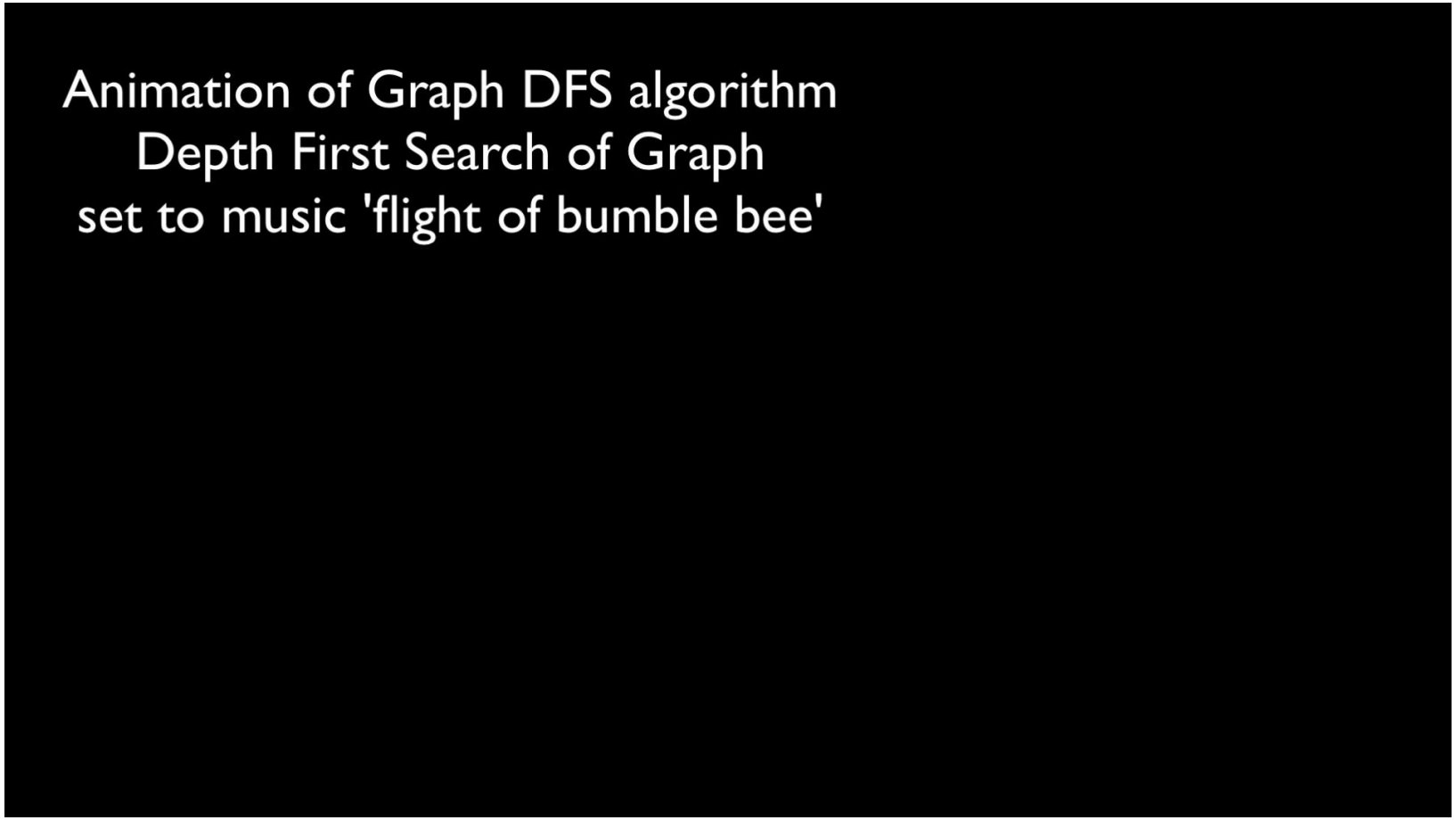
EDA 软件设计 I

Lecture 8

DFS遍历在较大图上的展示



DFS遍历的另一种实现方式

- DFS 遍历实现（经典实现）要点：
  - 可以用**栈(Stack: FILO)**来实现
  - 维护一个**“已访问”列表**
  - 也可以用**递归**实现

