* 条边。它是一条线段，显然是平面图。

  1 —— *2*

- ***B. K3***: 包含 *3* 个顶点和 *3* 条边。它是一个三角形，显然是平面图。



- ***C. K4***: 包含 *4* 个顶点和 *6* 条边。它可以被画成一个带两条对角线的正方形。虽然对角线会交叉，但我们可以重新绘制它，使得边不交叉（例如，将一个顶点放在其他三个顶点构成的三角形内部）。因此，*K4* 是平面图。



- ***D. K5***: 包含 *5* 个顶点和 $C(5,2)=10$ 条边。根据图论中的一个基本结论（库拉托夫斯基定理），***K5 是典型的非平面图***。无论如何绘制，它的边都必然会发生交叉。

  我们可以用欧拉公式的推论来证明这一点：对于任何一个顶点数 $v \geq 3$ 的连通平面图，其边数 e 和顶点数 v 必须满足不等式 $e \leq 3v-6$。

  - 对于 *K5*，我们有 $v=5$，$e=10$。
  - 将这些值代入不等式：$10 \leq 3(5)-6$。
  - 计算结果为：$10 \leq 15-6$，即 $10 \leq 9$。
  - 这个不等式显然是**不成立**的。

  因为 *K5* 不满足平面图所必须满足的条件，所以 *K5* 一定不是平面图。

152

> 在包含 n 个顶点的用邻接矩阵实现的图中，顶点 v 有 m 个邻居，遍历所有 m 个邻居的时间复杂度为：A. O(1) B. O(m) C. O(n) D. O(mn)

Solution 2. 正确答案是 *C*。

详细分析：

***(1)* 邻接矩阵的结构：** 一个包含 *n* 个顶点的图的邻接矩阵是一个 $n \times n$ 的二维数组，我们称之为 *'Adj'*。如果顶点 *'i'* 和顶点 *'j'* 之间有边，则 *'Adj[i][j]'* 为 *1*（或权重），否则为 *0*。

***(2)* 如何查找邻居：** 要找到顶点 *'v'* 的所有邻居，我们必须检查邻接矩阵中对应于 *'v'* 的那**一整行**，即 *'Adj[v][0], Adj[v][1], ..., Adj[v][n-1]'*。

***(3)* 时间复杂度分析：**

- 这一行总共有 *'n'* 个元素，对应图中所有的 *'n'* 个顶点。
- 我们需要遍历这一行中的所有 *'n'* 个元素，来判断哪些位置的值为 *1*（表示是邻居）。
- 无论顶点 *'v'* 有多少个实际的邻居（即 *'m'* 的值是多少），这个遍历过程都必须检查 *'n'* 次。
- 因此，遍历顶点 *'v'* 所有邻居的时间复杂度取决于顶点的总数 *'n'*，而不是邻居的数量 *'m'*。

***(4)* 与其他选项的对比：**

- *B. O(m):* 这个是使用**邻接表**实现图时，遍历邻居的时间复杂度。因为邻接表只存储了实际存在的边。
- *A. O(1)* 和 *D. O(mn):* 均不符合邻接矩阵的遍历特性。

结论： 在用邻接矩阵表示的图中，查找一个特定顶点的所有邻居，需要扫描该顶点对应的完整一行，其时间复杂度为 $O(n)$。

153

> 图 G 包含 n 个顶点 (n>0)，用邻接矩阵实现。在其中加入一个新的顶点后邻接矩阵增加了多少项？A. n+1 B. 2n C. 2n-1 D. 2n+1

Solution 3. 正确答案是 *D*。

详细分析：

***(1)* 初始状态：**

- 图 *G* 有 *'n'* 个顶点。
- 其邻接矩阵是一个 *'n x n'* 的方阵。
- 矩阵中的总项数（元素个数）为 $n \times n = n^2$。

***(2)* 加入新顶点后的状态：**

- 图 *G* 现在有 *'n + 1'* 个顶点。
- 其邻接矩阵变为一个 *'(n+1) x (n+1)'* 的方阵。
- 新矩阵中的总项数为 $(n+1) \times (n+1) = (n+1)^2$。

***(3)* 计算增加的项数：** 增加的项数等于新矩阵的总项数减去旧矩阵的总项数。

- 增加的项数 = $(n+1)^2 - n^2$
- 展开 $(n+1)^2$：$n^2 + 2n + 1$
- 计算差值：$(n^2 + 2n + 1) - n^2 = 2n + 1$

直观理解： 当从一个 $n \times n$ 矩阵变为一个 $(n+1) \times (n+1)$ 矩阵时，相当于增加了一行和一列。

- 增加的行有 *'n+1'* 个元素。

- 增加的列有 *'n+1'* 个元素。
- 其中，新行和新列的交叉点 *'A[n+1][n+1]'* 被计算了两次，所以要减去 *1*。
- 总增加量 = *(*新行的元素数*)* + *(*新列的元素数*)* - *(*重复计算的 *1* 个元素*)* = $(n+1)+(n+1)-1=2n+2-1=2n+1$。

