EDA 软件设计 I

Lecture 5

Review ends and Recall

Which chapter are we at?

EDA 常用图**算法**

All you need to know about an **Algorithm（算法）**

当你要学一个算法时，你需要学什么？

不仅限于这门课，以后了解任何一个算法，从哪些方面去了解它？

一个算法的全貌

算法核心四要素

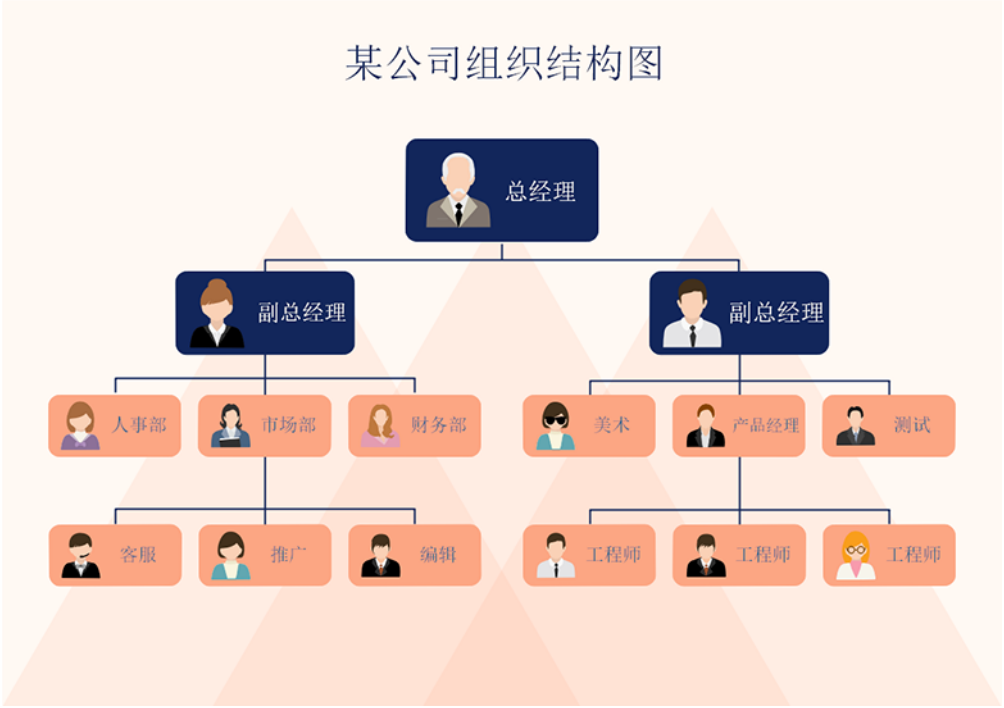
算法原理（概念和策略）

- 算法原理：解决问题的**核心理论和机制**，强调解决问题的思维方式和逻辑框架
  - ① 自然语言描述
  - ② 图示、可视化描述
  - ③ 简化版伪代码描述
- 常见的算法设计策略：
  1. 分治法（divide and conquer）：把复杂问题分解成更小、更易处理的子问题，逐个解决后再合并
  2. 贪心（greedy）：每次做出当下情况的最优选择，而不关心未来的全局最优
  3. 动态规划（dynamic programming）："记忆化"和"最优子结构"
  4. 回溯（backtracking）：逐步构建解决方案并在发现不符合条件时回退
- 锻炼抽象思维：从具体问题中抽象出通用的算法模型

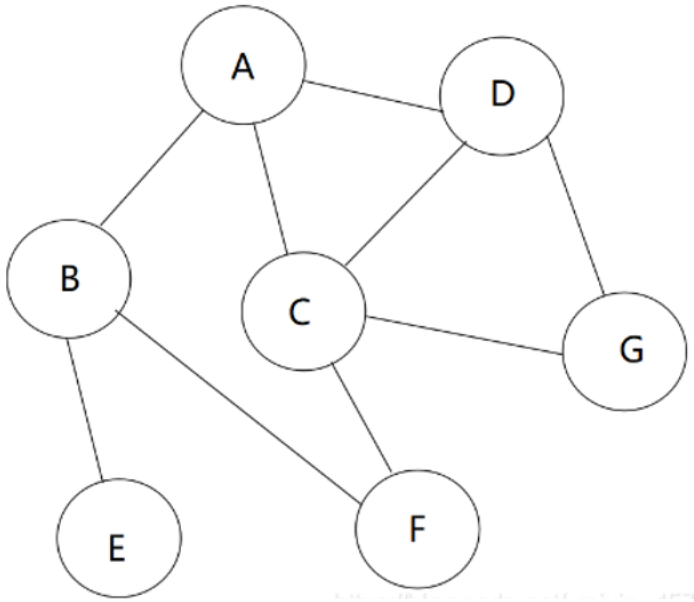
算法原理 @ BFS遍历

- **自然语言描述**——层次遍历：
  1. 从起始点开始逐层访问「图或树中」节点
  2. 首先访问起始节点的所有直接「邻居」
  3. 然后访问这些「邻居的邻居」
  4. 依次类推

- **类比描述**——“公司任务传递”



算法原理 @ BFS



BFS原理在**small graph**上的展示，

假设起始点是**G**：

G → [C D]

