研究生算法课课堂笔记

上课日期：2014-12-1 第(2)节课

组长：杨小东　　组员：杨恺

组员：杨彬　　组员：吴磊

注意：请提交 Word 格式文档。

---

1. SPFA 算法

（1） 处理的问题

对于给定的带权图 G(V,E),求 G 上所有节点到汇点 t 的最短距离。

（2） 算法描述

维护一个队列，里面存放所有需要进行迭代的点。初始时队列中只有一个点 t。用一个布尔数组记录每个点是否处在队列中。每次迭代，取出队首的点 v,依次枚举到 v 的边 u->v,设边的权重为 w(u,v),判断 d[v]+w(u,v)是否小于 d[u]，若小于则 d[u] = d[v]+ w(u,v)，如果 u 不在队列中，就将它放入队尾。这样一直迭代下去直到队列变空，结束算法。

```
procedure Shortest-Path-Faster-Algorithm(G, t)
 1   for each vertex v ≠ t in V(G)
 2       d(v) := ∞
 3   d(t) := 0
 4   Q.enque(t)
 5   while Q is not empty
 6       v := Q.deque()
 7       for each edge (u, v) in E(G)
 8           if d(v) + w(u, v) < d(u) then
 9               d(u) := d(u) + w(u, v)
10               if u is not in Q then
11                   Q.enque(u)
```

（3） 反向追踪

维护一个节点数组 path[]，在执行前边的算法每当 d[u]发生更新，则记录 path[u] =v;最后根据 path 数组就可以得到每个点到汇点的最短路径。

（4） 加速原因

1）每次只需要存储可以松弛的边

2）边松弛顺序趋近于最短路径的逆序。

（5） 时间复杂度分析

时间复杂度：O(k|E|)， 其中 k 为所有顶点进队的平均次数，k 一般小于等于 2。SPFA 在稀疏图上的表现的确非常不错，但稠密图会让它加速原因 1 不存在。所以表现的不如稀疏图那么好。

（6） 和 BFS，Dijkstra 算法的区别

1）和 BFS 的区别

基本思路类似，但是所不同的是 BFS 中每个节点只能被访问一次，而

SPFA 算法中有的节点可以多次访问。

2）和 Dijkstra 算法（优先队列实现）的区别

SPFA 使用 FIFO 队列实现，而 Dijkstra 算法使用优先队列实现。SPFA 中每个结点可进出队列多次，而 Dijkstra 中每个结点进出优先队列一次。

（7） 负环检测

为每个顶点设置一个更新次数的计数器，用来记录该顶点进入队列的次数。如果有某个顶点的计数器值大于等于 n,那么该图存在负环。

2. 多源最短路径算法

（1）处理的问题

对于给定的带权图 G(V,E),求 G 上所有顶点对(u,v)间的最短距离。

（2）算法描述

1）朴素解法

对每个顶点使用单源最短路径算法。

2）Floyd-Warshall 算法

Floyd-Warshall 算法是解决任意两点间的最短路径的一种算法。通常可以在任何图中使用，包括有向图、带负权边的图。

Floyd 算法是一种基于 DP(Dynamic Programming)的最短路径算法。

设图 G 中 n 个顶点的编号为 1 到 n。令 c [u,v,k]表示 u 经顶点 1…k 到 v 的最短路径长度（k 表示该路径中的最大顶点编号），也就是说 c[u,v,k]这条最短路径所通过的中间顶点最大不超过 k。因此，如果 G 中包含边<u,v>，则 c[u,v,0] =边<u,v>的长度(e(u,v))；若 u=v，则 c[u,v,0]=0；如果 G 中不包含边<u,v>，则 c[u,v,0]= +∞(e(u,v)= +∞)。c[u,v,n]则是从 u 到 v 的最短路径的长度。