因此，邻接矩阵增加了 *'2n + 1'* 项。

154

> 图的广度优先搜索访问各顶点的模式类似于二叉树的：A. 先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历

Solution 4. 正确答案是 *D*。

详细分析：

(1) 图的广度优先搜索 (BFS - Breadth-First Search):

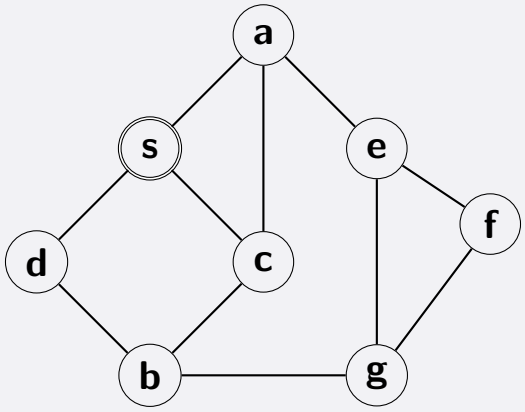
- *BFS* 从一个起始顶点开始，首先访问所有与起始顶点直接相邻的顶点。
- 然后，再逐一访问与这些邻接点相邻的、尚未被访问过的顶点。
- 这个过程一层一层地向外扩展，就像水波纹一样。
- *BFS* 通常使用一个队列（*Queue*）数据结构来实现，以保证"先发现的顶点，其邻居也先被探索"的顺序。

(2) 二叉树的遍历方式：

- ***A, B, C (先序、中序、后序遍历):*** 这三种都是**深度优先遍历 *(DFS - Depth-First Search)*** 的特例。它们会沿着一条路径尽可能深地访问下去，直到末端，然后再回溯。这与 *BFS* 的逐层扩展模式完全不同。
- ***D. 层次遍历 (Level-Order Traversal):*** 层次遍历从树的根节点开始，按照从上到下、从左到右的顺序，逐层访问树中的所有节点。它也是使用队列来实现的。

结论：图的广度优先搜索（*BFS*）和二叉树的层次遍历在思想和实现上是完全一致的。它们都是逐层地访问节点，并且都依赖于队列来管理待访问的节点顺序。因此，*BFS* 的访问模式类似于二叉树的层次遍历。

155

> 以顶点 s 为起点对以上无向图进行 BFS，同一顶点的邻居之间以 a z 为顺序，顶点 c 刚出队时队列中顶点从队头到队尾为：
>
> 
>

> A. d, e
> B. e, d
> C. d, a
> D. a, d

Solution 5. 正确答案是 *A*。

详细分析：

我们来逐步模拟广度优先搜索（*BFS*）的过程。*BFS* 使用一个队列来管理待访问的顶点。

(1) 初始状态：

- 起点为 ***s***。将 ***s*** 入队。
- 队列：*'[s]'*

(2) 第一次出队：

- 顶点 ***s*** 出队。
- 查找 ***s*** 的所有邻居：*a, c, d*。
- 按字母顺序排序：*a, c, d*。
- 将 *a, c, d* 依次入队。
- 队列：*'[a, c, d]'*

(3) 第二次出队：

- 顶点 ***a*** 出队。
- 查找 ***a*** 的所有邻居：*c, e, s*。
- 按字母顺序排序：*c, e, s*。
- 检查邻居是否已被访问（即是否已在队列中或已出队）：
  - *c:* 已在队列中，跳过。
  - *e:* 未被访问，将 ***e*** 入队。
  - *s:* 已被访问，跳过。
- 队列：*'[c, d, e]'*

(4) 第三次出队：

- 顶点 ***c*** 出队。
- 此时，我们观察队列的状态。
- 队列中剩下的顶点，从队头到队尾依次是：***d, e***。

因此，当顶点 *c* 刚出队时，队列中的顶点为 *d, e*。

156

> 对于用邻接表实现的包含 n 个顶点 e 条边的图，BFS 的时间复杂度为：
> A. O(n)
> B. O(n^2)
> C. O(n+e)
> D. O(n^2+e)

Solution 6. 正确答案是 *C*。

详细分析：

广度优先搜索（*BFS*）在邻接表实现的图上的时间复杂度可以从两个主要部分来分析：

***(1)* 顶点操作：**

- *BFS* 需要访问图中的每一个顶点。
- 每个顶点都会被入队（*enqueue*）一次和出队（*dequeue*）一次。
- 假设有 *'n'* 个顶点，这部分操作的总时间复杂度是 $O(n)$。