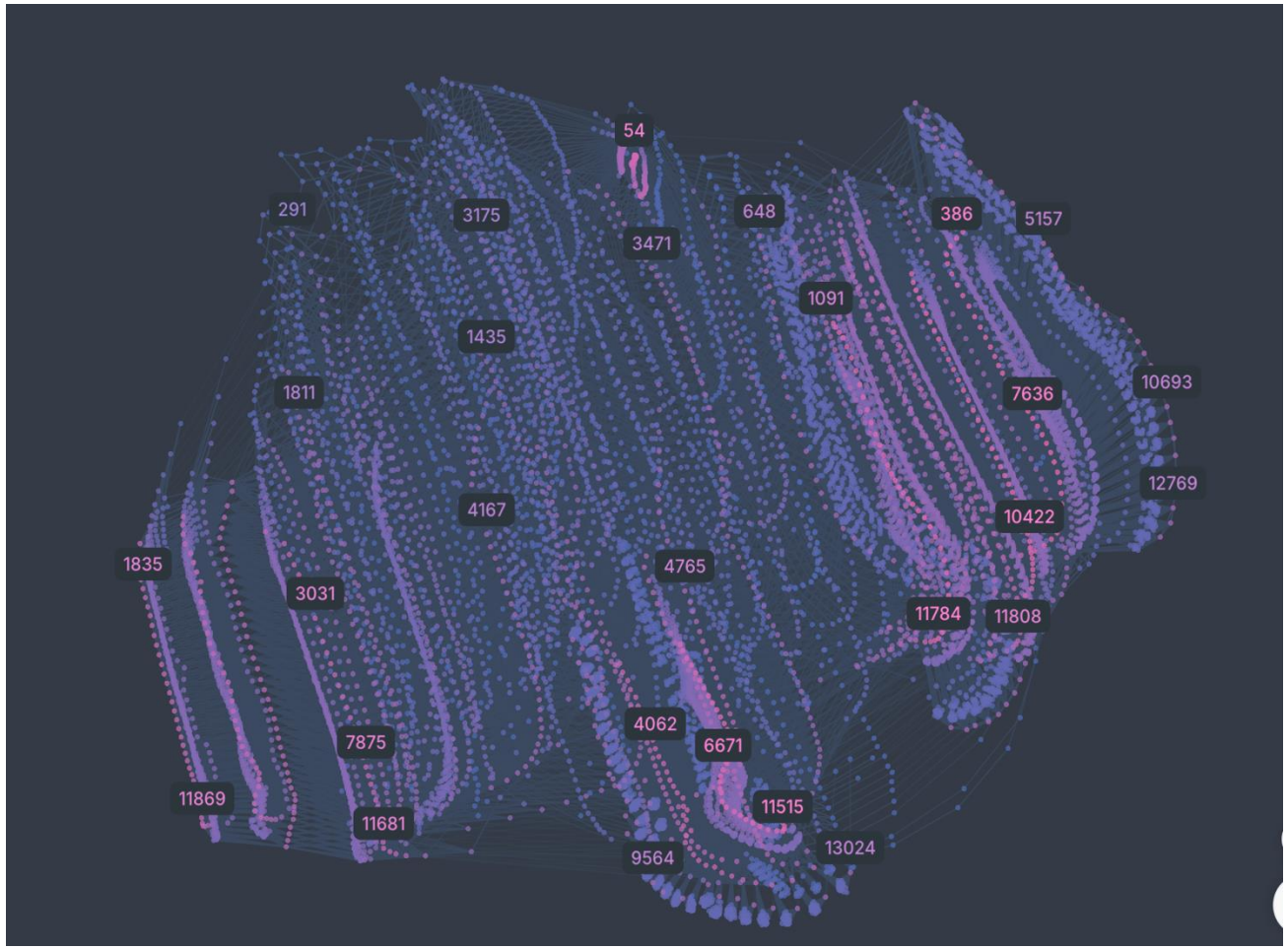
G → [C D] → [A F]

G → [C D] → [A F] → B

G → [C D] → [A F] → B → E

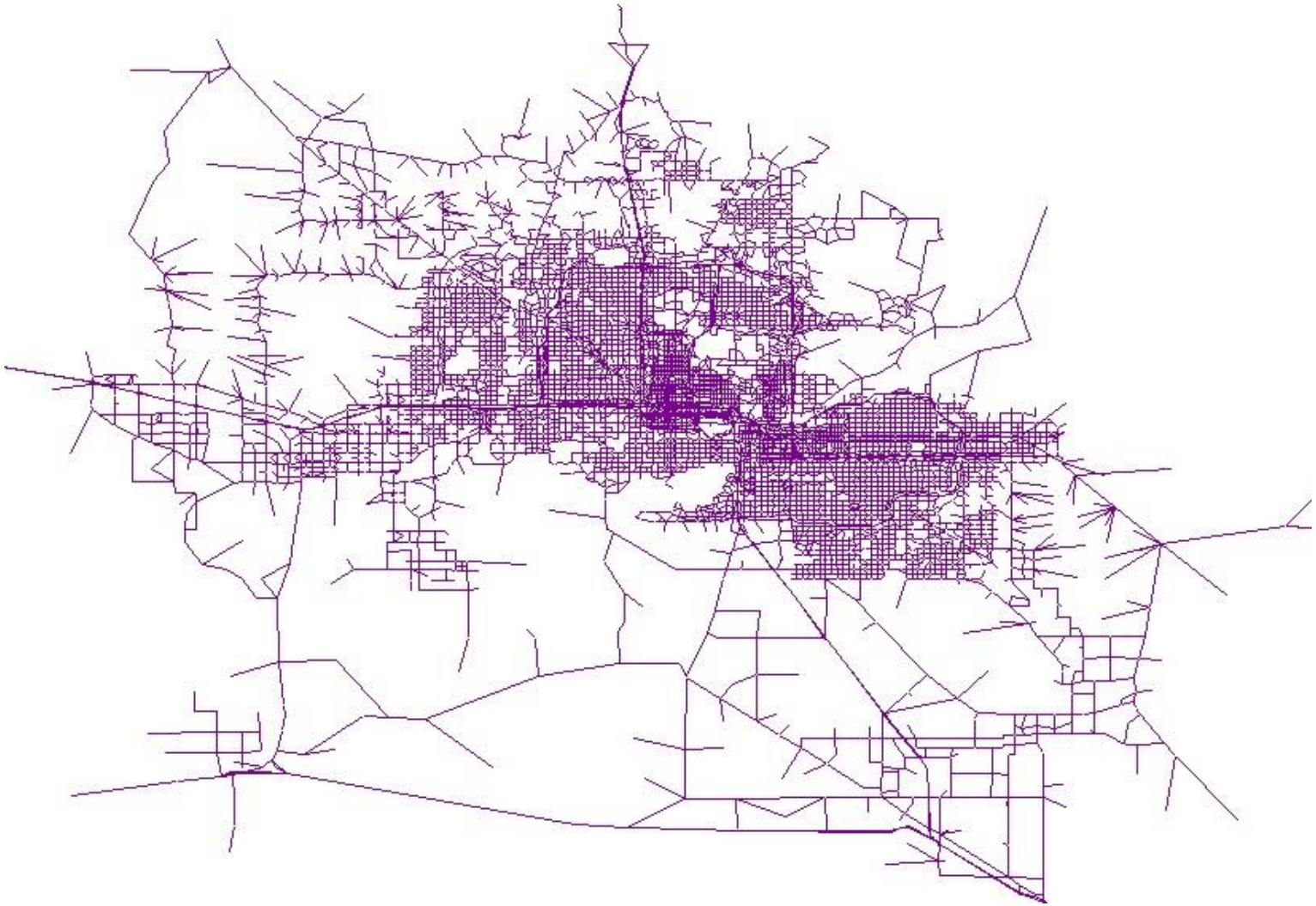
算法原理到算法实现 Why?

