5 最短路径：标签矫正算法

5.1 介绍

对于具有负循环的一般网络，寻找最短路径显得非常困难。需要考虑当存在负循环时，能够识别负循环，当网络不包含负循环，可以解决最短路径问题的算法。

标签校正算法，可以用于解决任意弧长的最短路径问题。

5.1 最优性条件

最短路径最优性条件:

定理 5.1：对于任意节点 j ∈N，设 d（j）表示从源节点 s 到节点 j 的有向路径的长度。当且仅当它们满足以下最短路径最优性条件时 d（j）表示最短路径距离:

$$d(j) \leq d(i) + c_{ij} \qquad \text{for all } (i,j) \in A. \qquad (5.2) \quad \blacklozenge$$

这一定理可以通过数学归纳法证明

最短路径最优性条件为一组距离标签定义最短路径长度提供了充要条件, 在标签校正算法的发展中起着核心作用。

定义弧缩减长度 reduced arc length $c_{ij}^{d}=c_{ij}+d（i）-d（j）$

对弧缩减长度的理解：如果从源节点到节点 j 必须经过弧 ij，所引起的最短路径长度的变化量

定理 5.2:

a）对于任意有向循环 W，$\sum_{(i,j)\in W} c_{ij}^{d} = \sum_{(i,j)\in W} c_{ij}.$

b）对于从节点 k 到节点 l 的任意有向路径 P，$\sum_{(i,j)\in P} c_{ij}^{d} = \sum_{(i,j)\in P} c_{ij} +$ d（k）-d（l）

c） 如果 d（*）代表最短路径距离，对于每个弧（i，j）∈A，$c_{ij}^{d}>=0$。

a 和 b 的证明来自缩短弧长的定义式，c 的证明来自定理 5.1

5.3 通用标签校正算法

标签校正算法在每个节点上保持一个距离标签，并迭代更新这些标签，直到距离标签满足最优性条件。

```
algorithm label-correcting;
begin
    d(s) : = 0 and pred(s) : = 0;
    d(j) : = ∞ for each j ∈ N − {s};
    while some arc (i, j) satisfies d(j) > d(i) + c_ij do
    begin
        d(j) : = d(i) + c_ij;
        pred(j) : = i;
    end;
end;
```

Figure 5.1 Generic label-correcting algorithm.

前置图:

在运算过程中每个节点 i 都有几个前置节点，称为 pred（i)。由每个弧（pred（i)，i）组成的图称为前置图。前置图上的弧将始终满足 $c_{ij}^{d} \leq 0.$ 的不变性（我觉得应该是 $c_{ij}^{d}=0$）.

算法结束时，前置图将成为最短路径树，该算法进行 O（n^2C）次迭代，当 C 的大小是指数级的时候，算法复杂度为 O（2^n)。

复杂度：O（min{n^2mC，$m2^n$}）

特点：

1.选择违反最优性条件的弧并更新距离标签。

2.需要 O（m）时间来识别违反最优性条件的弧。

3.非常一般：大多数最短路径算法都可以看作是该算法的特例。

4.运行时间为伪多项式，不具有吸引力。

改进的标签校正算法

一般的标签校正算法任意选择违反最优性条件的弧，并用它来更新距离标签。通常，识别违反最优性条件的弧将是通用标签校正算法的瓶颈，因为每次迭代都需要 O(m) 的时间，改进的标签校正算法能将这一时间减少到 O（m/n)。

```
algorithm modified label-correcting;
begin
    d(s) : = 0 and pred(s) : = 0;
    d(j) : = ∞ for each node j ∈ N − {s};
    LIST : = {s};
    while LIST ≠ ∅ do
    begin
        remove an element i from LIST;
        for each arc (i, j) ∈ A(i) do
        if d(j) > d(i) + c_ij then
        begin
            d(j) : = d(i) + c_ij;
            pred(j) : = i;
            if j ∉ LIST then add node j to LIST;
        end;
    end;
end;
```

Figure 5.5 Modified label-correcting algorithm.

