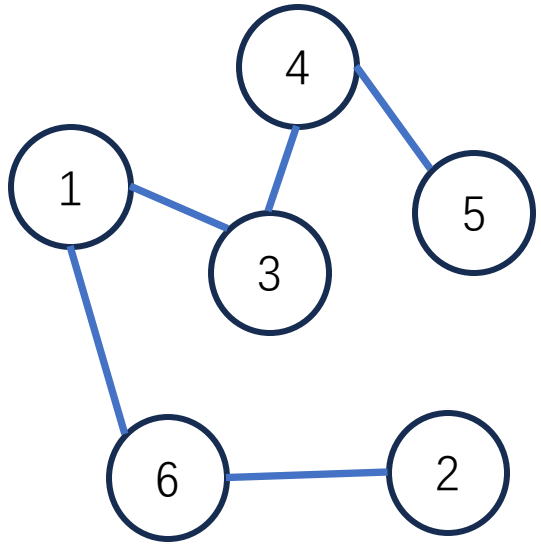
EDA 软件设计 I

Lecture 9

Quiz 1 讲解

- （9‘） BFS 遍历步骤可视化：Q 代表队列，V 代表“已访问”的节点，R代表输出结果（访问路径），起始节点为节点3，在下图的BFS遍历步骤中，写出每一步对应的 Q、V、R的状态变化（**注意：此处遵循邻接表内读取序号小的邻居先入队列）**：



Review:上节课重点

- DFS算法原理：
  1. 深入探索
  2. 回溯
- DFS 遍历的两种实现方式
  - 显示栈（核心：依靠 stack 的先进后出特性）
  - 递归（核心：依靠递归调用）
  - 共同点：维护已访问的节点（避免陷入无限循环）
- DFS算法复杂度
  - 时间：O(V + E)
  - 空间：O(V)

```
当栈 S 非空时:
    从栈 S 中pop出最后进入的节点作为当前节点
    对于当前节点的每个邻居:
        如果邻居不在 Visited 集合中:
            将邻居节点推入栈 S
            将邻居节点加入 Visited 集合
    将当前节点加入 Res 列表
```

```
DFS(node, visited):
    如果 node 在 visited 中:
        返回
    将 node 标记为已访问 (加入 visited 集合)
    对于 node 的每个邻居 neighbor:
        如果 neighbor 不在 visited 中:
            递归调用 DFS(neighbor, visited)
```

Review:上节课重点

- 递归:
  - 定义
  - 适合解决的问题
  - 基本结构

```
递归函数(parameter):
    如果满足基准情况:
        返回结果
    否则:
        进行递归调用
        返回递归调用的结果
```

Python

```
阶乘的递归计算:
    factorial(n):
        如果 n == 0:
            返回 1   (基准情况)
        否则:
            返回 n * factorial(n-1)   (递归调用)
```

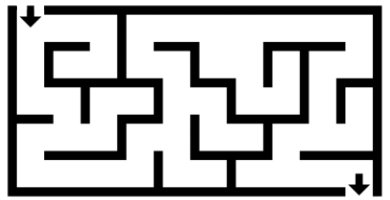
Python

```
斐波那契数列递归计算:
    fibonacci(n):
    如果 n == 0:
        返回 0   (基准情况1)
    如果 n == 1:
        返回 1   (基准情况2)
    否则:
        返回 fibonacci(n-1) + fibonacci(n-2)   (递归调用)
```

算法核心四要素 @ DFS

DFS应用

- **路径遍历**: 从某个起点出发，探索**一条**或者**所有可能**到达特定终点的路径
  - **在电路时序分析中**：用于确保信号在给定的时间内能够正确传递到输出端。
    - 通过DFS，可以遍历电路的所有路径，计算路径延迟，检查是否满足时序要求，进而检测出**关键路径**和**时序违规**。
- 连通性检测
  - 无向图中连通的定义
  - 通过从一个节点开始DFS遍历，如果能访问所有节点，则说明图是连通的；否则，图是不连通的，可以用来找出图中的**连通分量**
- 迷宫生成
  - Hint: 通过DFS遍历图的每个节点，并**随机选择**一个未访问的相邻节点递归访问，直到所有节点都被遍历完。这个过程可以生成一个随机的迷宫结构