- 在**small graph**上，不用靠计算机解决问题
- But in **large graph**, **unable to solve problem by hand and eye!**

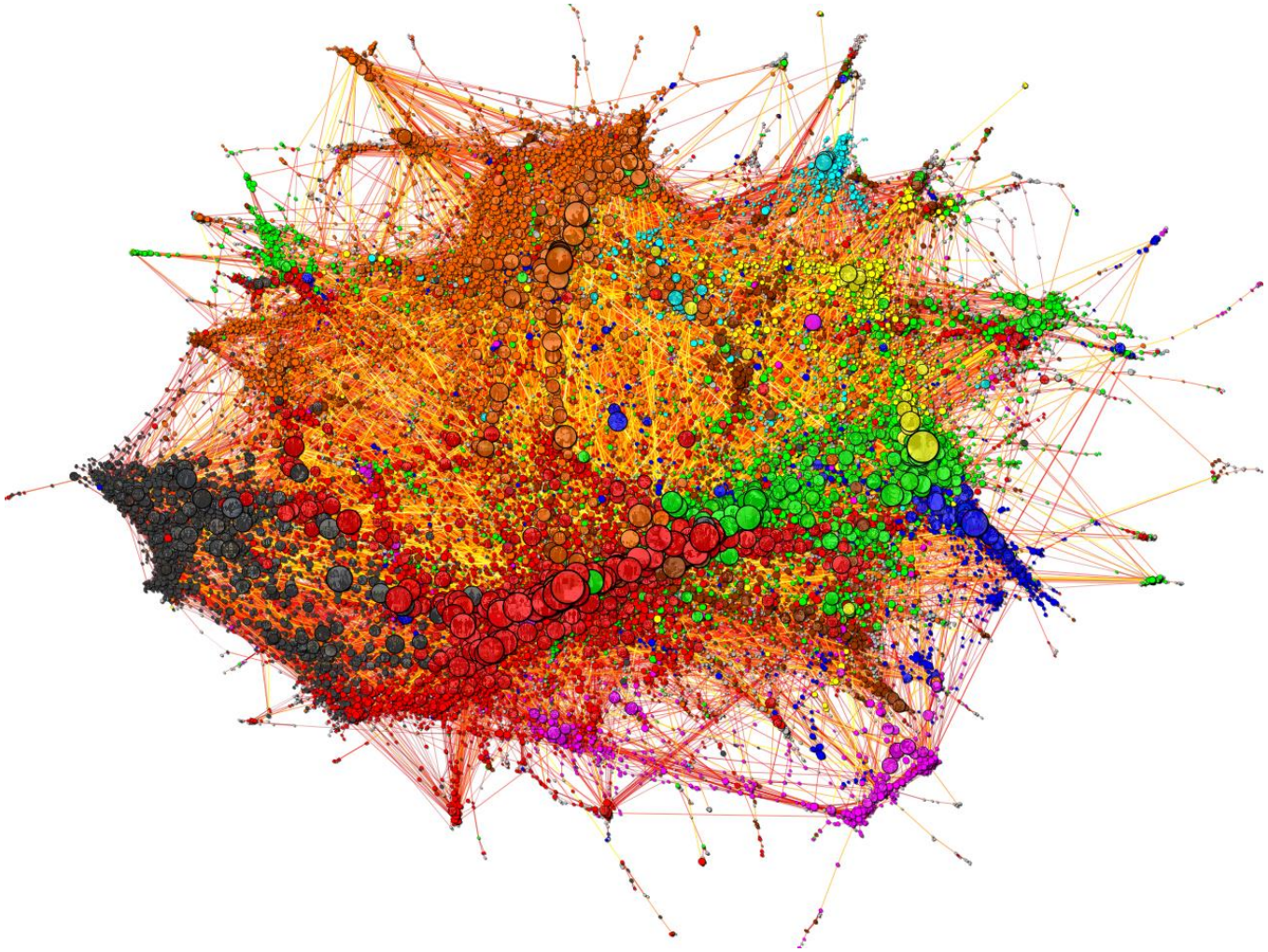


Real-world large graphs

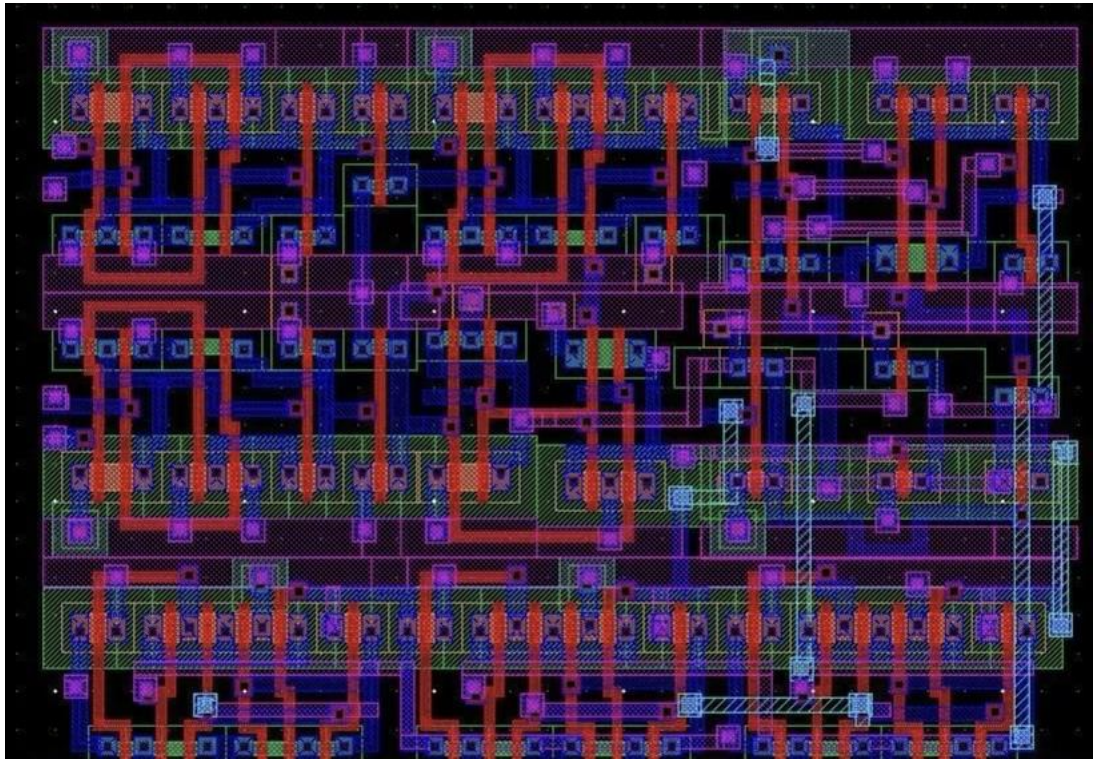
Real-world large graphs

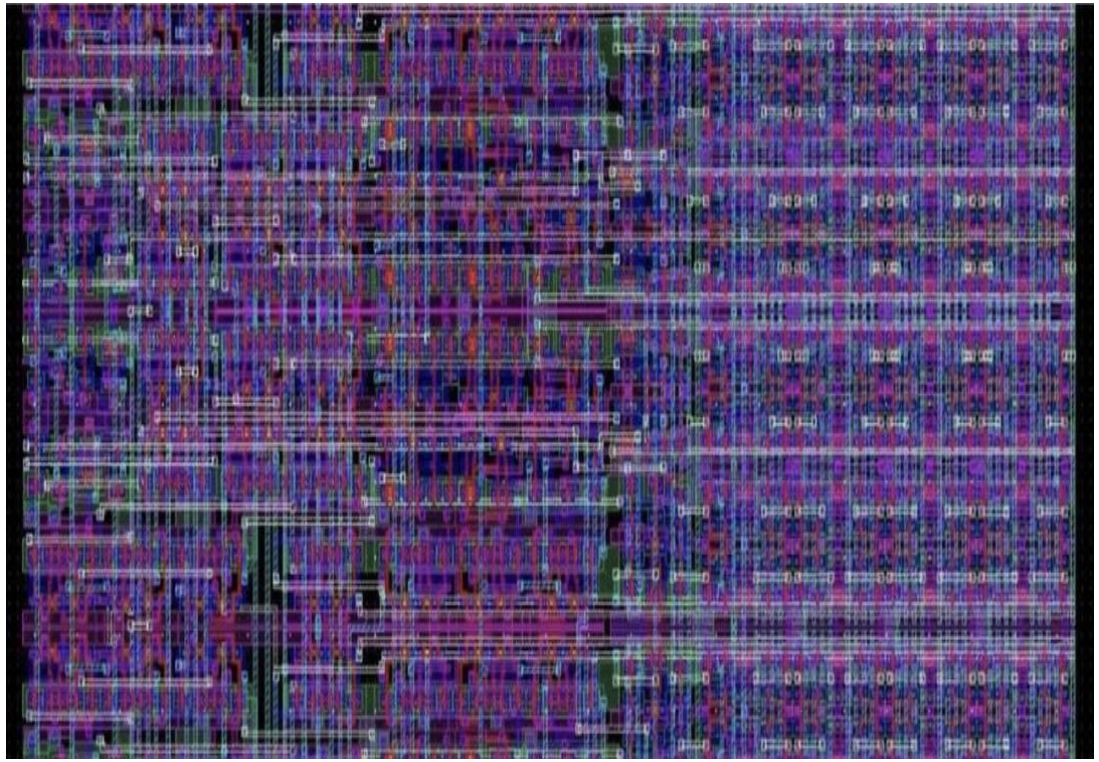


Real-world large graphs



Large “graphs” in EDA





算法核心四要素

算法实现

◆ 实现（implementation）：**通过编程，让计算机按照预定步骤解决问题**

1. 编程语言：掌握语言特性，选择合适语言
   - ① C：高性能、精细内存控制，适合底层开发，但需要手动管理内存，易出现内存错误
   - ② Java：跨平台、面向对象，适合大型应用，自动管理内存，但比 C慢，代码冗长
   - ③ Python：易学易用、开发效率高，适合快速开发和数据科学，但性能不如编译型语言
2. 数据结构：选择和使用适当的数据结构来支持算法的高效执行