***(2)* 边操作：**

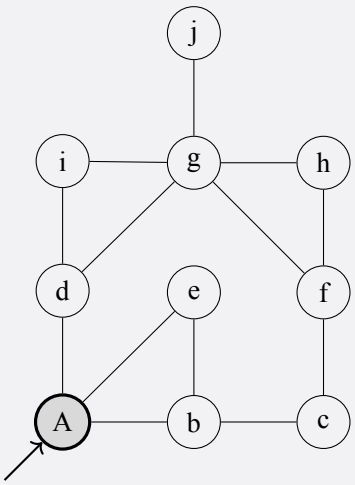
- 当一个顶点 *'v'* 从队列中出队时，*BFS* 会遍历 *'v'* 的邻接表，以找到所有与它相邻的顶点。
- 在整个 *BFS* 的执行过程中，每个顶点的邻接表都会被完整地遍历一次。
- 在邻接表表示法中，所有邻接表的长度之和等于图的总边数 *'e'*（对于有向图）或 *'2e'*（对于无向图）。无论哪种情况，这部分操作的总时间复杂度都与边的数量成正比，即 $O(e)$。

结论： 将顶点操作和边操作的时间复杂度相加，得到 *BFS* 在邻接表上实现的总时间复杂度为 $O(n) + O(e) = O(n + e)$。

（注：如果图是用邻接矩阵实现的，由于查找每个顶点的邻居都需要扫描一行，总时间复杂度会是 $O(n^2)$。）

157

以 A 为起点对以下无向图进行 DFS，同一顶点的邻居以 a z 为序，各顶点被访问的顺序为：



A. a, e, b, c, f, g, h, j, i, d
B. a, b, c, f, g, d, i, h, j, e
C. a, b, e, c, f, h, g, i, d, j
D. a, e, b, c, f, h, g, i, d, j

Solution 7. 正确答案是 *B*。

详细分析：

我们使用递归（或栈）来模拟深度优先搜索（*DFS*）的过程。首先，列出每个顶点的邻居，并按字母顺序排序：

- ***A**: b, d, e*
- ***b**: A, c, e*
- ***c**: b, f*

- ***d**: A, g, i*
- ***e**: A, b*
- ***f**: c, g, h*
- ***g**: d, f, h, i, j*
- ***h**: f, g*
- ***i**: d, g*
- ***j**: g*

***DFS* 步骤：**

(1)* 访问 *A。邻居是 *b, d, e*。选择第一个未访问的 ***b***。

(2)* 访问 *b。邻居是 *A, c, e*。*A* 已访问，选择第一个未访问的 ***c***。

(3)* 访问 *c。邻居是 *b, f*。*b* 已访问，选择第一个未访问的 ***f***。

(4)* 访问 *f。邻居是 *c, g, h*。*c* 已访问，选择第一个未访问的 ***g***。

(5)* 访问 *g。邻居是 *d, f, h, i, j*。*f* 已访问，选择第一个未访问的 ***d***。

(6)* 访问 *d。邻居是 *A, g, i*。*A, g* 已访问，选择第一个未访问的 ***i***。

(7)* 访问 *i。邻居是 *d, g*。*d, g* 都已访问。回溯到 *d*。

(8) 从 *d* 回溯到 *g*。

(9) 在 *g*，邻居是 *d, f, h, i, j*。*d, f, i* 已访问，选择下一个未访问的 ***h***。

(10)* 访问 *h。邻居是 *f, g*。*f, g* 都已访问。回溯到 *g*。

(11) 在 *g*，邻居是 *d, f, h, i, j*。*d, f, h, i* 已访问，选择下一个未访问的 ***j***。

(12)* 访问 *j。邻居是 *g*。*g* 已访问。回溯到 *g*。

(13) 从 *g* 回溯到 *f*，再到 *c*，再到 *b*。

(14) 在 *b*，邻居是 *A, c, e*。*A, c* 已访问，选择下一个未访问的 ***e***。

(15)* 访问 *e。邻居是 *A, b*。*A, b* 都已访问。回溯到 *b*。

(16) 从 *b* 回溯到 *A*。

(17) 在 *A*，所有邻居 *b, d, e* 都已被访问。搜索结束。

最终访问顺序为： $A \to b \to c \to f \to g \to d \to i \to h \to j \to e$

158

> u 和 v 为图中两个顶点，对图进行 DFS 后，dTime(u) < dTime(v) < fTime(v) < fTime(u)，则 u 和 v 在 DFS 森林中的关系是：A. 属于不同的连通分量 B. u 为 v 的祖先 C. v 为 u 的祖先 D. 互为兄弟

Solution 8. 正确答案是 *B*。

详细分析：

在图的深度优先搜索（*DFS*）中，每个顶点 *'v'* 都有两个时间戳：

- ***dTime(v)**: 发现时间（Discovery Time），即 'v' 第一次被访问的时间。*
- ***fTime(v)**: 完成时间（Finishing Time），即 'v' 的邻接表被完全扫描完毕，搜索从 'v' 回溯的时间。*

这些时间戳之间存在一个重要的性质，称为**“括号引理”（*Parenthesis Theorem*）**。它描述了任意两个顶点 *'u'* 和 *'v'* 的时间戳区间 *'[dTime, fTime]'* 的关系：

