Section 2.4 Shortest Paths 最短路径

译 by tim green

Sample Problem: Overfencing [Kolstad & Schrijvers, Spring 1999 USACO Open]

农夫 John 在外面的田野上搭建了一个巨大的用栅栏围成的迷宫。幸运的是，他在迷宫的边界上留出了两段栅栏作为迷宫的出口。更幸运的是，他所建造的迷宫是一个“完美的”迷宫：即你能从迷宫中的任意一点找到一条走出迷宫的路。

给定迷宫的宽 W(1

```
+-+-+-+-+-+
|       |
+-+ +-+ + +
|   | | |
+ +-+-+ + +
| |     |
+-+ +-+-+-+
```

如上图的例子，栅栏的柱子只出现在奇数行或奇数列。每个迷宫只有两个出口。

The Abstraction

给出:一个边的权为非负的有向图

两个顶点间的一条路径是由图中相邻的边以任意的顺序连接而成的

两个顶点之间的最短路径是指图中连接这两个顶点的路径中代价最小的一条，一条路径的代价是指路径上所有边的权的和

题目常常只要求出最短路径的代价而不需要求出路径本身。在这个例子中只用求出迷宫内部的点和出口之间的最短距离(代价)。最别的，它需在求出所有代价中最大的一个。

用 Dijkstra 算法来求加权图中的最短路径

给出:所有的顶点、边、边的权，这个算法按离源点的距离从近到远的顺序来"访问"每个顶点。

开始时把所有不相邻顶点间的距离(边的权)标为无穷大,自己到自己的距离标为 0。

每次,找出一个到源点距离最近并且没有访问过的顶点 u。现在源点到这个顶点的距离就被确定为源点到这个顶点的最短距离。

考虑和 u 相邻的每个顶点 v 通过 u 到源点的距离。考查顶点 v,看这条路径是不是更优于已知的 v 到源点的路径,如果是,更新 v 的最佳路径。

[In determining the shortest path to a particular vertex, this algorithm determines all shorter paths from the sou rce vertex as well since no more work is required to calculate all shortest paths from a single source to vertic es in a graph.]

参考: [Cormen, Leiserson, Rivest]的第 25 章

Pseudocode:

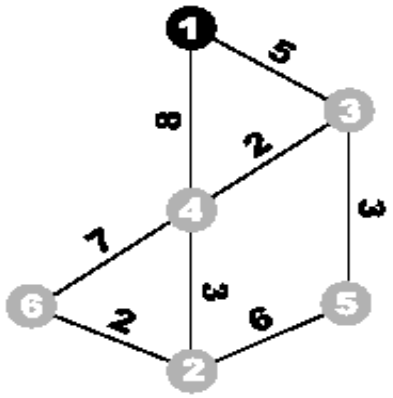
```
# distance(j) is distance from source vertex to vertex j
# parent(j) is the vertex that precedes vertex j in any shortest path
# (to reconstruct the path subsequently)
1 For all nodes i
2 distance(i) = infinity # not reachable yet
3 visited(i) = False
4 parent(i) = nil # no path to vertex yet
5 distance(source) = 0 # source -> source is start of all paths
6 parent(source) = nil
7 8 while (nodesvisited
```

这个算法的时间效率为 O(V2)。你可以使用堆来确定下一个要访问的顶点来获得 O(E log V)的效率(这里的 E 是指边数 V 是指顶点数)，但这样做会使程序变得复杂，并且在一个大而稀疏的图中效率只能提高一点点。

Sample Algorithm Execution

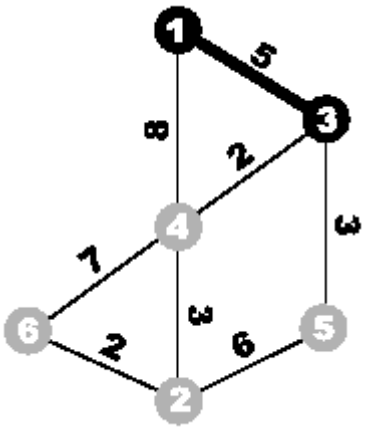
考虑下面的图

这是问题的初始状态:



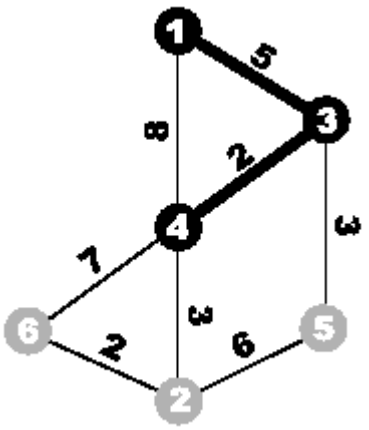
和顶点 1 相邻的有顶点 3、4

现在还没被访问的顶点中顶点 3 到源点的距离最小，所以它是下一个要被访问的点:

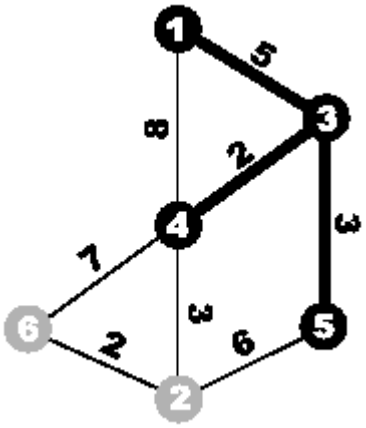


和顶点 3 相邻的顶点有 4、5 更新这些顶点:

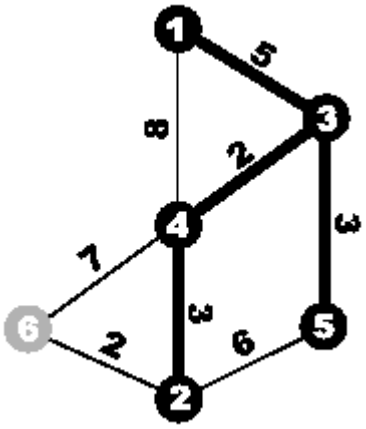
现在还没被访问的顶点中顶点 4 到源点的距离最小，它的邻点有 1、2、3、6, 因为别的点都已经被访问过了，所以只有顶点 2、6 需要被更新:



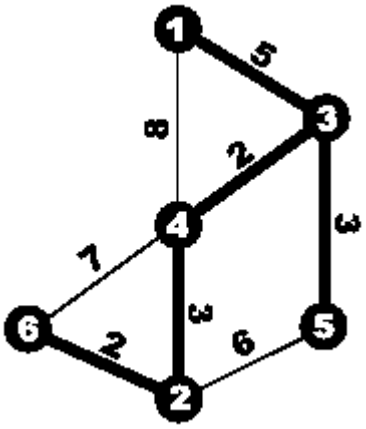
在剩下的三个顶点(2, 5, 和 6)中, 顶点 5 最接近源点，所以要被访问。它的邻点有 2、3,只有顶点 2 没有被访问过。经顶点 5 再到顶点 2 的距离是 14,比经顶点 4 的路径长(那条长为 10),所以顶点 2 不需要更新。



在剩下的两个顶点中最接近的是顶点 2 ，它的邻点是顶点 4、5、6,只有顶点 6 没被访问过。另外顶点 6 要被更新:



最后，只有顶点 6 没被访问过，它所有的邻点都被访问过了:



Sample Problem: Package Delivery 投递包裹

给出一个地点的集合，和连接它们的道路的长，和一个按排好的包裹要投递的地点的清单。找出按顺序投递所有包裹的最短路程。

分析:对于每一次投递，使用 Dijkstra 算法来确定两个点间的最短路程。如果要投递的次数超过 N，我们就用计算每对点之间的最短距离来代替计算每投递的最短路程，最后只用简单把每一次投递一路程相加就是答案。