   “数据结构是算法的骨架”
3. 代码优化：高效、可读、可维护

算法实现 @ BFS遍历

- **Review：**
  1. Breadth-First Search是一种机制/方式，标准情况下，带有 input 和 output 后，才叫一个算法
  2. For example:
     - input: Graph + 起始节点
     - output: 用BFS的方式获得的图节点访问顺序
     - 算法名称：**BFS traversal （广度优先遍历）**

算法实现 @ BFS遍历

- BFS Traversal 实现（经典实现）关键点：

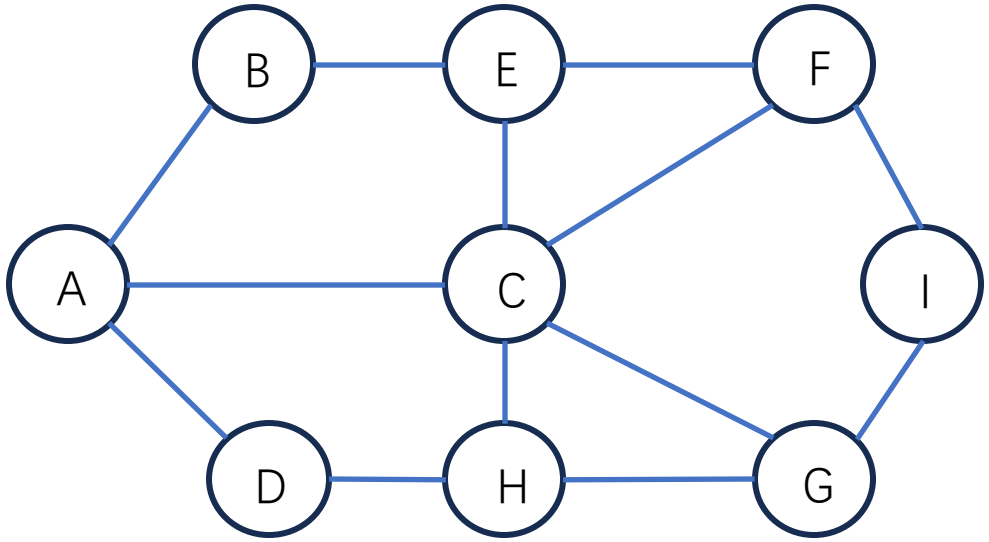
① BFS利用**队列**来管理待访问的节点：

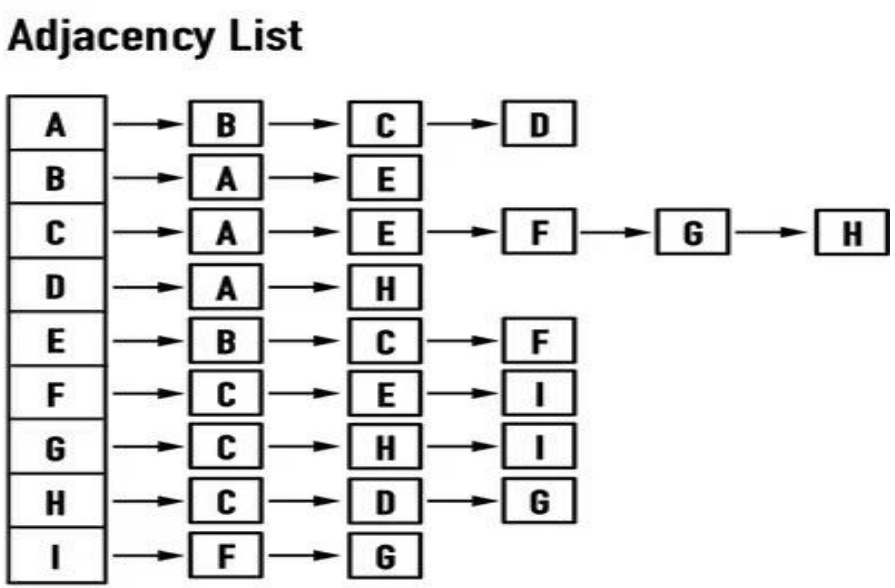
先进先出的特性确保**先发现的节点先被访问**，从而实现层次遍历。

② 避免重复访问：

通过维护一个**“已访问” 列表**，确保每个节点只被访问一次，防止无限循环或重复计算。

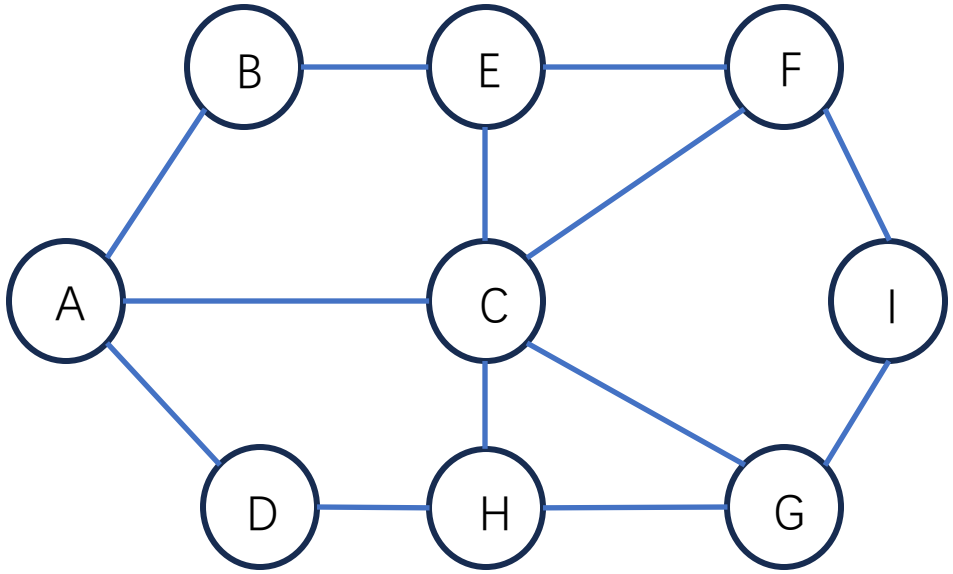
BFS遍历算法实现可视化





Input: 邻接表+起始节点A

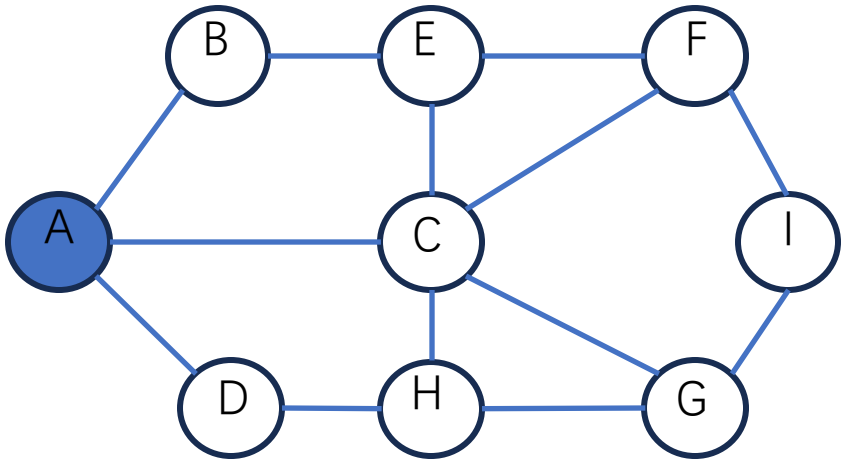
BFS遍历算法实现可视化



- **Q（仅是名称）：**
  - ① 实质数据结构：**队列**
  - ② 作用：维护将要访问的节点
- **Visited（仅是名称）：**
