算法与复杂性　作业九

516021910528 - SHEN Jiamin

2020 年 4 月 19 日

1　0413

1. 对于给定的二叉树，求其最小深度，即从根节点到最近的叶子的距离。

广度优先搜索最先到达的叶子节点有最小的深度。

最差情况下遍历了整棵树，时间复杂度为 $O(N)$，其中 N 为节点的数量。

算法 1 求二叉树的最小深度

输入: $root$（给定二叉树的根节点）

输出: $mindepth$（二叉树中的最小深度）

```
1:  q ← Queue.Init
2:  Queue.Push(q, ⟨root, 0⟩)
3:  while ¬Queue.isEmpty(q) do
4:      ⟨current, depth⟩ ← Queue.Pop(q)
5:      if current.isLeaf then                          ▷ 当前节点是叶子节点
6:          return depth                    ▷ BFS 最先到达的叶子节点有最小的深度
7:      end if
8:      if current.left then                                    ▷ 左子树入队
9:          Queue.Push(q, ⟨current.left, depth + 1⟩)
10:     end if
11:     if current.right then                                   ▷ 右子树入队
12:         Queue.Push(q, ⟨current.right, depth + 1⟩)
13:     end if
14: end while
```

2. 设 G 是有向非循环图，其所有路径最多含 k 条边。设计线性时间算法，将所有顶点分为 $k+1$ 组，每一组中任意两个点之间不存在路径。

DAG 中一定存在入度为 0 的节点，且一定存在出度为 0 的节点，最长路径出现在入度为 0 的节点和出度为 0 的节点之间（否则可以沿两端点继续扩展路径，得到更长的路径）。

按照拓扑排序的顺序，删除 DAG 中入度为 0 的节点后，上述最长路径的长度一定减小 1。且在拓扑排序中同一批被移除的节点之间一定不存在路径（否则删除起点时，终点的入度一定不为 0）。因此可按照拓扑排序中删除节点的批次将所有节点分为 $k+1$ 组，且每一组中任意两个节点之间不存在路径。

当图以出边邻接表给出时，算法的时间复杂度为 $O(|V|+|E|)$。

算法 2 拓扑分组

输入: $G=(V,E)$（给定 DAG 的出边表）

输出: $vtop\,[1\ldots|V|]\,, bnd$（$vtop\,[bnd[i]\ldots bnd[i+1]]$ 中的节点不存在路径）

```
1: vtop ← VECTOR.INIT, bnd ← VECTOR.INIT
2: indeg ← ARRAY.INIT(|V|, 0)                                  ▷ 构造入度表
3: for (v, w) in E do
4:     indeg[w] ← indeg[w] + 1
5: end for
6: for v in V where indeg[v] = 0 do                            ▷ 找到入度为 0 的节点
7:     VECTOR.PUSHBACK(vtop, v)
8: end for
                                                               ▷ 拓扑排序
9: inf ← 1, VECTOR.PUSHBACK(bnd, inf)
10: sup ← VECTOR.SIZE(vtop), VECTOR.PUSHBACK(bnd, sup)
11: while VECTOR.SIZE(vtop) < |V| do
12:     for idx ← inf to sup do          ▷ 对于每一个上一阶段找到的入度为 0 的节点
13:         for (v, w) in E where v = vtop[idx] do             ▷ 找到其后继节点
14:             indeg[w] ← indeg[w] − 1                        ▷ 减后继节点的入度
15:             if indeg[w] = 0 then                           ▷ 并检查是否减到了 0
16:                 VECTOR.PUSHBACK(vtop, w)
17:             end if
18:         end for
19:     end for
20:     inf ← sup + 1
21:     sup ← VECTOR.SIZE(vtop), VECTOR.PUSHBACK(bnd, sup)
22: end while
```

3. 给定连通无向图 G 以及 3 条边 a, b, c，在线性时间内判断 G 中是否存在一个包含 a 和 b 但不含 c 的闭链。

> 删除边 c，在剩余的图中判断是否存在包含 a, b 的回路。
>
> 在剩余的无向图中划分双连通分支（线性时间），若 a, b 在同一个双连通分支里，则存在一条包含 a, b 但不包含 c 的回路。

2 0416

1. 求 $n \times m$ 棋盘上任意两点之间马能够走的最短路径长度

> 构造图 $G = (E, V)$
>
> - E 为棋盘上所有格点
> - 若“马”能从棋盘上的点 v 经过一步走到 w，则 $(v, w) \in V$
>
> 对上图进行广度优先搜索即可找出最短路径长度。进一步地，构图和广度优先搜索可以同时进行。

2. 设计线性时间算法求树的最大匹配

> 树中一个非叶子节点连向其子节点的若干条边及连向其父节点的边中，至多只有一条边属于匹配。
>
> 记函数 $f(v, b)$ 为以 v 为根节点的子树中最大匹配的边数，其中 b 为布尔值
>
> - 当 $b = 1$ 时，存在一条与 v 直接相连的边在最大匹配中
> - 当 $b = 0$ 时，与 v 直接相连的边都不在最大匹配中
>
> 对于树的根节点 v，假设其有 3 个子节点 w_1, w_2, w_3 并有相应地与之相连的三条边 e_1, e_2, e_3，则
>
> $$f(v) = \max_b f(v, b) = \max \begin{cases} f(w_1, 1) + f(w_2, 1) + f(w_3, 1) & (b = 0) \\ \max \begin{cases} f(w_1, 0) + f(w_2, 1) + f(w_3, 1) + 1 \\ f(w_1, 1) + f(w_2, 0) + f(w_3, 1) + 1 \\ f(w_1, 1) + f(w_2, 1) + f(w_3, 0) + 1 \end{cases} & (b = 1) \end{cases}$$
>
> 观察可知对于每一棵子树，只需考察 $b = 1$ 和 $b = 0$ 两种情况；且 $b = 0$ 当且仅当对于其所有子树 $b = 1$。当 v 是叶子节点时，$f(v) = f(v) = 0$；所以当 v 的所有子节点都是叶子节点时，

$f(v) = f(v, 1) = 1$。对树做一次后序遍历，并按照上述规则计算出所有节点的 $f(v, 0)$ $f(v, 1)$ 即可。

每个节点被访问两次（下行入栈一次，上行出栈一次）；计算 f 时比较的情况数等于该节点的度数，所有节点的度数和为 $2|E|$。所以算法的时间复杂度为 $O(|V| + |E|)$

3. 无向图 G 的顶点覆盖是指顶点集合 U，G 中每条边都至少有一个顶点在此集合中。设计线性时间算法为树寻找一个顶点覆盖，并且使该点集的规模尽量小。

初始化所有结点的度。对于所有度数为 1 的节点，首先标记其相邻的点都在 U 中，然后删除与 U 中的点直接相邻的所有边。重复上述步骤，直至所有边都被删除。