对于任意的 k>0，通过分析可以得到：中间顶点不超过 k 的 u 到 v 的最短路径有两种可能：该路径含或不含中间顶点 k。若不含，则该路径长度应为 c[u,v,k-1]，否则长度为 c[u,k,k-1] +c [k,v,k-1]。c[u,v,k]可取两者中的最小值。

把上述的分析精简化为动态转移方程：

$$OPT(u,v) = e(u,v), k = 0$$

$$OPT(u,v) = \min\{OPT(u,v), OPT(u,k) + OPT(k,v)\}, k > 0$$

Floyd 算法转化为伪代码：

```
procedure Floyd-Warshall Algorithm
1    for i = 1 to n
2        for j = 1 to n
3            d(i,j) = (i==j)? 0:INF;
4            if 有一条有向边连接(i,j)
5                d(i,j) = e(i,j);
6     for k = 1 to n
7         for i = 1 to n
8             for j = 1 to n
9                 d(i,j) = min{d(i,j), d(i,k)+d(k,j)};
```

（3）反向追踪

Floyd 算法不仅能够给出两点之间的最短路径距离，还能给出路径上具体有哪些点：用一个与邻接矩阵 A 相等维数的矩阵 P 来记录。P 中的每一个元素 next(i,j)表示 i 到 j 的最短路径中 i 的下一个顶点序号：

初始化时，next(i,j) = NaN，if e(i,j)∉E，否则 next(i,j) = j;

Floyd 算法更新 d(i,j) = d(i,k) + d(k,j)时，next(i,j) = next(i,k);

输出时，要输出 i 与 j 的最短路径上的顶点：

```
Output( i, j)
1    Printf(i)
2    If next(i,j) == NaN return
3    Output( next(i,j),j)
```

（4）时间空间复杂度

这个算法使用邻接矩阵来储存边，同时邻接矩阵也用来储存算法运行过程中的 d(i,j)，因此算法的空间复杂度是 $O(n^2)$，时间复杂度是 $O(n^3)$。

（5）负环检测

1）检查 d(i,i)的值，也就是邻接矩阵中对角线的值，如果 d(i,i) < 0，则说明有负环。因为一个环中，一个点必定可以通过这个环回到自己，如果这个距离为负值，说明这个环是负环。

2）Floyd 算法运行结束后得到最短距离矩阵 D，对邻接矩阵 A 求转置 A^T，若 B = D + A^T这个矩阵中有某个元素 d(i,j) < 0，则图中有负环。因为对 D 中的某个元素 d(i,j)，表示的是 i 到 j 的最短距离，转置后 A^T中的相应位置的元素 a(j,i)表示 j 到 i 的边的权重，两者相加即为一个环的权重，如果为负值，说明这个环是负环。

3. 双重标准的“较短”路径：

我们之前讨论的图的最短路径是单一标准的最短路径，但是在现实生活中，两个顶点的距离标准可能不唯一，即连接两个顶点的边上有两个或者更多权重。此处先考虑两个标准的最短路径。

在双重标准下的最短路径不是良定义的：

找不到一条由 s 到 t 的路径能够使两个标准都达到最短，那么能否给出一条路径，使得结果不算太差：双重标准都满足要求的范围。

（1）问题描述：

在图 G（V，E）中，边（u，v）表示顶点 u 和 v 之间有一条直达路径，在这条路径上，有两个权值：d_1（u，v）、d_2（u，v），能否有一个算法找出一条“相对可以接受”的路径，使得 $\sum d_1 \le L_1$； $\sum d_2 \le L_2$