  - ① 实质数据结构：**待定（数组？链表？集合？……？）**
  - ② 作用：维护已经走过的节点
- 第一步：将起始节点 A 放进 **Q** 和**Visited**，创建**Res代表output，即遍历顺序**

BFS遍历算法实现可视化

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【A】，**Visited** = 【A】

Res = 【】

当前节点：A

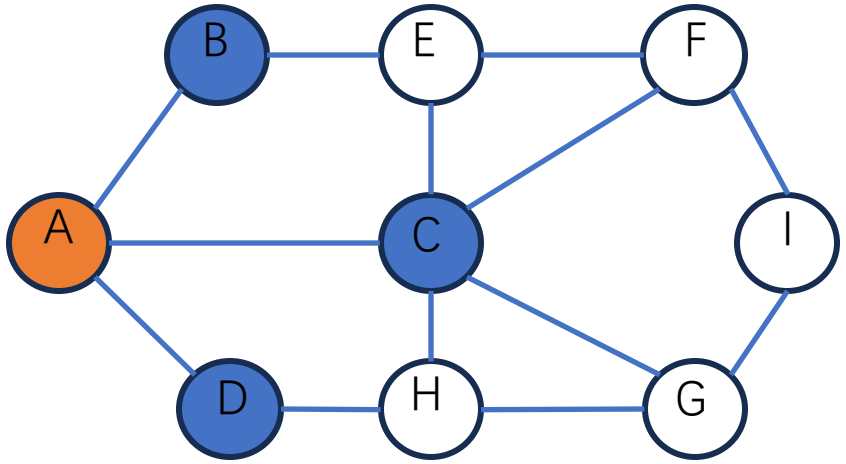
Q = 【A】→ 【】 → 【D, C, B】

Visited = 【A】 → 【A,D,C,B】

Res = 【】→ 【A】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中: push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【D, C, B】, **Visited** = 【A,D,C,B】

Res = 【A】

当前节点：D

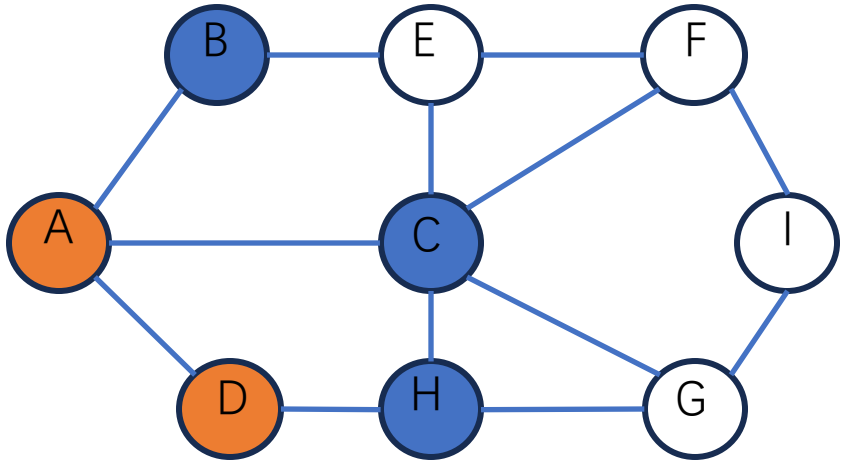
Q = 【D, C, B】→ 【C, B】→ 【C, B, H】

Visited = 【A,D,C,B】→ 【A, D,C,B,H】

Res = 【A】→ 【A,D】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【C,B,H】, **Visited** = 【A,D,C,B,H】