DFS应用

- 适用于全排列问题：
  - 给定一组不重复的数字，生成它们的所有排列组合——leetcode medium难度
  - **Hint:** DFS在构建排列时，逐步选择每个元素并将其加入当前路径中，当路径构成一个完整排列时将其记录，否则在路径不满足条件时回溯
- 适用于组合问题：
  - 给定整数 n 和 k，从 1 到 n 中选出 k 个数字的所有组合——leetcode medium难度
  - **Hint:** DFS逐步选择数字，将其加入当前组合，递归生成后续数字。如果组合大小达到 k，则回溯。

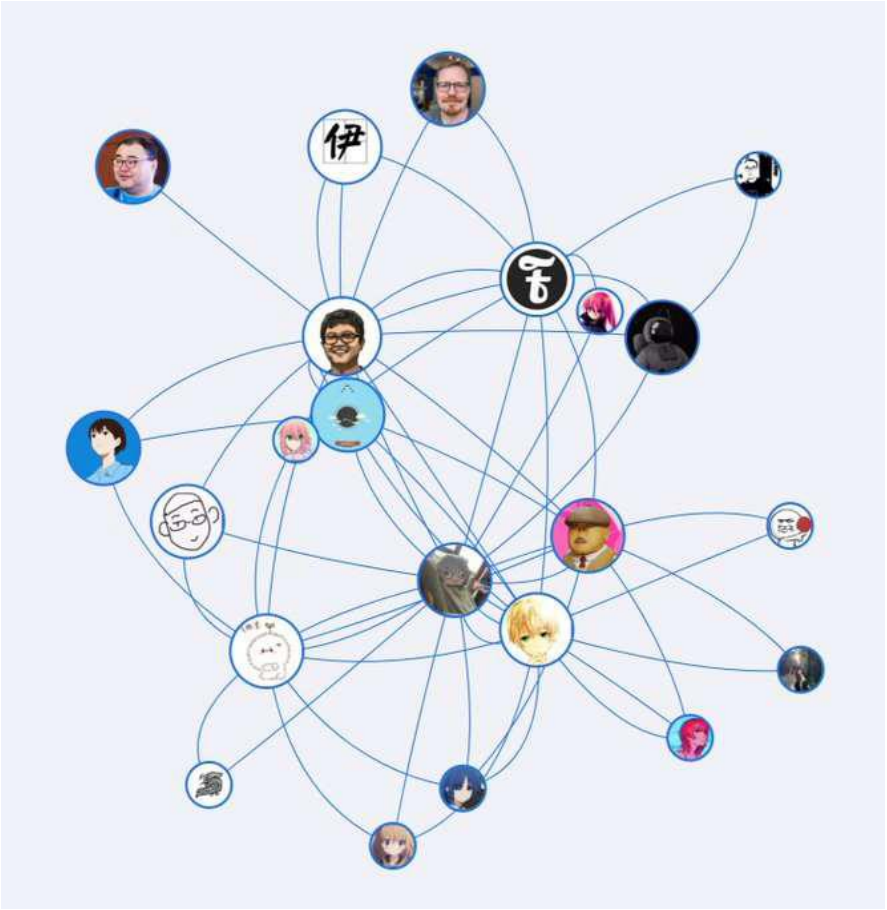
Progress so far

- BFS与DFS
  - 一个是"扫地僧"，先扫门前雪再向外扩张
  - 一个是"探险家"，一条路走到头，无路可走后再回溯
  - 目前为止共同点：作用在**无向图**上
- 图模型
  - 定义：一组由节点和边组成的**数学模型**，表示对象（节点）及其相互关系（边）
  - 分类：
    - **无向图**和**有向图**
    - **无权图**和**加权图**