递归（Recursion）

- 递归的定义：
  - ✓ 一个函数在其定义中调用自身的编程技巧
  - ✓ 一种在定义或求解问题时通过自身调用自身的技术
  - ✓ “自己调用自己”
- 背后思想：
  - “大事化小”
  - 将大问题转化为类似的规模较小的问题
- 策略Advantage:
  - 只需**少量的程序**就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量

使用递归的情况（“心法”）

递归特别适合处理**可以通过相同类型的子问题来解决**的问题

为了描述问题的某一状态，必须用到该状态的上一个状态；而如果要描述上一个状态，又必须用到上一个状态的上一个状态...这样用自己来定义自己的方法就是递归

递归“天然适合”解决的问题

- **分治法问题**：通过递归将大问题分解为更小的子问题，解决后再合并结果，例如：
  1. 归并排序：递归地将数组分成两半，对每一半进行排序，最后合并两个有序数组
  2. 快速排序：选择一个基准元素，递归地对基准元素两侧的子数组进行排序
- **处理树结构**：树的每个节点都可以看作一棵子树，递归遍历或处理子树是非常自然的方式，例如：
  1. 树的前、中、后序遍历

```python
def inorder_traversal(node):
    if node is None:
        return
    inorder_traversal(node.left)
    print(node.value)
    inorder_traversal(node.right)
```

递归“天然适合”解决的问题

- **递归可以轻松地处理一些嵌套结构**，例如
  1. 解析嵌套的括号
  2. JSON 结构
  3. XML 文档
- **一些具有递归性质的数学问题**，例如
  1. 阶乘： n! = n * (n-1)!
  2. 斐波那契数列： F(n) = F(n-1) + F(n-2)

递归的基本结构

① **基准情况（终止条件）：**

- 每个递归过程必须有一个或多个基准情况，当满足这些条件时递归调用停止（若没有基本情况陷入无限循环）
- 基准情况是处理最简单的情况，不需要再进行递归

② **递归情况：**

- 函数在每次调用时如何继续执行的部分
- 通常将问题分解为规模更小的子问题，最终通过多次递归调用和返回结果来解决问题

```
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
```

递归经典例子

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)
```

DFS递归特性

- DFS 机制：
  1. 深入探索：从起始节点出发，沿着一条路径尽可能深入，直到无法继续再回溯
  2. 回溯：当无法继续深入时，算法会回溯到上一个节点并探索其他路径
- DFS递归实现的**核心原理**：
  ① 访问当前节点并标记已访问
  ② 递归访问邻居节点
  ③ 直到没有未访问的邻居节点，回溯并探索其他路径
    ◆回溯机制：当所有邻居节点都已访问时，函数返回上一级节点，继续遍历其他路径

DFS的递归实现

- 递归实现的步骤：

  1. **起点**开始为当前节点，标记它为已访问
  2. **递归深入**：对当前节点的每一个未访问的邻居节点进行递归调用DFS
  3. **回溯**：如果当前节点的所有邻居节点都已经访问过或没有更多邻居节点可访问，则回溯到上一个节点继续搜索
  4. **终止条件**：当所有节点都被访问过时，或者到达搜索目标时，递归结束

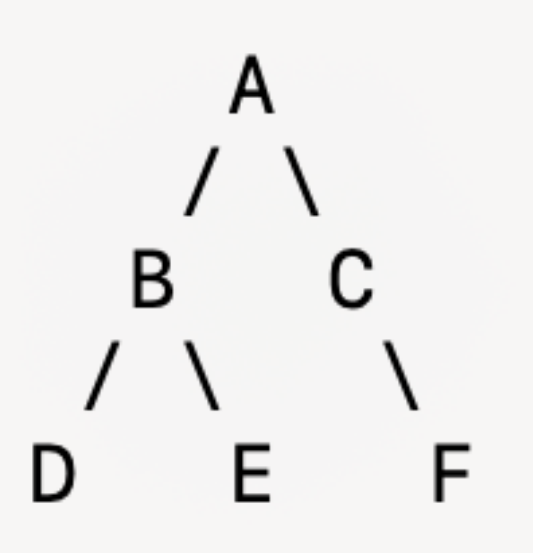
- DFS 【递归实现】伪代码：