Res = 【A,D】

当前节点：C

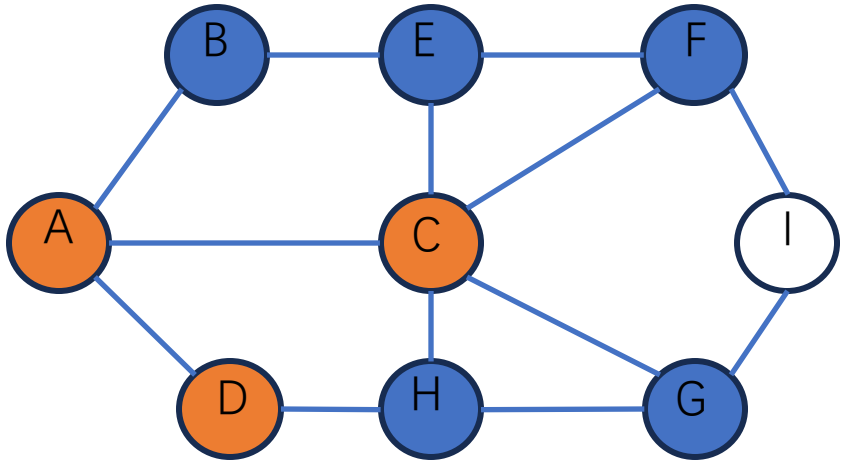
Q = 【C, B, H】 → 【B, H】 → 【B,H,G,F,E】

Visited = 【A,D,C,B,H】 → 【A,D,C,B,H,G,F,E】

Res = 【A,D】 → 【A,D,C】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【B,H,G,F,E】

Visited = 【A,D,C,B,H,G,F,E】

Res = 【A,D,C】

当前节点：B

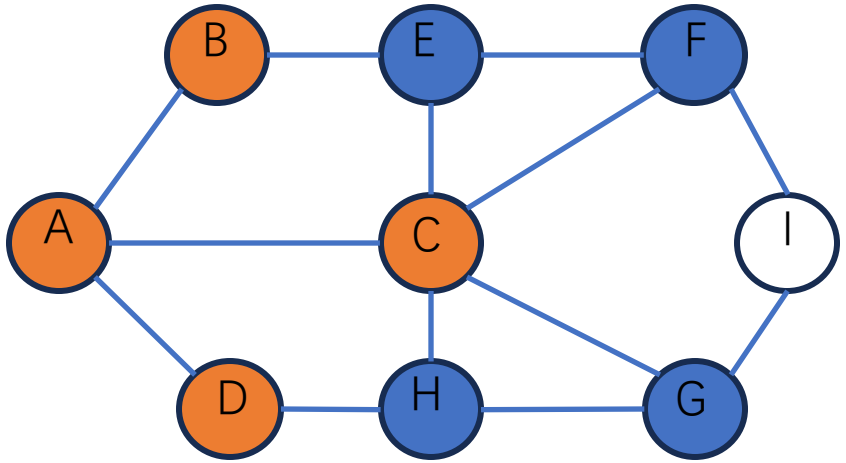
Q = 【B,H,G,F,E】→【H,G,F,E】

Visited = 【A,D,C,B,H,G,F,E】

Res = 【A,D,C】 →【A,D,C,B】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【H,G,F,E】

Visited = 【A,D,C,B,H,G,F,E】

Res = 【A,D,C,B】

当前节点：H

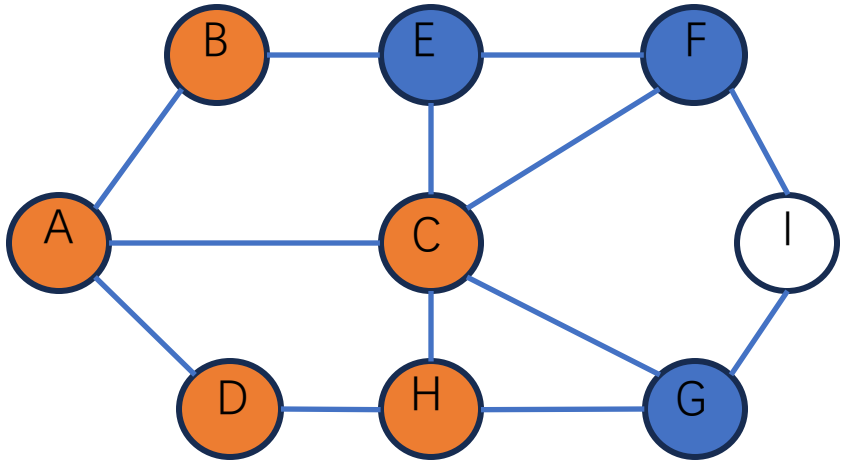
Q = 【H,G,F,E】→【G,F,E】

Visited = 【A,D,C,B,H,G,F,E】

Res = 【A,D,C,B】→【A,D,C,B,H】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【G,F,E】

Visited = 【A,D,C,B,H,G,F,E】

Res = 【A,D,C,B,H】

当前节点：G

Q = 【G,F,E】→【F,E】→【F,E,I】

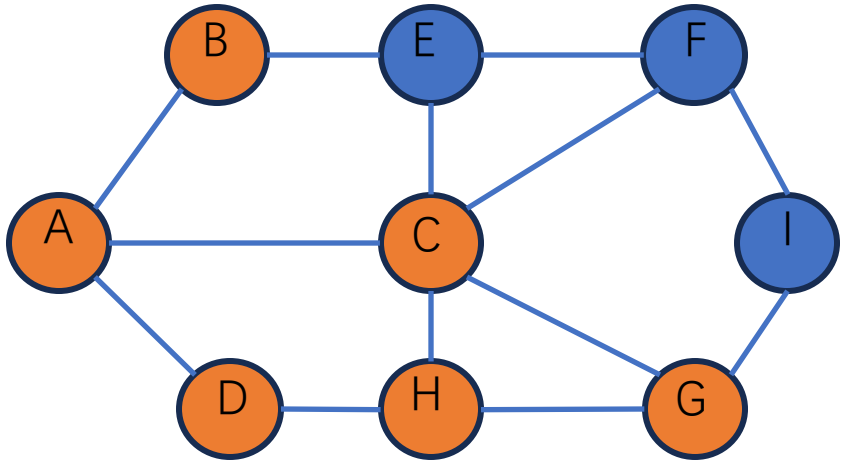
Visited = 【A,D,C,B,H,G,F,E】→

【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H】→【A,D,C,B,H,G】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【F,E,I】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G】