无向图到有向图

社交网络图

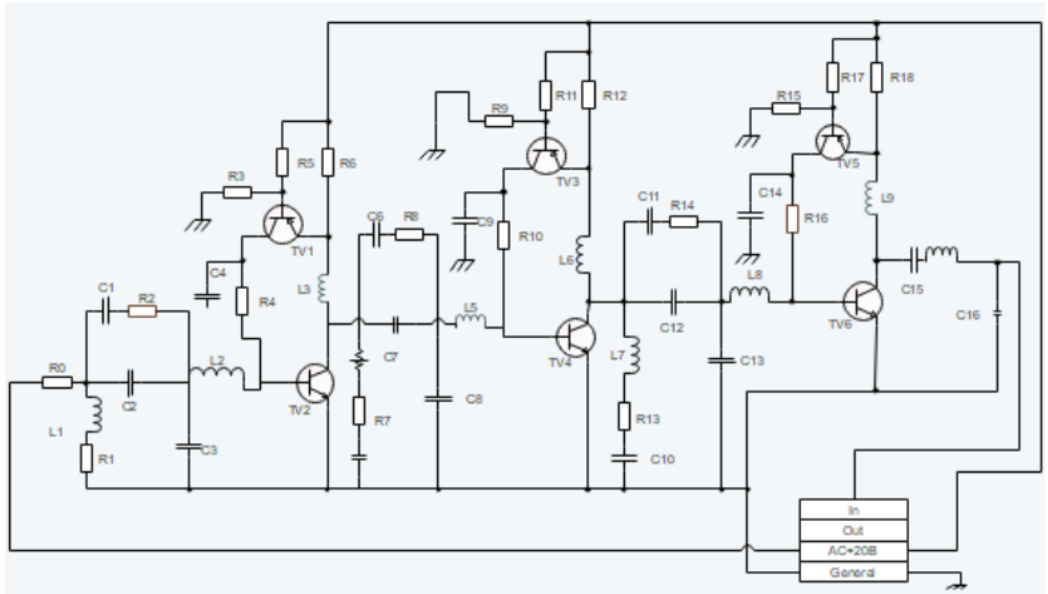
- 节点：用户
- **无向图的边**：友谊
  - 已经相互关注
  - 彼此是朋友
- **有向图的边**：关注关系
  - 包含单向关注



无向图到有向图

逻辑电路图

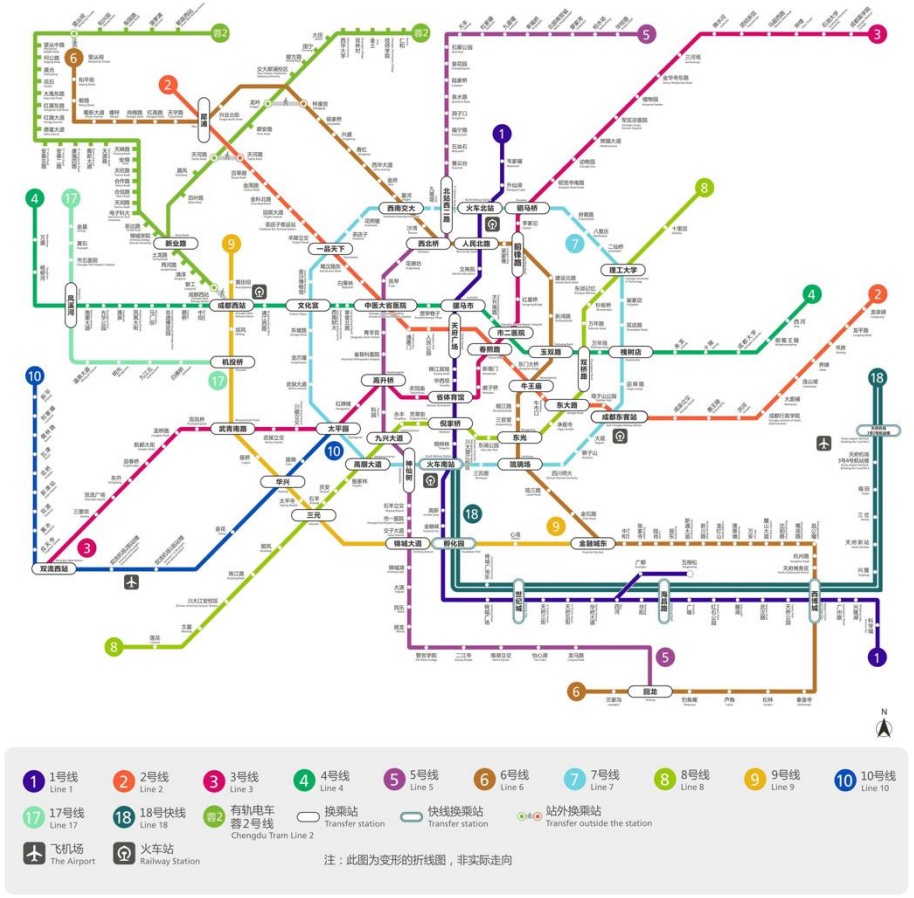
- 节点：不同的逻辑门（如 AND、OR、NOT、NAND、NOR 等）
- **无向图的边**：单纯代表逻辑门间的连接关系
- 逻辑门之间传递电信号实现特定功能（比如逻辑运算）
- **有向图的边**：信号的传输方向性



