数据结构与算法 II 作业 (12.8)

中国人民大学　信息学院　崔冠宇　2018202147

24.1-3　给定 $G = (V, E)$ 是一带权重且没有权重为负值的环路的有向图，对于所有结点 $v \in V$，从源结点 s 到结点 v 之间的最短路径中，包含边的条数的最大值为 m。（这里，判断最短路径的根据是权重，不是边的条数。）请对算法 BELLMAN-FORD 进行简单修改，可以让其在 $m + 1$ 遍松弛操作之后终止，即使 m 不是事先知道的一个数值。

解: 类似于冒泡排序中维护一个变量记录一趟扫描中是否发生交换，可以将 BELLMAN-FORD 算法中的松弛操作修改一下，记录一趟松弛过程中是否有某节点的距离上界发生了更新。

```
RELAX-MODIFIED(u, v, w):
    if u.d + w(u, v) < v.d:
        updated = true
        v.d = u.d + w(u, v)
        v.from = u
BELLMAN-FORD-MODIFIED(G, w, s):
    INITIALIZE-SINGLE-SOURCE(G, s)
    updated = true
    while updated:
        updated = false
        for each edge (u, v):
            RELAX-MODIFIED(u, v, w)
```

正确性的简单说明：根据路径松弛性质和收敛性质，若各结点都没有发生更新，说明已经进行了 $m + 1$ 轮松弛（最后一轮才发现没有更新）。

24.3-8　给定带权重的有向图 $G = (V, E)$，其权重函数为 $w : E \to \{0, 1, 2, \cdots, W\}$，这里 W 为某个非负整数。请修改 Dijkstra 算法来计算从给定源结点 s 到所有结点之间的最短路径。该算法时间应为 $O(WV + E)$。

解: 因为各边的权 $w(u, v)$ 有上界 W，而 V 个结点的图的最短路径树中任何路径的边数不超过 $V - 1$，因此最短路径长度有上界 WV （没有路径除外）。

通过一个数组 `A[0...WV+1]` 来实现优先队列：运行 Dijkstra 算法时，当一个顶点的最短路径上界为 d，将它放在 `A[d]` 的链表中（`A[WV+1]` 存放距离上界为 $+\infty$ 的结点）。由于边权非负，所以 EXTRACT-MIN 只需要不断向下标增大的方向寻找即可，于是用聚合分析的方法，最多只需要从下标为 0 处扫描至下标为 $WV + 1$ 处，复杂度 $O(WV)$。修改键值只需要简单将结点挪到 `A[d']` 链表的头部，单次时间复杂度为 $O(1)$，共进行 $O(E)$ 次。因此总时间复杂度为 $O(WV + E)$。

25.2-7　在 Floyd-Warshall 算法中构建最短路径的另一种办法是使用 $\phi_{ij}^{(k)}$，其中 $i, j, k = 1, 2, \cdots, n$，$\phi_{ij}^{(k)}$ 是从结点 i 到结点 j 的一条中间所有结点都取自集合 $\{1, 2, \cdots, k\}$ 的最短路径上编号最大的中间结点。请给出 $\phi_{ij}^{(k)}$ 的一个递归公式，并修改 Floyd-Warshall 过程来计算 $\phi_{ij}^{(k)}$ 的值，并重写 PRINT-ALL-PAIRS-SHORTEST-PATH 过程，使其以矩阵 $\Phi = (\phi_{ij}^{(n)})$ 作为输入。矩阵 Φ 与 15.2 节所讨论的链式矩阵乘法中的表格存在何种相似点?

解: $\phi_{ij}^{(k)}$ 的含义如同题意。$\phi_{ij}^{(k)}$ 递归定义如下:

$$\phi_{ij}^{(k)} = \begin{cases} NIL, & k = 0 \\ k, & k \geq 1 \text{且} d_{ik}^{(k-1)} + d_{kj}^{(k-1)} < d_{ij}^{(k-1)} \\ \phi_{ij}^{(k-1)}, & k \geq 1 \text{且} d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \geq d_{ij}^{(k-1)} \end{cases}$$

修改后的 FLOYD-WARSHALL 和 PRINT-ALL-PAIRS-SHORTEST-PATH 如下:

```
FLOYD-WARSHALL-MODIFIED(W):
    n = W.rows
    // 初始矩阵
    D_0 = W
    Phi_0 = n * n matrix of NIL
    // 迭代 n 次
    for k = 1 to n:
        D_k = matrix of (d_k[i][j])
        Phi_k = matrix of (phi_k[i][j])
        // 遍历矩阵元素
        for i = 1 to n:
            for j = 1 to n:
                // 更新
                if d_(k-1)[i][k] + d_(k-1)[k][j] < d_(k-1)[i][j]:
                    d_k[i][j] = d_(k-1)[i][k] + d_(k-1)[k][j]
                    phi_k[i][j] = k
                else:
                    d_k[i][j] = d_(k-1)[i][j]
                    phi_k[i][j] = phi_(k-1)[i][j]
```

```
PRINT-ALL-PAIRS-SHORTEST-PATH-MODIFIED(Phi, i, j):
    // 仅一顶点
    if i == j:
        print(i)
    // 没有更新过
    else if Phi[i][j] == NIL:
        // 要么是直接可达
        if W[i][j] < INFINITY:
            print(i, j)
        // 要么没有路
```

```
        else:
            print("No path from", i, "to", j)
    // 递归
    else:
        PRINT-ALL-PAIRS-SHORTEST-PATH-MODIFIED(Phi, i, Phi[i][j])
        PRINT-ALL-PAIRS-SHORTEST-PATH-MODIFIED(Phi, Phi[i][j], j)
```

本题中矩阵 Φ 和链式矩阵乘法的功能很像，都是在递归调用产生解时起到了中介作用，都是将原问题分解成左右两个子问题解决。