当前节点：F

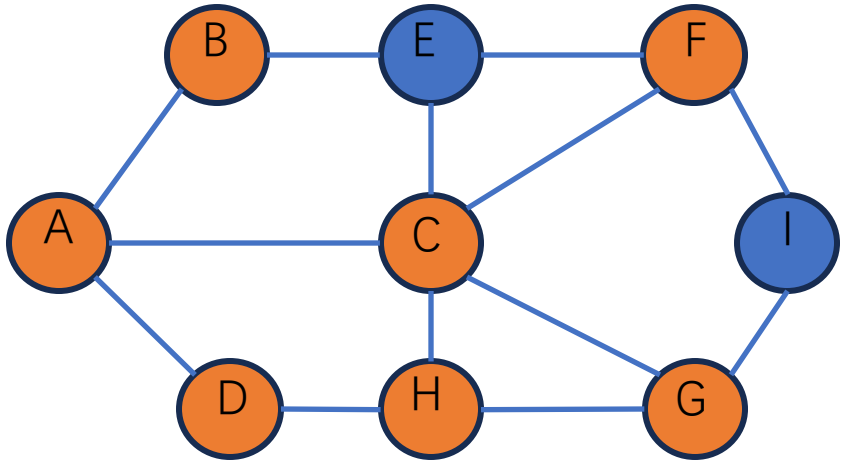
Q = 【F,E,I】→【E,I】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G】→【A,D,C,B,H,G,F】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【E,I】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G,F】

当前节点：E

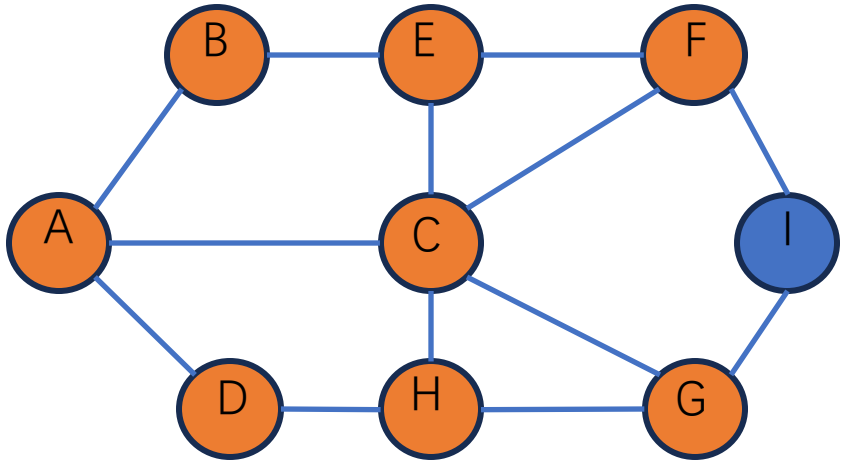
Q = 【E,I】→【I】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G,F】→【A,D,C,B,H,G,F,E】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【E,I】→【I】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G,F】→【A,D,C,B,H,G,F,E】

当前节点：E

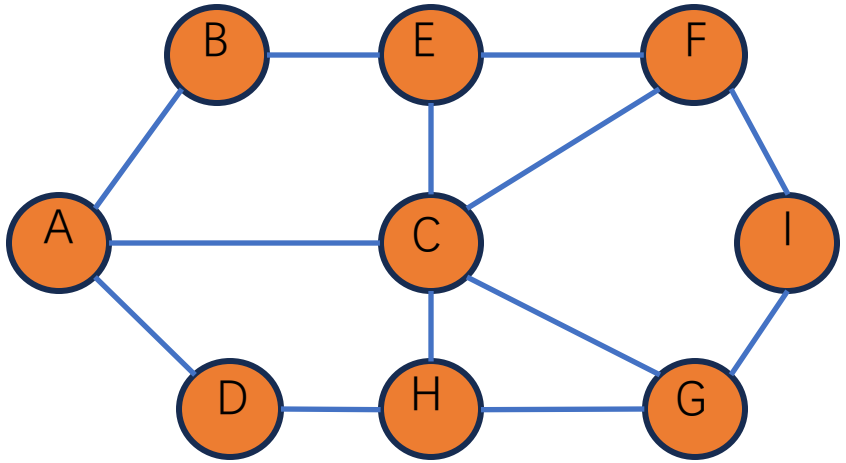
Q = 【I】→【】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G,F,E】→【A,D,C,B,H,G,F,E,I】

BFS遍历经典实现过程

```
当 Q 非空时:
    Q pop 出最先进（最左端）的节点为当前节点
    对于当前节点的邻居，若其不在 Visted 中：push入 Q 并“放入” Visited
    将当前节点“放入” Res 中
```



Q = 【】

Visited = 【A,D,C,B,H,G,F,E,I】

Res = 【A,D,C,B,H,G,F,E,I】

此刻 Q 为空，停止循环

算法实现细节问题

1. BFS遍历的输出结果是**唯一确定**的吗？
2. **Visited**用什么数据结构好？为什么？