加权图（weighted graph）

轨道交通图

- 节点：车站或者交叉路口
- 边：地铁线路
- **边上的权重**：
  - ◆距离
  - ◆时间
  - ◆费用



无向无权图：表示相对基础的连接关系、拓扑结构

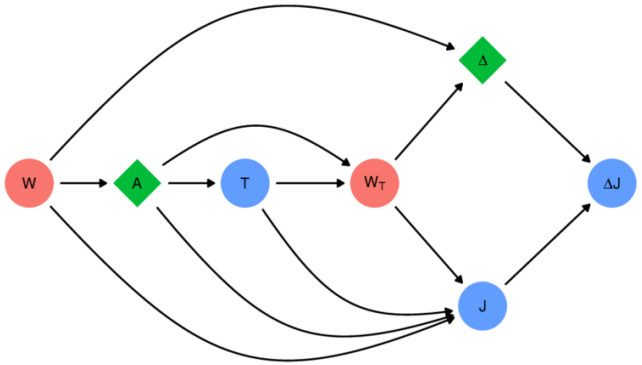
有向图、加权图：能描述对象（节点）之间的更加丰富的关系结构

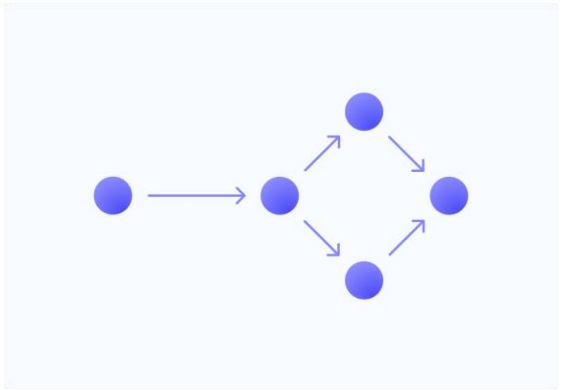
- 具有方向性的关系（有向图）
- 描述关系的强度、成本、容量等（加权图）

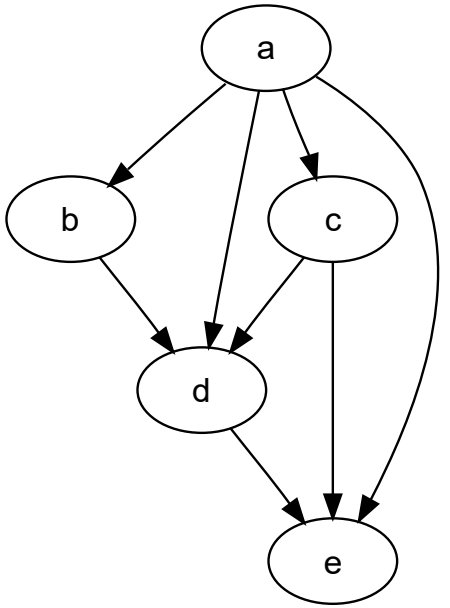
一种重要的有向图：**有向无环图**

有向无环图（DAG）

- DAG：directed acyclic graph
- 环的定义：**一条起点和终点相同的路径**
  - 在无向图中：从某节点出发，通过若干边，回到该节点
  - 在有向图中：从某节点出发，通过若干有向边，回到该节点
- 无环性：在图中**不存在任何**从某个顶点 u 出发，沿着有向边经过若干个顶点最终又回到 u 的路径



