图遍历算法详解

MLE 算法指北

2025 年 3 月 21 日

1 答题框架

1. **识别图的结构：** 邻接表、邻接矩阵、边列表等。
2. **确定图的类型：** 有向图 / 无向图，带权图 / 无权图。
3. **选择合适的算法：**
    - DFS / BFS：图遍历、找连通块
    - Union-Find：判断连通性、找环
    - 拓扑排序：依赖关系
    - 最短路径：Dijkstra / Bellman-Ford
4. **实现细节：** 访问标记、队列/栈、边界处理

2 图的遍历算法详解

图的遍历算法是解决图类问题的核心工具，主要包括深度优先搜索（DFS）、广度优先搜索（BFS）、拓扑排序、并查集和最短路径算法（如 Dijkstra）。本节将对这些算法的原理、公式及适用场景进行详细讲解。

2.1 1. 深度优先搜索（DFS）

原理： 使用递归或栈沿一个方向尽可能深入探索，再回溯至前一个分支。
时间复杂度： $O(V+E)$，其中 V 是顶点数，E 是边数。
适用场景：

- 图中连通块数量
- 检测环（有向图）
- 构建路径、递归枚举

代码框架：

```
DFS(u):
    visited[u] = true
    for each neighbor v of u:
        if not visited[v]:
            DFS(v)
```

2.2 2. 广度优先搜索（BFS）

原理： 使用队列，从起点开始按层（距离）扩展，逐层访问图节点。
时间复杂度： $O(V+E)$
适用场景：

- 最短路径（无权图）
- 分层遍历（如网格）
- 拓扑排序（Kahn's 算法）

代码框架：

MLE 算法指北

```
BFS(start):
    queue = [start]
    visited[start] = true
    while queue:
        u = queue.pop()
        for each neighbor v of u:
            if not visited[v]:
                queue.append(v)
                visited[v] = true
```

2.3 3. 拓扑排序（Topological Sort）

原理： 拓扑排序是对有向无环图（DAG）中所有节点的线性排序，使得对每一条边 $(u \to v)$，节点 u 都排在 v 前面。

两种实现方式：

- **DFS 逆后序遍历：** 回溯时加入排序结果中，最后反转
- **Kahn's 算法（BFS）：** 从入度为 0 的节点开始逐步剥离图

时间复杂度： $O(V + E)$ **Kahn 算法步骤：**

1. 统计每个节点的入度
2. 将入度为 0 的节点加入队列
3. 每次从队列取出节点，更新其邻居的入度
4. 若所有节点都被访问，则排序成功；否则存在环

2.4 4. 并查集（Union-Find）

原理： 并查集用于动态维护集合划分，支持合并（union）和查找（find）操作。

实现关键：

- **路径压缩（Path Compression）**：加速查找
- **按秩合并（Union by Rank/Size）**：加速合并

路径压缩：在查找过程中，让节点直接挂到根节点 1 上，从而降低树高。**复杂度：** 均摊 $O(\alpha(n))$，近乎常数级别

适用场景：

- 图中找环（如冗余边）
- 判断是否连通
- 最小生成树（如 Kruskal）

代码框架：

```
find(x):
    if parent[x] != x:
        parent[x] = find(parent[x])
    return parent[x]

union(x, y):
    rootX, rootY = find(x), find(y)
    if rootX == rootY:
        return
    parent[rootY] = rootX
```

2.5 5. 最短路径：Dijkstra 算法

原理： 适用于正权图。维护一个最小堆，按距离扩展最短路径。

复杂度：

- 使用堆实现：$O((V + E)\log V)$

核心思想： 每次扩展当前距离最短的点，并更新邻接点的最短距离。

适用场景：

- 单源最短路径（带权图）
- 网络延迟、传输最短时间

代码框架：

```
dist[source] = 0
minHeap = [(0, source)]
while minHeap:
    d, u = heappop(minHeap)
    for neighbor v:
        if dist[v] > d + weight(u, v):
            dist[v] = d + weight(u, v)
            heappush(minHeap, (dist[v], v))
```

3 1. 深度优先搜索（DFS）

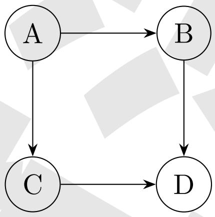
原理与应用

原理：沿路径向下递归探索到底，然后回溯。

适用场景：

- 图中连通块数量
- 判断图中是否有环
- 遍历所有路径、状态枚举

可视化图示（DFS）



遍历顺序 A-B-D-C

LeetCode 示例：200. Number of Islands

题意：统计网格中岛屿数量（连通的 1）

思路： DFS 遍历每个陆地，沉没访问过的陆地

Listing 1: DFS 解决岛屿数量

```
def numIslands(grid):
    if not grid:
        return 0

    rows, cols = len(grid), len(grid[0])

    def dfs(r, c):
        if r < 0 or r >= rows or c < 0 or c >= cols:
            return
        if grid[r][c] == '0':
            return
        grid[r][c] = '0'
        dfs(r+1, c)
        dfs(r-1, c)
        dfs(r, c+1)
        dfs(r, c-1)

    count = 0
    for r in range(rows):
        for c in range(cols):
            if grid[r][c] == '1':
                dfs(r, c)
                count += 1
    return count
```