***(1)* 区间不相交：** *'dTime(u) < fTime(u) < dTime(v) < fTime(v)' (或 u,v 交换)*。这表示 *'u'* 和 *'v'* 在 *DFS* 森林中没有祖先-后代关系。它们可能在不同的 *DFS* 树中，也可能在同一棵树但属于不同的分支。

(2) 区间包含： *'dTime(u) < dTime(v) < fTime(v) < fTime(u)'*。这表示 *'v'* 的访问过程完全包含在 *'u'* 的访问过程中。

我们来分析题目给出的条件：*'dTime(u) < dTime(v) < fTime(v) < fTime(u)'*。

- *'dTime(u) < dTime(v)'* 意味着 *'u'* 比 *'v'* 先被发现。
- *'fTime(v) < fTime(u)'* 意味着 *'v'* 比 *'u'* 先完成。

这整个过程可以这样理解：

(1) *DFS* 开始，发现了 *'u'*。

(2) 在探索 *'u'* 的后代时，发现了 *'v'*。

(3) *DFS* 继续探索 *'v'* 的所有后代，并完成了对 *'v'* 的访问。

(4) *DFS* 回溯，最终完成了对 *'u'* 的访问。

这种时间戳的嵌套关系，恰好说明了 *'v'* 是在 *'u'* 的 *DFS* 子树中被发现和完成的。因此，在 *DFS* 森林中，***u* 是 *v* 的祖先**。

其他选项分析：

- *A.* 属于不同的连通分量：时间区间会不相交。
- *C. v* 为 *u* 的祖先：时间关系应为 *'dTime(v) < dTime(u) < fTime(u) < fTime(v)'*。
- *D.* 互为兄弟：时间区间会不相交，例如 *'dTime(u) < fTime(u) < dTime(v) < fTime(v)'*。

159

> 对同一个无向图分别运行广度优先算法和深度优先算法，得到的树边数量：A. 广度优先的树边更 B. 深度优先的树边更多 C. 两种算法得到的树边一样多 D. 数量关系不确定

Solution 9. 正确答案是 *C*。

详细分析：

(1) 遍历与生成树： 无论是广度优先搜索（*BFS*）还是深度优先搜索（*DFS*），当它们在一个图上运行时，都会将图的边分为两类：树边（*Tree Edge*）和非树边。树边是那些在遍历过程中连接到一个未访问顶点的边。所有树边集合起来，会构成一个**生成树**（*Spanning Tree*）或**生成森林**（*Spanning Forest*）。

(2) 生成树的性质：

- 对于一个有 *'n'* 个顶点的**连通图**，其任何一个生成树都恰好包含 *'n-1'* 条边。这是连接 *'n'* 个顶点而又不形成环路所需的最少边数。
- 对于一个有 *'n'* 个顶点和 *'k'* 个连通分量的**非连通图**，其生成森林由 *'k'* 棵生成树组成，总边数为 *'n-k'*。

(3) 比较 *BFS* 和 *DFS*：

- 对一个连通图运行 *BFS*，会访问所有 *'n'* 个顶点，并产生一个包含 *'n-1'* 条树边的 *BFS* 树。
- 对同一个连通图运行 *DFS*，同样会访问所有 *'n'* 个顶点，并产生一个包含 *'n-1'* 条树边的 *DFS* 树。
- 即使图不连通，两个算法也都会访问所有 *'n'* 个顶点，并为 *'k'* 个连通分量分别生成树，总边数都是 *'n-k'*。

结论： 虽然 *BFS* 和 *DFS* 生成的生成树（或森林）的**结构**（形状）通常是不同的（*BFS* 树通常更“矮胖”，*DFS* 树通常更“瘦高”），但它们包含的**树边数量**是完全相同的。这个数量只取决于图本身的顶点数和连通分量数，与所使用的遍历算法无关。

160

> 对图进行DFS, 以下哪种情况意味着该图包含环路A. 有TREE边B. 有BACKWARD边C. 有FORWARD边D. 有CROSS边

Solution 10. 正确答案是 *B*。

详细分析:

在深度优先搜索（*DFS*）过程中，图中的边可以根据其连接的顶点的状态（发现时间、完成时间）分为四类:

(1)* 树边 *(TREE edge): 在 *DFS* 森林中连接父子节点的边。这是第一次发现一个新顶点时经过的边。树边本身不构成环路。

(2)* 后向边 *(BACKWARD edge): 连接一个顶点 *'u'* 到其在 *DFS* 树中的一个**祖先** *'v'* 的边。当 *DFS* 从顶点 *'u'* 探索到其邻居 *'v'*，而 *'v'* 已经被访问过且尚未完成（即 *'v'* 是 *'u'* 的祖先，仍在当前的递归调用栈中）时，就发现了一条后向边。这条后向边 *'(u, v)'* 与 *DFS* 树中从 *'v'* 到 *'u'* 的路径共同构成了一个**环路**。