```
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)
```

- 细节：**不同于通过stack实现，递归实现时visited作为参数**

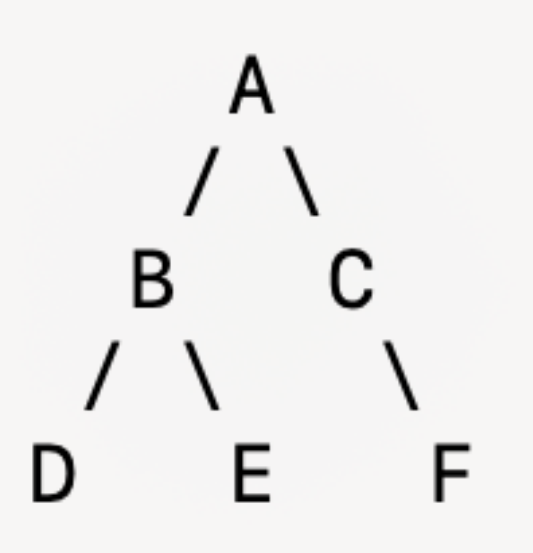
DFS遍历递归过程

递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A



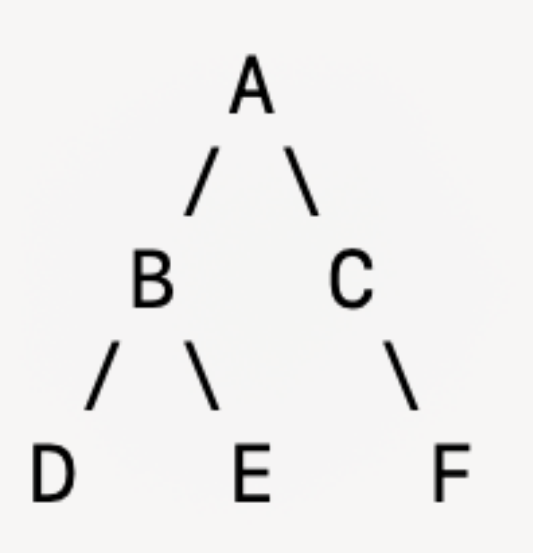
DFS遍历递归过程

递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B

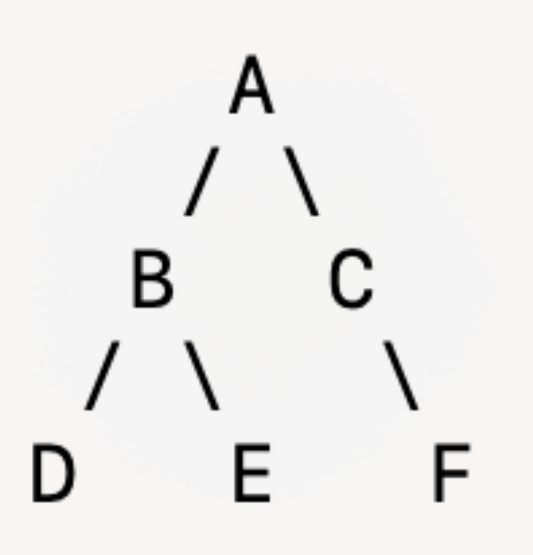


DFS遍历递归过程

递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D

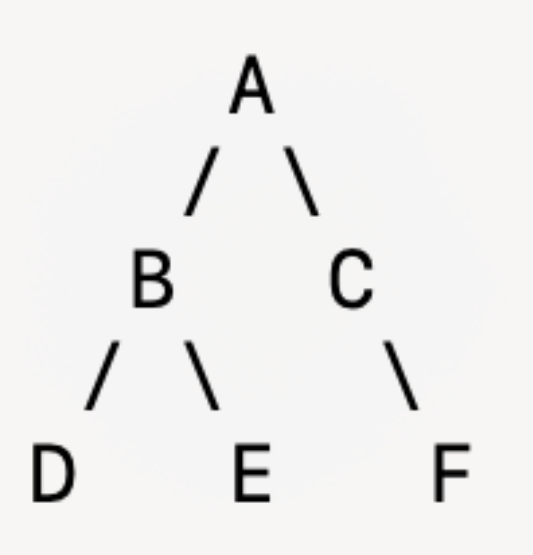


DFS遍历递归过程



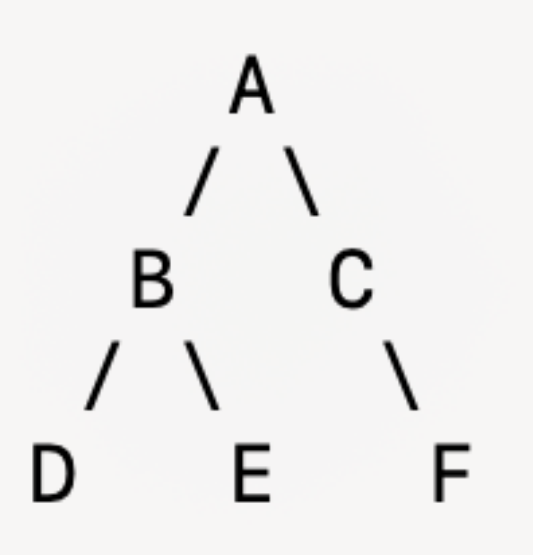
递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D

DFS遍历递归过程



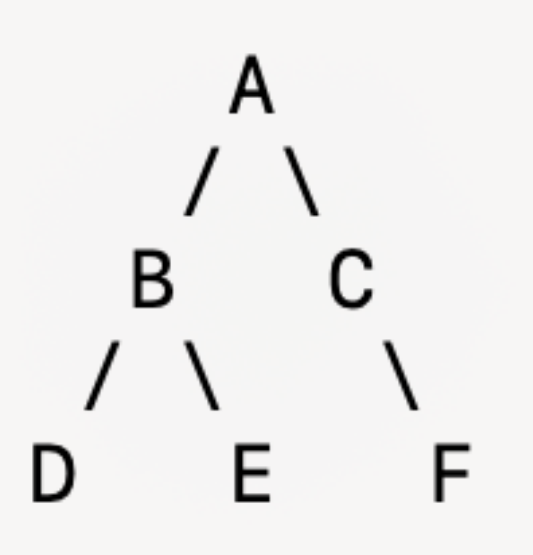
递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D
5. **访问B的邻居节点E**：进入E节点，将E标记为访问过	A → B → D → E

DFS遍历递归过程



