EDA 软件设计 I

Lecture 11

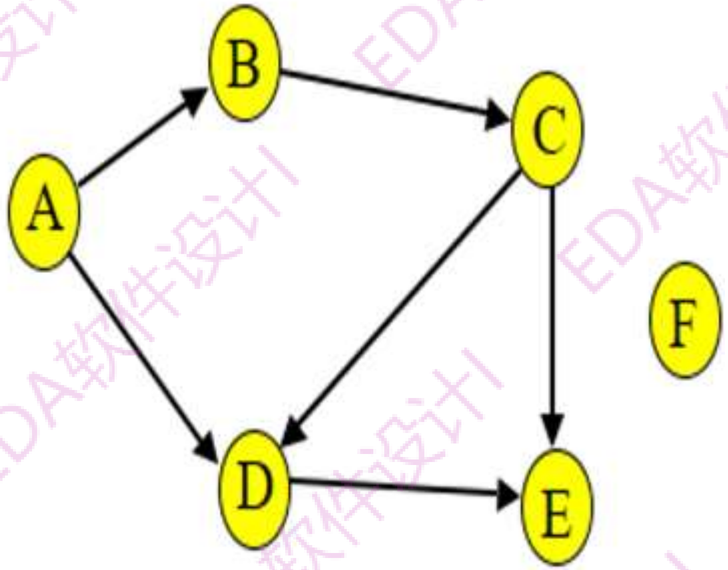
Review:拓扑排序

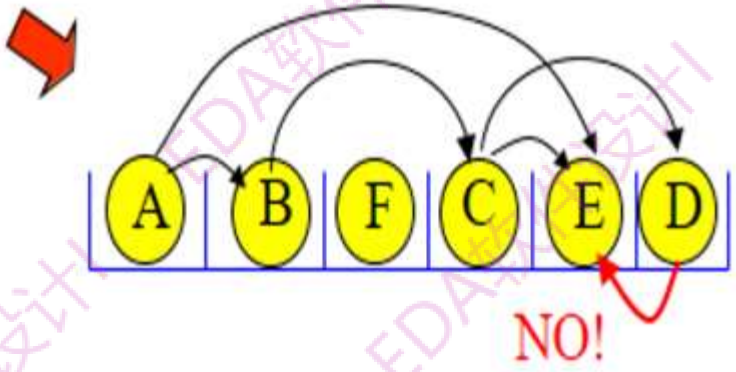
- 我们有**一组任务**和它们之间的**依赖关系/优先约束**
  - 如任务A指向任务B（**A→B**），即约束为**"任务A必须在任务B之前完成"**
- **拓扑顺序**：一种符合给定依赖关系（优先约束）的线性排序
  - 对于任何一组依赖关系，都符合"前序任务出现在后继任务之前"
- **拓扑排序目标**：找到任务的拓扑顺序，或者确定不存在这样的排序

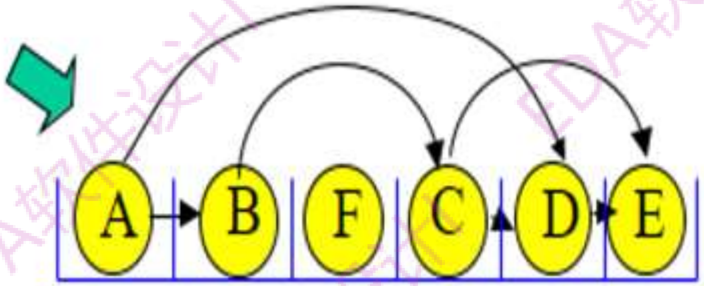
Review:拓扑排序（Formally）

- 假设在有向图 $G=(V,E)$ 中，顶点集合 V 表示任务，每条边 $(u,v)\in E$ 表示任务 u 必须在任务 v 之前完成
- 找到一种顶点的线性排序，使得对于每条边 (u,v)，i.e., $u\rightarrow v$，顶点 u 排在顶点 v 之前
- 这样对顶点的排序称为图 G 的**拓扑排序（Topological Sort)**