(3)* 前向边 *(FORWARD edge): 连接一个顶点 *'u'* 到其在 *DFS* 树中的一个**后代** *'v'* 的边（非树边）。

(4)* 横跨边 *(CROSS edge): 连接两个没有祖先-后代关系的顶点的边，它们可能在 *DFS* 森林的不同分支或不同的树中。

结论:

- *A. TREE* 边是构成 *DFS* 生成树（森林）的基本骨架，本身是无环的。
- ***B. BACKWARD* 边是存在环路的决定性标志。**发现一条后向边就意味着找到了一个环。
- *C. FORWARD* 边和 *D. CROSS* 边（主要在有向图中）并不指向当前路径上的祖先节点，因此它们本身不构成环路。

因此，在 *DFS* 中，判断一个图是否包含环路的标准方法就是检查是否存在后向边。

161

> 在含20个顶点的简单无向图中, 边的数量最多为: _____ 此时度最小的顶点的度为: _____

Solution 11. 详细分析:

第一部分: 边的数量最多为多少?

(1) 一个"简单无向图"是指图中没有自环（一个顶点到自身的边）也没有重边（两个顶点之间有多于一条边）。

(2) 当一个简单无向图的边数最多时，它是一个**完全图**（*Complete Graph*），记作 K_n。在完全图中，任意两个不同的顶点之间都恰好有一条边。

(3) 对于一个包含 n 个顶点的图，边的数量就是从这 n 个顶点中选取 *2* 个顶点进行组合的数量。

(4) 计算公式为: $C(n,2) = \frac{n(n-1)}{2}$。

(5) 在本题中，$n = 20$，所以边的最大数量为: $C(20,2) = \frac{20\times(20-1)}{2} = \frac{20\times19}{2} = 10 \times 19 = 190$。

第二部分：此时度最小的顶点的度为多少？

(1) “此时”指的是在边数最多的情况下，即在完全图 K_{20} 中。

(2) 在一个完全图 K_n 中，每个顶点都与其他所有 $n-1$ 个顶点相连。

(3) 因此，每个顶点的度（*degree*）都是 $n-1$。

(4) 在本题中，$n=20$，所以 K_{20} 中每个顶点的度都是 $20-1=19$。

(5) 因为所有顶点的度都是 *19*，所以度最小的顶点的度也是 *19*。

在含 *20* 个顶点的简单无向图中，边的数量最多为：_____***190***_____ 此时度最小的顶点的度为：_____***19***_____

162

> 某宴会一共有 7 个人参加，与会者之间进行了亲切的握手。已知他们中的每个人进行握手的次数分别为：3, 1, 2, 2, 3, 1, 2 请问宴会上总共发生了多少次握手？_____

Solution 12. 详细分析：

这个问题可以利用图论中的**握手引理（*Handshaking Lemma*）**或**度数总和公式**来解决。

(1) **建立图模型：**

- 我们可以将每个参加宴会的人看作图中的一个**顶点**（*Vertex*）。
- 两个人之间的一次握手可以看作连接对应顶点的**边**（*Edge*）。
- 每个人握手的次数就是对应顶点的**度**（*Degree*）。

(2) **应用握手引理：**握手引理指出：图中所有顶点的度数之和等于图中边数的两倍。

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

其中，$|E|$ 是边的总数，也就是握手的总次数。

(3) **计算度数总和：**根据题目给出的数据，我们将所有人的握手次数相加：

$$3+1+2+2+3+1+2=14$$

(4) **计算握手总次数：**根据公式，度数总和是握手次数的两倍。因为每一次握手都会给两个人各增加 *1* 的度数，所以总度数是握手次数的两倍。

- $2 \times$ (握手总次数) $= 14$
- 握手总次数 $= 14/2 = 7$

请问宴会上总共发生了多少次握手？_____***7***_____

163

> 对于包含 n 个顶点 e 条边的简单无向图，以下关于它的邻接矩阵 A 的说法中错误的是：A. A 有 n 行 e 列，其中元素取值于 0, 1 B. A 的第 k 行中 1 的个数等于顶点 k 的度 C. A = A^T D. A 中位于第 u 行 v 列的元素为 1 当且仅当顶点 u 和顶点 v 邻接

Solution 13. 错误的是 A。

详细分析：

- ***A. A 有 n 行 e 列，其中元素取值于 {0, 1}*** 这个说法是**错误**的。对于一个包含 *'n'* 个顶点的图，其邻接矩阵 *'A'* 是一个 ***'n x n'*** 的方阵 **，而不是 *'n x e'* 的矩阵。它的行和列都对应于图的顶点。
- ***B. A 的第 k 行中 1 的个数等于顶点 k 的度*** 这个说法是**正确**的。在邻接矩阵中，*'A[k][j] = 1'* 表示顶点 *'k'* 和顶点 *'j'* 之间有一条边。因此，第 *'k'* 行中所有元素的和（即 *1* 的个数）就等于与顶点 *'k'* 相连的边的数量，这正是顶点 *'k'* 的度（*degree*）的定义。
- ***C. $A = A^T$*** 这个说法是**正确**的。由于图是无向的，顶点 *'u'* 和 *'v'* 之间的邻接关系是相互的。即，如果 *'u'* 和 *'v'* 邻接，那么 *'v'* 和 *'u'* 也邻接。这反映在矩阵上就是 *'A[u][v] = A[v][u]'*。一个矩阵满足这个条件时，它就是对称矩阵，即等于其自身的转置（*'$A = A^T$'*）。
- ***D. A 中位于第 u 行 v 列的元素为 1 当且仅当顶点 u 和顶点 v 邻接*** 这个说法是**正确**的。这正是邻接矩阵（对于无权图）的**定义**。