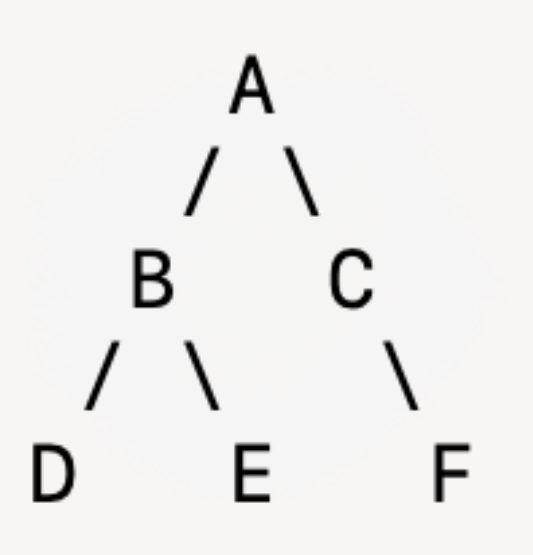
递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D
5. **访问B的邻居节点E**：进入E节点，将E标记为访问过	A → B → D → E
6. **E没有未访问的邻居**：回溯到节点B，并进一步回溯到A	A → B → D → E

DFS遍历递归过程



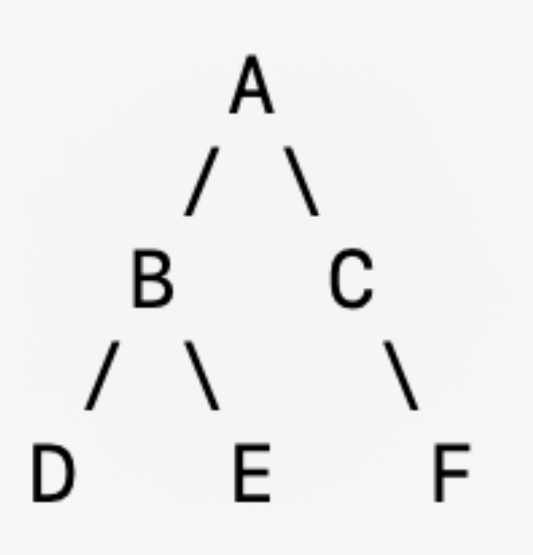
递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D
5. **访问B的邻居节点E**：进入E节点，将E标记为访问过	A → B → D → E
6. **E没有未访问的邻居**：回溯到节点B，并进一步回溯到A	A → B → D → E
7. **访问A的邻居节点C**：进入C节点，将C标记为访问过	A → B → D → E → C

DFS遍历递归过程



递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D
5. **访问B的邻居节点E**：进入E节点，将E标记为访问过	A → B → D → E
6. **E没有未访问的邻居**：回溯到节点B，并进一步回溯到A	A → B → D → E
7. **访问A的邻居节点C**：进入C节点，将C标记为访问过	A → B → D → E → C
8. **访问C的邻居节点F**：进入F节点，将F标记为访问过	A → B → D → E → C → F

DFS遍历递归过程



递归过程	「最终输出路径」当前状态
1. **开始于A**：我们从节点A开始，将其标记为访问过	A
2. **访问A的邻居节点B**：进入B节点，并将B标记为访问过	A → B