Review:拓扑排序（Graphic）







注意：F点的位置

Review:拓扑排序

- **解决的问题：** 是否可以按照一种顺序执行图 G 中的所有任务，并且符合图中的所有优先要求（即每条边的约束）？
- **Claim**："可以"，当且仅当（⇔）有向图 G 中没有环！
  - 如果图中存在环，则会出现"**死锁**"现象，无法找到一个符合所有优先约束的任务执行顺序
- 这样的图 G 称为**有向无环图**（Directed Acyclic Graph, **DAG**）
  - 无环性：对于任意一点，都不存在从该节点出发，经过若干边，再返回到该节点的路径

拓扑排序算法：Kahn（卡恩）算法

- **“基于入度的广度优先搜索”**
- **核心idea:** 先找入度低的点，使其排序在前, 以“入度低”作为**priority**，来构建**基于priority的广度优先搜索**
- 步骤:
  ① **找到所有入度为零的节点**：这些节点没有任何其他节点依赖于它们，因此可以直接放入拓扑排序的结果序列
  - ◆一定存在入度为零的点吗？为什么？如果没有说明什么？

  ② **移除节点及更新入度**：从图中移除该节点，并将与之相邻的所有出边删除; 对于每条出边 (u, v)，将目标节点 v 的入度减 1

  ③ **重复直到处理完所有节点**：不断重复上述过程，直到所有节点都被处理

算法的正确性

1. **形式化证明（Theoretical Proofs）**：通过严格的数学证明来确认算法能够正确地解决其设定的问题
   - 理论保障算法的正确性
2. **测试验证（Empirical Testing）**：尽管形式化证明是算法正确性的理论保障，**测试验证**是工程实践中不可或缺的一环。通过实际测试验证算法在不同输入下的表现，可以确保算法在真实环境中的正确性
   - 用于测试算法的某种实现（implementation）是正确的

算法正确性证明方法（常用）

01

归纳法

方法：数学归纳法, 证明通过两个步骤：

① **基准情况**：证明算法在最简单的输入（如空集合、单一元素）上是正确的

② **归纳步骤**：假设算法在规模与n的输入上是正确的，推导证明它在规模为n+1的输入上也能正确运行

应用：适用于**递归**或**迭代算法**，尤其是**分治算法**

02

循环不变式

方法：用于证明带有循环结构的算法的正确性，循环不变式是一种在每次循环迭代时都保持为真的性质，证明通常分为三个步骤：

① **初始化**：证明不变式在第一次迭代之前成立

② **保持性**：证明如果在某次迭代之前不变式成立，那么在该次迭代之后不变式仍然成立

③ **终止性**:证明当循环终止时，不变式结合终止条件能导出正确的答案

应用：**排序算法（如插入排序、选择排序）**的正确性通常通过循环不变式来证明

算法正确性证明方法（常用）

03

直接证明法

方法：通过**直接逻辑推导**证明算法每一步的正确性：分析算法的每一步，无需依赖递归或迭代，直接证明其正确性

应用：适用于每一步都能通过逻辑推导证明正确的算法

04

矛盾法

方法：**假设算法不正确**，推导出矛盾，从而证明假设错误。

应用：当直接证明较为困难时，通过错误假设推导矛盾来证明正确性

05

反证法

方法：**证明原命题的逆否命题**，从而间接证明原命题正确

应用：当直接证明困难时，通过证明逆否命题来间接证明正确性

Kahn Algorithm 正确性证明（归纳法）

1. **基准状态**：算法首先找到所有入度为 0 的顶点：入度为 0 的顶点没有依赖关系，因此它们可以安全地排在拓扑排序的开头
2. **归纳假设**：假设在某一时刻，已输出的顶点序列是合法的拓扑排序（n）
   - 即所有已经输出的顶点序列S满足图中的依赖关系：对于 $u \in S$，任何存在的边 (u, v) 中，顶点 v 还没有被输出（即 $v \notin S$ ）
3. **归纳步骤**：现在考虑队列中的下一个入度为 0 的顶点 v（n+1）：
   - 根据定义（算法步骤），顶点 v 没有未被输出的依赖顶点：因为所有指向 v 的顶点都已经被处理并输出，因此将 v 输出是合法的
   - 然后，移除顶点 v 以及与之相连的所有出边。如果某个目标顶点的入度因此变为 0，则该顶点可以加入队列，并将在稍后安全地输出
4. **终止条件**：该过程重复，最终所有顶点都被输出，或者如果图中存在环，则有些顶点的入度永远不会变为 0，因此不会被输出