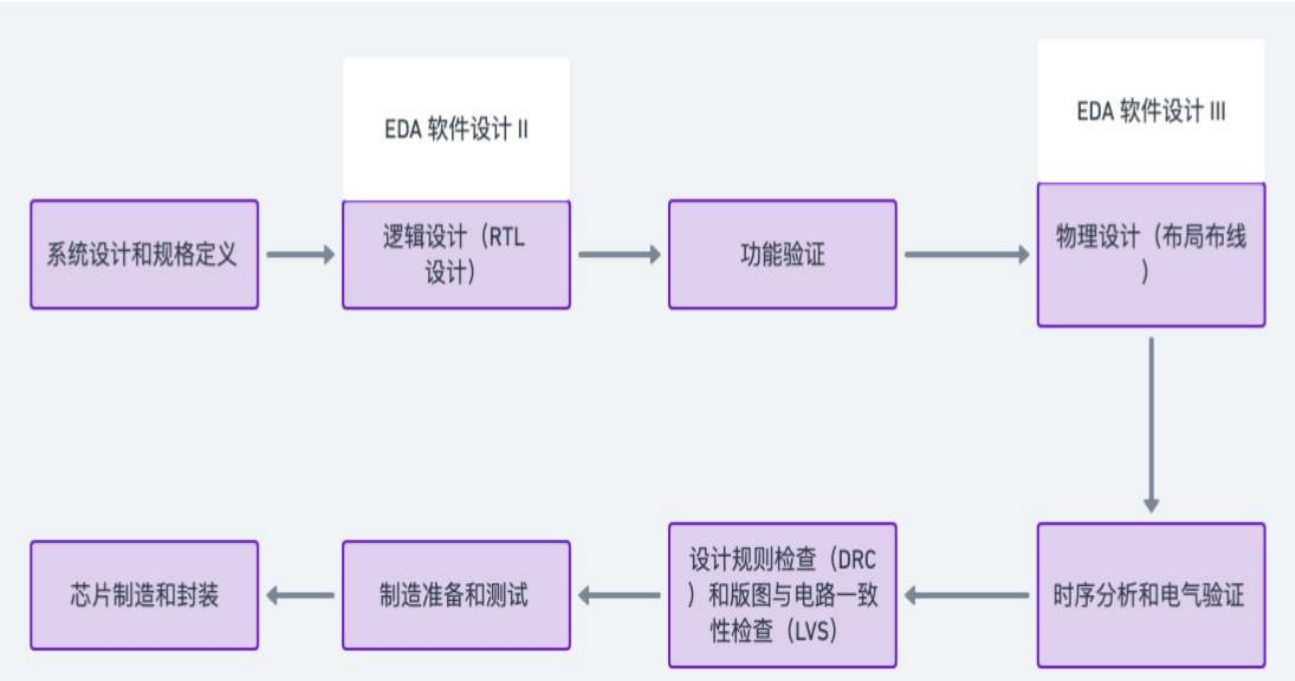




Why Study DAG?（DAG适用性）

大部分流程化的过程都可以被建模成有向无环图——芯片设计全流程

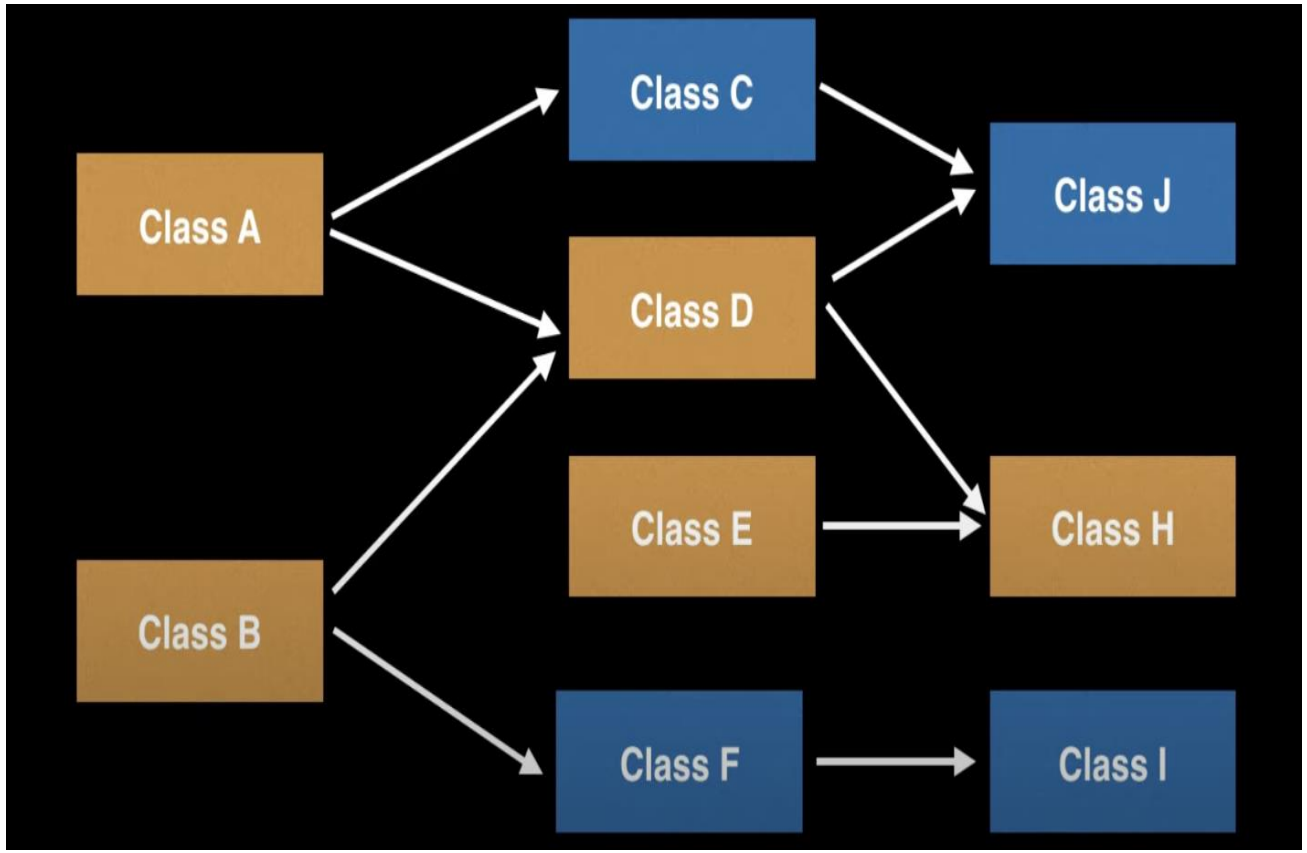
- 有向性：流程的每一步通常都有一个明确的方向，即每个步骤都会有前驱和后继任务。
  - 例如，逻辑设计必须在功能验证之前完成
- 无环性：大部分流程化过程是没有循环的，即如果步骤A是步骤B的前驱，那步骤A通常就不会是步骤B的后继