—

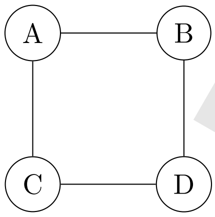
4 2. 广度优先搜索（BFS）

原理与应用

原理： 层级遍历，先访问距离近的节点。
适用场景：

- 无权图最短路径
- 多源扩散类问题

BFS 图示



遍历顺序 A-B-C-D

LeetCode 示例：542. 01 Matrix

题意： 给每个 1 距离最近的 0 的最短距离。
思路： 多源 BFS，从所有 0 向外扩展。

```
from collections import deque

def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    dist = [[float('inf')] * cols for _ in range(rows)]
    queue = deque()

    for r in range(rows):
        for c in range(cols):
            if mat[r][c] == 0:
                dist[r][c] = 0
                queue.append((r, c))

    directions = [(-1,0), (1,0), (0,-1), (0,1)]
    while queue:
        r, c = queue.popleft()
        for dr, dc in directions:
            nr, nc = r+dr, c+dc
            if 0 <= nr < rows and 0 <= nc < cols and dist[nr][nc] > dist[r][c] + 1:
                dist[nr][nc] = dist[r][c] + 1
                queue.append((nr, nc))

    return dist
```

—

5 3. 拓扑排序（Topological Sort）

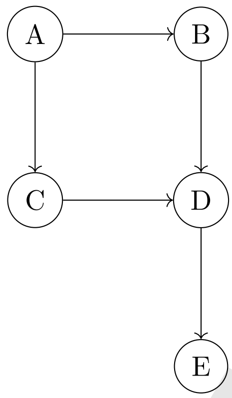
原理与应用

定义： DAG（有向无环图）上的线性排序。
适用场景：

- 课程/任务调度
- 依赖关系解析

ML算法指北

DAG 图示



拓扑排序可能结果：`A → B → C → D → E` 或 `A → C → B → D → E`

LeetCode 示例：210. Course Schedule II

题意： 输出课程完成的合法顺序。

思路： Kahn 算法（BFS + 入度）

```python
from collections import defaultdict, deque

def findOrder(numCourses, prerequisites):
    graph = defaultdict(list)
    indegree = [0] * numCourses

    for dest, src in prerequisites:
        graph[src].append(dest)
        indegree[dest] += 1

    queue = deque([i for i in range(numCourses) if indegree[i] == 0])
    res = []

    while queue:
        node = queue.popleft()
        res.append(node)
        for nei in graph[node]:
            indegree[nei] -= 1
            if indegree[nei] == 0:
                queue.append(nei)

    return res if len(res) == numCourses else []
```

—

6 4. 并查集（Union-Find）

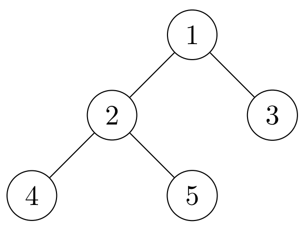
原理与应用

原理： 动态维护集合划分，查找是否连通。

适用场景：

- 检测无向图成环
- 计算连通分量

图示：并查集森林



ML算法指北

LeetCode 示例：684. Redundant Connection

题意： 找出导致图成环的边。
思路： Union-Find 判断是否已连通。

```
def findRedundantConnection(edges):
    parent = [i for i in range(len(edges)+1)]

    def find(x):
        if parent[x] != x:
            parent[x] = find(parent[x])
        return parent[x]

    def union(x, y):
        rootX, rootY = find(x), find(y)
        if rootX == rootY:
            return False
        parent[rootY] = rootX
        return True

    for u, v in edges:
        if not union(u, v):
            return [u, v]
```

—

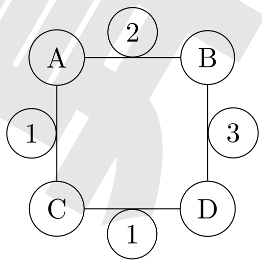
7 5. 最短路径：Dijkstra 算法

原理与应用

原理： 利用优先队列每次扩展最短路径。
适用场景：

- 正权图最短路径
- 网络延迟问题

图示：Dijkstra 路径



从 A 出发，最短路径为：`A → C → D`，总距离为 2。

LeetCode 示例：743. Network Delay Time

题意： 节点发出信号，多久传遍所有节点。
思路： Dijkstra 最短路径

```
import heapq
from collections import defaultdict

def networkDelayTime(times, n, k):
    graph = defaultdict(list)
    for u, v, w in times:
        graph[u].append((v, w))

    dist = {i: float('inf') for i in range(1, n+1)}
    dist[k] = 0
    heap = [(0, k)]

    while heap:
        time, node = heapq.heappop(heap)
```

```
        for nei, wt in graph[node]:
            if dist[nei] > time + wt:
                dist[nei] = time + wt
                heapq.heappush(heap, (dist[nei], nei))

    max_dist = max(dist.values())
    return max_dist if max_dist < float('inf') else -1
```

MLE 算法指北