3. **访问B的邻居节点D**：进入D节点，将D标记为访问过	A → B → D
4. **D没有未访问的邻居**：回溯到节点B	A → B → D
5. **访问B的邻居节点E**：进入E节点，将E标记为访问过	A → B → D → E
6. **E没有未访问的邻居**：回溯到节点B，并进一步回溯到A	A → B → D → E
7. **访问A的邻居节点C**：进入C节点，将C标记为访问过	A → B → D → E → C
8. **访问C的邻居节点F**：进入F节点，将F标记为访问过	A → B → D → E → C → F
9. **F没有未访问的邻居**：回溯到节点C，并进一步回溯到A，此时整个图遍历结束	A → B → D → E → C → F

DFS的两种实现方式

1. 栈（显示栈）
2. 递归调用（调用栈）

- Question: 是否是能使用显示栈（stack）来实现的算法都可以用递归来实现?

算法核心四要素 @ DFS

算法分析 @ DFS遍历

- 对于一个有 V 个顶点和 E 条边的图
- 时间复杂度：
  - DFS会访问每个顶点一次，**并且会检查每条边一次**
  - 因此，时间复杂度为：O(V + E)
- 空间复杂度：
  - **递归深度**：在最坏情况下（如线性结构），递归深度可能达到图中顶点的数量 V，因此栈的空间复杂度为 O(V)
  - **显式栈**：非递归版本中，栈的最大深度也是 O(V)，因为在最坏的情况下，栈中可能存储所有的节点

Quiz 1 Time

- 快速找到属于你的 quiz座位
- **把手机放在旁边的桌子上**
- 到时间停笔直接离开，试卷放到桌子（提前交的交的讲台上）
- **如有作弊情况：所有quiz成绩为 0**