Why Study DAG?（DAG适用性）

大学选课系统

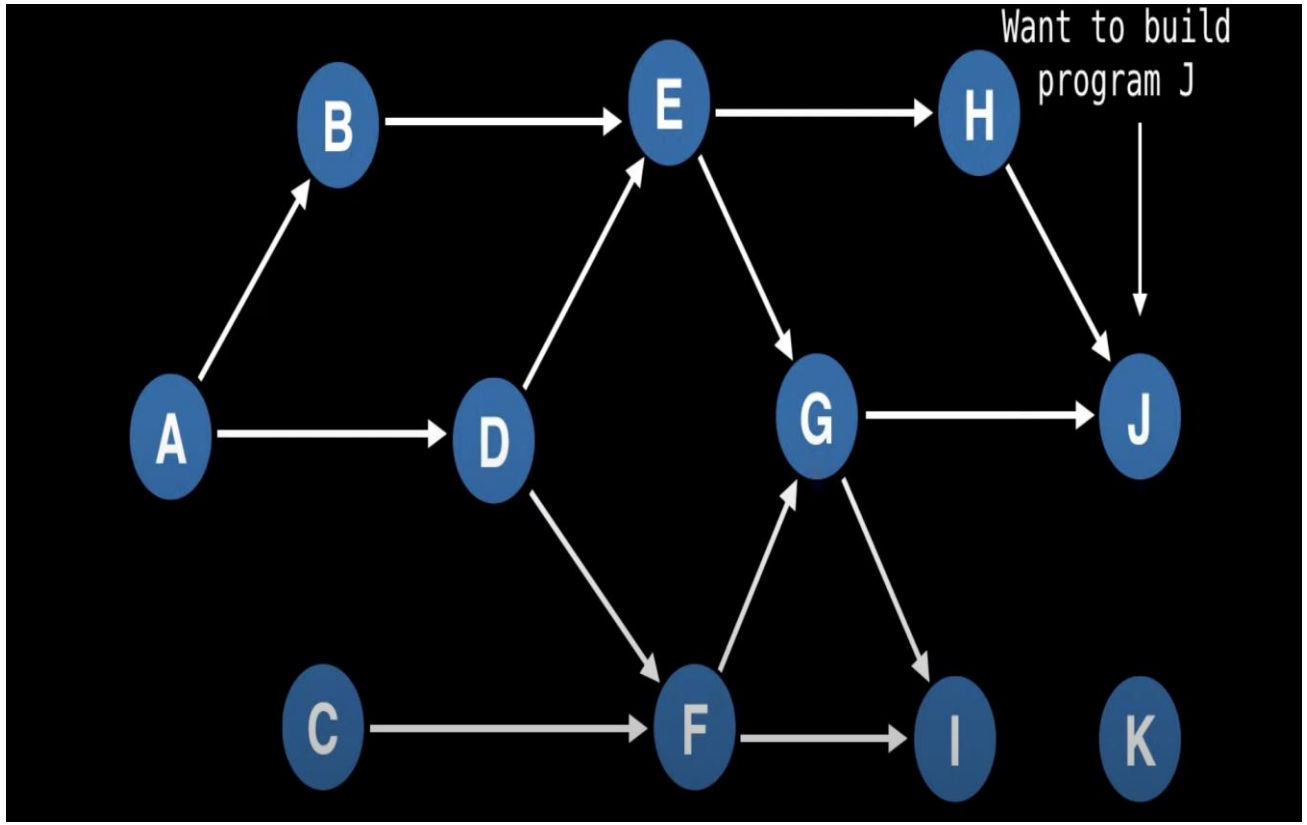
- Prerequisite: 前置必修课程
- 假设你是某大学的学生并且想选修 Class H，那你必须先修过 Class A、B、D、E作为 prerequisites
- 选课系统成一个有向无环图：
  - 有向边：**先修课程**的依赖关系
  - 无环性：不存在循环依赖，否则系统陷入矛盾状态，形成**“死锁(Deadlock)”**