164

> 用邻接矩阵实现含 n 个顶点 e 条边的图，其空间复杂度为：A. O(1) B. O(n) C. O(nlogn) D. O(n^2)

Solution 14. 正确答案是 D。

详细分析：

***(1)* 邻接矩阵的定义：** 对于一个包含 *'n'* 个顶点的图，其邻接矩阵是一个 *'n x n'* 的二维数组（方阵）。矩阵的行和列都由图的顶点索引。

***(2)* 空间需求：**

- 这个 *'n x n'* 的矩阵总共包含 $n \times n = n^2$ 个元素。
- 无论图中实际有多少条边（*'e'* 的值是多少），我们都需要为这 n^2 个元素分配存储空间。
- 每个元素存储一个值（例如 *0* 或 *1*，或者边的权重），占用固定的空间。

***(3)* 复杂度分析：**

- 空间复杂度衡量的是算法或数据结构所需的存储空间随输入规模变化的趋势。
- 在这里，输入规模由顶点数 *'n'* 决定。
- 所需空间与 n^2 成正比，因此空间复杂度为 $O(n^2)$。

结论： 邻接矩阵的空间复杂度只与顶点的数量 *'n'* 有关，与边的数量 *'e'* 无关。即使图是稀疏的（边的数量远小于 n^2），它仍然需要 $O(n^2)$ 的空间，这是它相对于邻接表（空间复杂度为 $O(n+e)$）的主要缺点之一。

165

> G 是简单无向图，A 为 G 的邻接矩阵，M 为 G 的关联矩阵，D 是对角线上第 i 个元素为顶点 i 的度的对角矩阵，它们的关系是：A. A=M B. A+D=MM^T C. A+D=M^TM D. 没有直接关系

Solution 15. 正确答案是 B。

详细分析：

我们来分析矩阵乘积 MM^T 的结果。设图 G 有 n 个顶点和 e 条边。

- ***A (邻接矩阵)***: 是一个 *'n x n'* 的矩阵。
- ***M (关联矩阵)***: 是一个 *'n x e'* 的矩阵。*'M(i, j) = 1'* 如果顶点 *'i'* 是边 *'j'* 的一个端点，否则为 *0*。
- ***D (度数矩阵)***: 是一个 *'n x n'* 的对角矩阵。
- ***M^T***: 是 *'M'* 的转置，为一个 *'e x n'* 的矩阵。

乘积 MM^T 是一个 $(n \times e) \times (e \times n)$ 的矩阵，结果是一个 $n \times n$ 的矩阵。我们来计算这个结果矩阵的每一个元素 $[MM^T]_{i,j}$。

(1) 对角线元素 (i = j): $(MM^T)[i][i]$ 是 M 的第 i 行与 M^T 的第 i 列（即 M 的第 i 行）的点积。

$$(MM^T)[i][i] = \sum_{k=1}^{e} M[i][k] \cdot M[i][k] = \sum_{k=1}^{e} (M[i][k])^2$$

因为 $M[i][k]$ 的值只能是 *0* 或 *1*，所以 $(M[i][k])^2$ 的值也只能是 *0* 或 *1*。这个求和操作实际上是在计算有多少条边 k 以顶点 i 为端点。这个数量正是顶点 i 的度数 $\deg(i)$。因此，MM^T 的对角线元素 $(MM^T)[i][i] = \deg(i) = D[i][i]$。

(2) 非对角线元素 ($i \neq j$): $(MM^T)[i][j]$ 是 M 的第 i 行与 M^T 的第 j 列（即 M 的第 j 行）的点积。

$$(MM^T)[i][j] = \sum_{k=1}^{e} M[i][k] \cdot M[j][k]$$

乘积项 *'M[i][k] * M[j][k]'* 等于 *1*，当且仅当 *'M[i][k]'* 和 *'M[j][k]'* 都等于 *1*，这意味着顶点 *'i'* 和顶点 *'j'* 都是边 *'k'* 的端点。

- 如果顶点 i 和 j 之间有一条边，那么在求和中恰好有一项为 *1*，所以 $(MM^T)_{i,j} = 1$。这与邻接矩阵 $A_{i,j}$ 的值相等。
- 如果顶点 i 和 j 之间没有边，那么在求和中所有项都为 *0*，所以 $(MM^T)_{i,j} = 0$。这也与邻接矩阵 $A_{i,j}$ 的值相等。