（2）同学提出：

根据这两个标准，分别求出他们的最短路径，然后看这两个边集合有没有交集。这个想法求出的路径都是单一标准下最短路径的组成边。但是，存

在这样的边，它在单一标准下不能构成最短路径，因此不在这两个集合中，但是它可以构成两个标准下满足条件的路径。

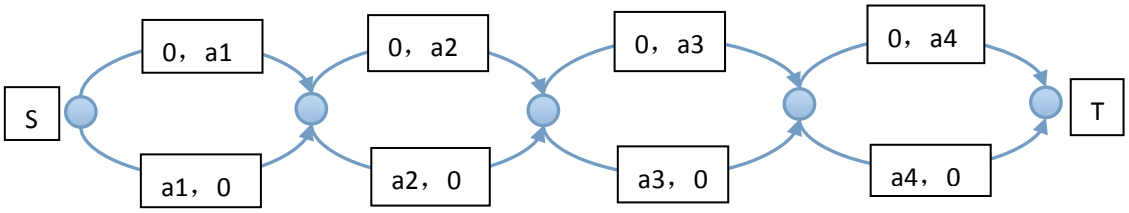
实际上这个问题是 NP 完全问题，子集和问题可以归约到这个问题。

（3）证明：

1）如果 Y 是一个 NP 完全问题，X 属于 NP 且 $Y \leq_p X$，则 X 是 NP 完全的。

2）子集和问题描述：给定一个正整数集合 S 和整数目标 W，试问是否存在一个子集 $S' \subseteq S$，其元素之和为 W。

问题的关键是如何规约到子集和问题。



构造：

这样一个图 G（V，E）如上图所示，有五个节点包括源点 S 和汇点 T，节点依次定义为 x1，x2，x3，x4，x5。每两个相邻的节点都有两条平行的边连接，例：x1 到 x2 的下面的边定义为 e1 和上面的边 e1'，后面的边定义以此类推，每条边都有两个权值 d1 和 d2。

我们构建的这个图是一个容易规约的例子，当然，如果不习惯相邻点有两条边，那么可以将其中一条边添加一个点，其本质和这个图是一样的。

规约：

从 S 到 T 的路径必须从{ ei，ei' }中选择一条来走。这四种独立的选择，对应子集和问题中的选择，选择 ei 相当于将 d1=ai 加入集合 P 中，选择 ei' 相当于只将 d2=ai 加入 Q 集合中。假设随便一条路径由 S 到 T，得到 $\sum P = W$；

$\sum_{i=1}^{4} a_i = W_0$，那么 $\sum Q = W_0 - W$。那么我们不妨就设两个标准的范围 $L_1 = W$；

$L_2 = W_0 - W$。若要同时满足 $\sum d_1 \leq L_1$；$\sum d_2 \leq L_2$，只能选择特定路径使得两个集合 $\sum P = W$；$\sum Q = W_0 - W$。问题就转化为在集合{a1，a2，a3，a4}中选择子集 S，使得子集元素之和等于 W。至此，将此问题规约成子集和问题，该图的解法是 NP 完全问题，这个图是我们构造的双标准下最短路径的一个特例，因此，双标准的最短路径是 NP 完全问题。

（4）Dijkstra、SPFA、Floyd-Warshall 算法的比较

算法	思想	应用场合	实现	进出队列次数

Dijkstra	贪心：基于已经确定最短路径的顶点更新尚未确定最短路径的顶点的路径值	求解单源点最短路径；边的权值非负	优先队列+邻接表（Dijkstra 算法有多种实现形式，详见附录 1）	每个结点进出最小堆一次
SPFA	动态规划：合理安排边的松弛顺序，避免 Bellman-Ford 算法的无用松弛	求解单汇点最短路径；有负边，无负环	队列+邻接表	平均情形下，每个结点进出队列 k 次，k 值因图而变化
Floyd-Warshall	动态规划：第 k 轮循环时，基于最优子结构更新(i,j)以 1…k 为中间结点时的最短路径	求解多源点最短路径；有负边，无负环	邻接矩阵	不需要进出队列

附录 1：Dijkstra 算法的实现

PQ implementation	insert	delete-min	decrease-key	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman-Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
Brodal queue (Brodal 1996)	$O(1)$	$O(\log n)$	$O(1)$	$O(m + n \log n)$