Extended Problem: All Pairs, Shortest Paths 每对点之间的最短路程

这个扩展问题就是确定一个表格 a：

这里 ai,j = 顶点 i 和 j 之间的最短距离，或者无穷大如果顶点 i 和 j 无法连通。

这个问题常常是其它大的问题的子问题，如投递包裹。

Dijkstra 算法确定单源最短路径的效率为 O(N2)。我们在每个顶点上执行一次的效率为 O(N3)。

如果不要求输出路径，还有一个更简单的算法效率也是 O(N3)

The Floyd-Warshall Algorithm

Floyd-Warshall 算法用来找出每对点之间的最短距离。它需要用邻接矩阵来储存边，这个算法通过考虑最佳子路径来得到最佳路径。

注意单独一条边的路径也不一定是最佳路径。

从任意一条单边路径开始。一个两点之间的距离是边的权，或者无穷大如呆两点之间没有国相连。
对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比已知的路径更短。如果是更新它。

不可思议的是，只要按排适当，就能得到结果。

想知道更多关于这个算法是如何工作的，请参考[Cormen, Leiserson, Rivest]第 26 章。

```
Pseudocode:
# dist(i,j) is "best" distance so far from vertex i to vertex j
# Start with all single edge paths.
For i = 1 to n do
  For j = 1 to n do
    dist(i,j) = weight(i,j)
For k = 1 to n do # k is the `intermediate' vertex
  For i = 1 to n do
    For j = 1 to n do
        if (dist(i,k) + dist(k,j)
```

这个算法的效率是 O(V3)。它需要邻接矩阵来储存图。

这个算法很容易实现，只要几行。

即使问题是求单源最短路径，还是推荐使用这个算法，如果时间和空间允许(只要有放的下邻接矩阵的空间，时间上就没问题)。

Problem Cues

如果问题要求最佳路径或最短路程，这当然是一个最短路径问题。即使问题中没有给出一个图，如果问题要求最小的代价并且这里没有很多的状态，这时常常可以构造一个图。最大的一点要注意的是:最短路径 = 搜索做某件事的最小代价。

Extensions

如果图是不加权的，最短路径包括最少的边。这种情况下用宽优搜索(BFS)就可以了。如果图中有很多顶点而边很少，这样会比 Dijkstra 算法要快(看例子 Amazing Barn)

如果充许负权边，那么 Dijkstra 算法就错了。幸运的是只要图中没有负权回路 Floyd-Warshall 算法就不受影响(如果有负权回路，就可以多走几圈得到更小的代价)。所以在执行算法之前必须要检查一下图。

可以再增加一些条件来确定最短路径(如，边少的路径比较短)。只要用从距离函数中得到比较函数，问题就是一样的。在上面的例子中距离函数包括两个值 代价和边数。两个值都要比较如果必要的话。

Sample Problems

Graph diameter 图的直径

给出:一个无向无权的连通图,找出两点距离最远的顶点。

分析:找出所有点之间的最短距离，再找出最大的一个。

Knight moves 爵士的移动

给出:两个 N*N 的棋盘.确定一个爵士从一个棋盘移动到另一个所需的最少步数。

分析:把棋盘变成一个 64 个顶点的图。每个顶点有 8 条边表示爵士的移动。

Amazing Barn (abridged) [USACO Competition Round 1996]

考虑一个非常大的谷仓由 N(N "最中心的畜栏",一个畜栏到其它的平均距离最小则被认为是"最中心的畜栏"的。

分析:计算每对顶点间的最短距离来确定平均距离。任何 O(N3)的算法在 N=2500 的情况下都是不行的。注意到图中的边比较少(一个 4 条边)，用 BFS 加上队列是可行的，一次 BFS 的较率为 O(E)，所以 2500 x 10,000 = 2.5 x 106 比 25003 = 1.56 x 1010 要好的多。