Kahn Algorithm 正确性证明（矛盾法）

A. **假设: 算法的输出不是一个合法的拓扑排序**
   A. 即存在一条边 (u, v)，使得顶点 v 在 u 之前被输出

B. **Contradiction:** 但根据算法的步骤，只有在 u 被输出之后，才会移除与 u 相连的出边 (u, v)，这样才有可能使 v 的入度变为 0，进而被输出。所以，v 不可能在 u 之前被输出，这与假设矛盾

C. 因此，算法输出的顺序必然是合法的拓扑排序

Review:基于DFS的拓扑排序

① Run **DFS** （**for unvisited node)**

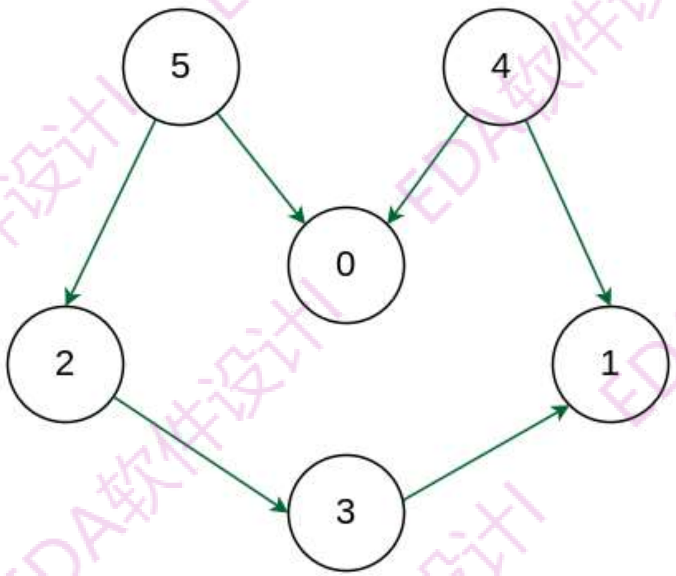
◆对于图中还未访问的节点

② 在DFS中，用递归的方式处理节点，**回溯时记录节点顺序**

◆Note: 区别于DFS遍历，在深入探索时便记录节点顺序

③ 反转（倒置）节点顺序 → 拓扑排序顺序

可利用DFS实现的基础：DFS可以先处理所有依赖的节点（作为前序任务的节点），然后再处理当前节点，从而保证了每对依赖关系在排序结果中都合法



基于DFS的拓扑排序

① Run **DFS** （**for unvisited node)**

◆对于图中还未访问的节点

② 在DFS中，用递归的方式处理节点，**回溯时记录节点顺序**

◆Note: 区别于DFS遍历，在深入探索时便记录节点顺序

③ 反转（倒置）节点顺序 → 拓扑排序顺序

插入到链表前端（代替用Stack记录）： 当每个顶点的深度优先搜索完成时，将该顶点插入到一个链表的前端

◆在DFS完成每个顶点时，将其插入到链表的最前面，这样就可以确保依赖的顶点排在后面

返回链表： 最终，链表中的顶点顺序就是图 G 的拓扑排序

Python重点知识点

- **数据结构：**
  - **列表（List）**：用来存储节点、邻接列表等，掌握列表的基本操作（如 append、pop、indexing）
  - **字典（Dictionary）**：用于表示图的邻接表，以及记录节点的状态（如访问标记），理解键值对的操作是关键
  - **集合（Set）**：用于存储已访问节点，确保算法不会重复访问
  - **队列（Queue）**：在 BFS 中常用，可以使用 collections.deque 实现高效的队列操作
  - **堆栈（Stack）**：在 DFS 的迭代版本中可以使用列表（list）实现堆栈的功能
- **模块（module）**
  - **collections 模块**：了解 deque、defaultdict，以及如何使用这些高效的数据结构。
  - **heapq 模块**：如果要处理优先队列，可以用 heapq 模块来实现（在某些变种算法中，BFS 可能需要优先队列
- **技巧：**
  - **列表推导式（List Comprehensions）**：能使代码更简洁，如在初始化图的邻接表时
  - **字典推导式（Dictionary Comprehension）**：用来简洁地构建字典
  - **生成器（Generators）**：在遍历图的节点时，可以使用生成器来优化内存的使用