因此，MM^T 的非对角线元素 $[MM^T]_{i,j} = A[i][j]$。

结论： 综合以上两点，矩阵 MM^T 的对角线部分等于度数矩阵 D，非对角线部分等于邻接矩阵 A。所以，$MM^T = A + D$。这个关系在图论和谱图理论中非常重要。矩阵 *'L = D - A'* 被称为图的拉普拉斯矩阵，而 *'A + D'* 被称为无符号拉普拉斯矩阵。

166

> G 是有向无环图, (u, v) 是 G 中的一条由 u 指向 v 的边。对 G 进行 DFS 的结果是: A. dTime(u) > dTime(v) B. dTime(u) < dTime(v) C. fTime(u) > fTime(v) D. fTime(u) < fTime(v)

Solution 16. 正确答案是 C。

详细分析：

这是一个关于有向无环图（*DAG*）在深度优先搜索（*DFS*）中时间戳性质的重要结论，这个结论是拓扑排序算法的基础。

我们分析当 *DFS* 在访问顶点 *'u'* 并遇到边 *'(u, v)'* 时，顶点 *'v'* 可能处于的状态：

(1) 情况 1：v 是“白色”的（未被发现）

- 此时，*DFS* 会通过边 *'(u, v)'* 第一次访问 *'v'*。*'v'* 成为 *'u'* 在 *DFS* 树中的一个后代。
- 根据括号引理，后代的访问区间 *'[dTime(v), fTime(v)]'* 完全包含在祖先的访问区间 *'[dTime(u), fTime(u)]'* 之内。
- 这意味着 *'dTime(u) < dTime(v) < fTime(v) < fTime(u)'*。
- 在这种情况下，显然 *'fTime(u) > fTime(v)'*。

(2) 情况 2：*v* 是“灰色”的（已被发现但未完成）

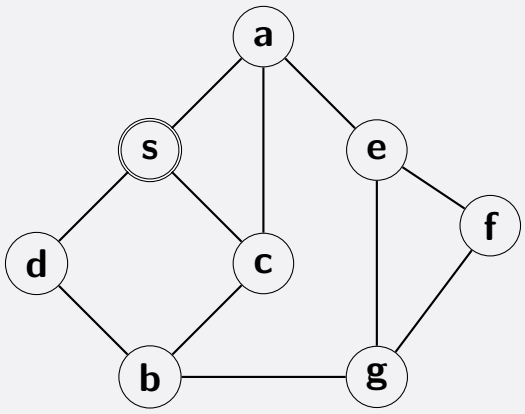
- 如果 *‘v‘* 是灰色的，意味着 *‘v‘* 是 *‘u‘* 在 *DFS* 树中的一个祖先。
- 那么边 *‘(u, v)‘* 就是一条后向边（*Backward Edge*）。
- 后向边的存在意味着图中有一个环路（*‘v -> ... -> u -> v‘*）。
- 这与题目给出的“*G* 是有向无环图”的条件相矛盾。
- 因此，**这种情况在 *DAG* 中不可能发生**。

(3) 情况 3：*v* 是“黑色”的（已完成访问）

- 如果 *‘v‘* 是黑色的，意味着对 *‘v‘* 及其所有后代的访问已经全部完成。
- 这说明 *‘v‘* 的完成时间 *‘fTime(v)‘* 已经确定。
- 而此时对 *‘u‘* 的访问尚未完成（*‘u‘* 还是灰色的），所以 *‘fTime(u)‘* 还没有确定，但它一定大于当前时间。
- 因此，我们有 *‘fTime(v)* < 当前时间 < *fTime(u)‘*。
- 在这种情况下，也必然有 *‘fTime(u) > fTime(v)‘*。

结论：对于有向无环图（*DAG*）中的任意一条边 *‘(u, v)‘*，在 *DFS* 遍历中，其起点的完成时间必定晚于其终点的完成时间，即 *‘fTime(u) > fTime(v)‘*。

167

> 从 s 开始，对以上无向图进行 BFS，同一顶点的邻居之间以 a z 为序，求顶点的 dTime：
>
> 
>
>
> s 的 dTime = 0
> a 的 dTime =_____
> b 的 dTime =_____
> e 的 dTime =_____
> f 的 dTime =_____

Solution 17. 详细分析：

我们模拟广度优先搜索（*BFS*）的过程来确定每个顶点的发现时间（*dTime*）。*dTime* 可以看作是顶点出队的顺序编号，从 *0* 开始。

根据以上 *BFS* 过程：

- 顶点 ***a*** 在第 *2* 步出队，其 *dTime* 为 ***1***。
- 顶点 ***e*** 在第 *5* 步出队，其 *dTime* 为 ***4***。
- 顶点 ***b*** 在第 *6* 步出队，其 *dTime* 为 ***5***。