这一方法的核心在于每当向列表中添加弧时，都会添加从单个节点（其距离标签减小）发出的所有弧。这表明，我们不必维护可能违反最优性条件的所有弧的列表，而是可以维护一个节点列表。

复杂度：O(min{nmC, m2^n})
特点：

1.改进了通用标签校正算法。

2.该算法维护了一组节点列表：每当距离标签 d（j）发生变化时，我们就将节点 j 添加到列表中。该算法从列表中删除节点 i，并检查 a（i）中的弧以更新距离标签。

3.非常灵活，因为我们可以用多种方式维护列表。

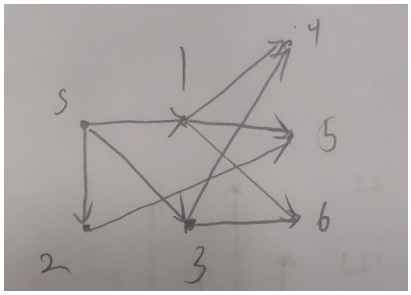
4.运行时间仍然不吸引人。

FIFO 实现

将集合列表作为队列，避免了最大弧长 C 对算法效率的影响。

复杂度：O(nm)

我觉得这个问题可以这样理解：



在如图所示的网络中有 7 个节点，在第一次迭代时节点 1、2、3 将首先进入 LIST，这一次迭代将计算所有通过由 1 个或更少弧组成的最短路径（即和 s 之间相连）连接到源节点的节点的最短路径距离，将 1、2、3 作为一批进行第二“波”迭代，4、5、6 进入 LIST，这一波迭代将计算所有通过由 2 个或更少弧组成的最短路径（即和 s 通过一条或两条弧可以连接）连接到源节点的节点的最短路径距离，这样，最多需要 n-1 波迭代就能找到所有最短路，每一次迭代最多把所有弧遍历一次（m），因此复杂度为 O（nm），我认为这种一波一波的迭代方法是 fifo 实现的巧妙之处。

特点：

1.改进的标签矫正算法的具体实现。

2.将集合列表作为队列进行维护，从而按先进先出的顺序检查列表中的节点。

3.实现了求解任意弧长最短路径问题的最佳强多项式运行时间。

4.实践效率比较高。

5.在 O（nm）时间内，还可以识别负循环的存在。

dequeue 实现

复杂度：O（min{nmC，m2^n}）

特点：

1.改进的标签校正算法的另一个具体实现。

2.维护集合列表为 dequeue。如果算法先前更新了其距离标签，则在 dequeue 的前面添加这个节点，否则在后面添加这个节点。

3.在实践中非常有效（可能是线性时间）。

4.最坏情况下的运行时间没有吸引力。

5.5 识别负循环

以重复的间隔检查前置图是否包含有向循环：首先将源节点指定为已标记，将所有其他节点指定为未标记。然后，一个接一个地检查每个未标记的节点 k 并执行以下操作：标记节点 k，跟踪从节点 k 开始的前置索引，并标记遇到的所有节点，直到到达第一个已标记的节点，比如节点 l。如果 k=l，前置图包含一个循环

FIFO 标签校正算法能够很容易地检测负循环的存在：在每"波"迭代中每个节点最多出现 1 次，最多检测 n-1 波，因此每个节点最多被检测 n-1 次，可以通过记录节点被检测的次数检测负循环

5.6 全对最短路径问题（用一种算法求出任意节点间的最短路）

1.对于每个点都使用一次最短路径算法，如果 S（n，m，C）表示求解弧长非负的最短路径问题所需的时间，则该方法需要 O（nS（n，m，C））的时间求解全对最短路径问题

2. floyd 算法

```
algorithm Floyd–Warshall;
begin
    for all node pairs [i, j] ∈ N × N do
            d[i, j] : = ∞ and pred[i, j] : = 0;
    for all nodes i ∈ N do d[i, i] : = 0;
    for each arc (i, j) ∈ A do d[i, j] : = c_ij and pred[i, j] : = i;
    for each k : = 1 to n do
            for each [i, j] ∈ N × N do
                if d[i, j] > d[i, k] + d[k, j] then
                begin
                    d[i, j] : = d[i, k] + d[k, j];
                    pred[i, j] : = pred[k, j];
                end;
end;
```

Figure 5.7 Floyd–Warshall algorithm.

复杂度：O（n3）

特点：

1.系统地更正距离标签，直到它们表示最短路径距离。

2.很容易实现。

3.实现密集网络的最佳可用运行时间。

4.需要 O（n^2）空间储存。

在 Floyd-Warshall 算法中，只要在对某个节点 i 更新 d[i，i]时检查条件 d[i，i]<0，就可以检测出负循环的存在。