Why Study DAG?（DAG实用性）

程序构建依赖关系

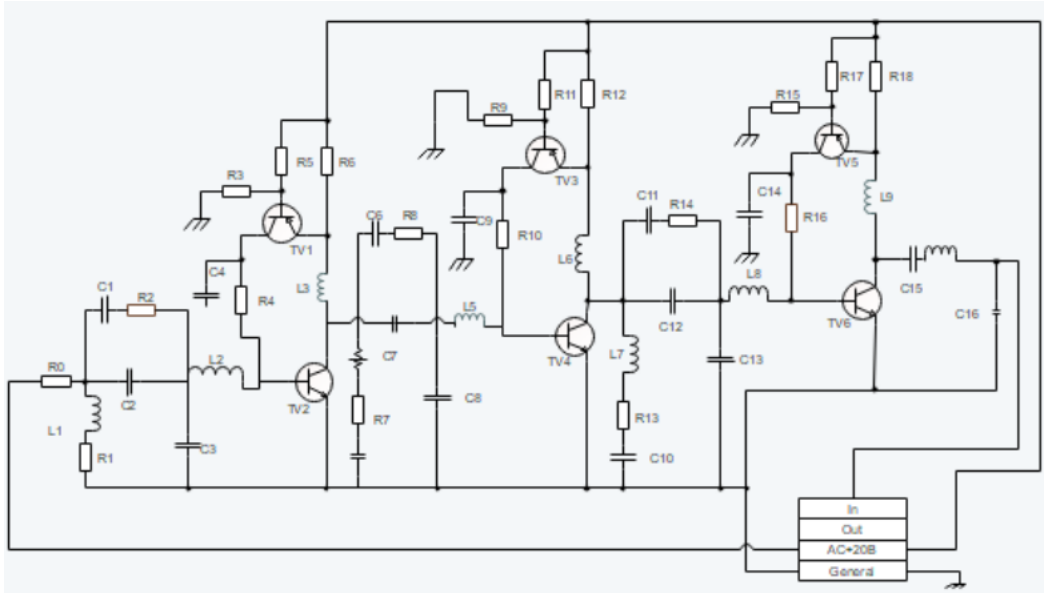
- 节点： 程序或模块
- 有向边：模块之间的依赖关系
- 无环性：模块之间不存在相互依赖的关系



Why Study DAG?（DAG实用性）

逻辑电路图

- 节点：不同的逻辑门（如 AND、OR、NOT、NAND、NOR 等）
- 有向边：每条边表示信号传递的方向性
- 无环性：信号的传递是单向的



领域	DAG 应用	In detail
数据处理与工作流管理	大规模数据处理	在分布式计算框架中（如 Hadoop、Spark 等），DAG 常用来表示数据处理的依赖关系。
	工作流管理	在自动化工作流管理中，DAG 用于描述任务之间的依赖关系和执行顺序。
版本控制	Git版本控制系统	Git 的分支管理和合并过程可以用 DAG 来表示。
电路设计与分析	时序分析	DAG 用于分析电路中的信号路径，确保信号传输在规定的时钟周期内完成。
数据库和查询优化	查询计划优化	在数据库中，DAG 可以用于表示 SQL 查询的执行计划。
机器学习和深度学习	神经网络架构	DAG 可用于描述神经网络的前向传播过程。
区块链和加密货币	区块链	某些区块链系统（如 IOTA）使用 DAG 来替代传统的线性区块链结构。

有向图与无向图存储的区别

无向图的邻接矩阵：

- 是一个对称矩阵 M，M[u][v] = 1 和 M[v][u] = 1，表示u和v之间有无向边

无向图的邻接表：

- 对于每个节点 u，它的邻接列表包含所有与它相连的节点 v

示例无向图：

邻接矩阵：

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

示例邻接表：

A: B, C
B: A, C
C: A, B

有向图与无向图存储的区别

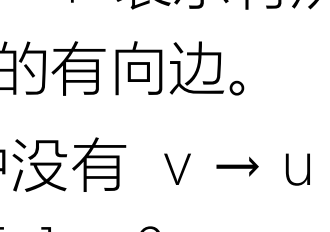
有向图的邻接矩阵：

- M[u][v] = 1 表示有从节点 u 到节点 v 的有向边。
- 如果图中没有 v → u 的反向边，则 M[v][u] = 0 。

有向图的邻接表：

- 只存储从当前节点**出边**。对于每个节点 u，邻接表只包含所有被 u 指向的节点 v

示例有向图：



邻接矩阵：

	A	B	C
A	0	1	1
B	0	0	0
C	0	0	0

示例邻接表：

A: B, C
B:
C:

Natural question：如何判定一个有向图为为DAG?