步骤	出队顶点	dTime	队列状态 (队头 → 队尾)
0 (初始)	-	-	‘[s]‘
1	**s**	**0**	‘[a, c, d]‘ (s 的邻居 a,c,d 按字母序入队)
2	**a**	**1**	‘[c, d, e]‘ (a 的邻居 c,e,s 中，只有 e 未访问，入队)
3	c	2	‘[d, e, b]‘ (c 的邻居 a,b,s 中，只有 b 未访问，入队)
4	d	3	‘[e, b]‘ (d 的邻居 b,s 均已访问)
5	**e**	**4**	‘[b, f, g]‘ (e 的邻居 a,f,g 中，f,g 未访问，按序入队)
6	**b**	**5**	‘[f, g]‘ (b 的邻居 c,d,g 均已访问)
7	**f**	**6**	‘[g]‘ (f 的邻居 e,g 均已访问)
8	g	7	‘[]‘ (g 的邻居 b,e,f 均已访问)

- 顶点***f*** 在第 *7* 步出队，其 *dTime* 为 ***6***。

---

s 的 *dTime = 0*

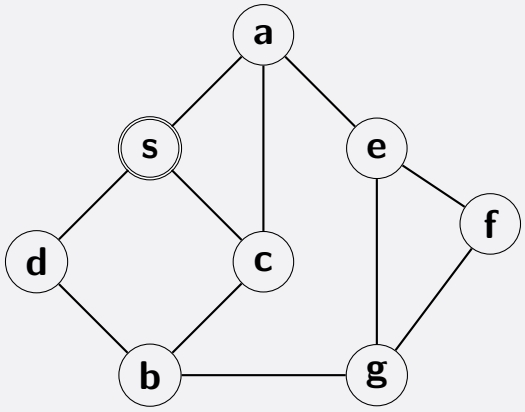
a 的 *dTime* =_____***1***_____

b 的 *dTime* =_____***5***_____

e 的 *dTime* =_____***4***_____

f 的 *dTime* =_____***6***_____

168

> 从 s 开始，对以上无向图进行 DFS，同一顶点的邻居之间以 a z 为序，求各顶点的 dTime 和 fTime
>
> 
>
>
> s 的 dTime =0
> s 的 fTime =15
> c 的 dTime =_____
> c 的 fTime =_____
> g 的 dTime =_____
> g 的 fTime =_____

Solution 18. 详细分析:

我们通过模拟深度优先搜索 (*DFS*) 的递归过程来追踪每个顶点的发现时间 (*dTime*) 和完成时间 (*fTime*)。

一个全局计时器 *'time'* 从 *0* 开始。

***(1) DFS(s)**: 'time=0', 'dTime(s)=0', 'time=1'. s 的邻居 (a,c,d).*

***(2) DFS(a)**: 'time=1', 'dTime(a)=1', 'time=2'. a 的邻居 (c,e,s).*

***(3) DFS(c)**: 'time=2', 'dTime(c)=2', 'time=3'. c 的邻居 (a,b,s).*

***(4) DFS(b)**: 'time=3', 'dTime(b)=3', 'time=4'. b 的邻居 (c,d,g).*

***(5) DFS(d)**: 'time=4', 'dTime(d)=4', 'time=5'. d 的邻居 (b,s) 都已访问. **'fTime(d)=5'**, 'time=6'. 返回 b.*

***(6)** 回到 b. 邻居 (c,d) 已访问.*

***(7) DFS(g)**: 'time=6', 'dTime(g)=6', 'time=7'. g 的邻居 (b,e,f).*

***(8) DFS(e)**: 'time=7', 'dTime(e)=7', 'time=8'. e 的邻居 (a,f,g).*

***(9) DFS(f)**: 'time=8', 'dTime(f)=8', 'time=9'. f 的邻居 (e,g) 都已访问. **'fTime(f)=9'**, 'time=10'. 返回 e.*

***(10)** 回到 e. 邻居 (a,f,g) 都已访问. **'fTime(e)=10'**, 'time=11'. 返回 g.*

***(11)** 回到 g. 邻居 (b,e,f) 都已访问. **'fTime(g)=11'**, 'time=12'. 返回 b.*

***(12)** 回到 b. 邻居 (c,d,g) 都已访问. **'fTime(b)=12'**, 'time=13'. 返回 c.*

***(13)** 回到 c. 邻居 (a,b,s) 都已访问. **'fTime(c)=13'**, 'time=14'. 返回 a.*

***(14)** 回到 a. 邻居 (c,e,s) 都已访问. **'fTime(a)=14'**, 'time=15'. 返回 s.*

***(15)** 回到 s. 邻居 (a,c,d) 都已访问. **'fTime(s)=15'**, 'time=16'. 结束.*

结果汇总：

- *c: dTime = 2, fTime = 13*
- *g: dTime = 6, fTime = 11*

---

s 的 dTime =0

s 的 fTime =15

c 的 dTime = ***2***

c 的 fTime = ***13***

g 的 dTime = ***6***

g 的 fTime = ***11***