Section 6.3: Shortest Path and Minimal Spanning Tree

October 29, 2007

Abstract

Several algorithms are described for tracing the shortest path between two nodes for a simple, positively weighted, connected graph. This is a simpler problem than the traveling salesman problem, and we might hope that a solution algorithm is available.

In addition, algorithms for finding a minimal spanning tree are also described, which are useful for trimming a graph to a subgraph which leaves all nodes connected, but eliminates "unnecessary" or "redundant" connections.

Notice that several of the algorithms we study in this section are actually found in the exercise portion of the section – you'll have to hunt for them there.

1 Shortest Path Algorithms

How might you find the shortest path between two nodes? Some suggestions might include

- Examining all paths (exhaustive search) and choosing the shortest;
- maybe Recursion.

We note that, if there are n nodes, you need at most a path of length $n-1$.

It is relatively easy to come up with algorithms to solve this problem, but of course some ways are better than others. We'll look at several standard algorithms for carrying out this task, and focus on the advantages and disadvantages of each.

1.1 Dijkstra's Algorithm

This algorithm was first described by ¡A HREF="http://www.dcs.ed.ac.uk/home/stg/pub/G/grapl Edsger W. Dijkstra ¡/A¿ [1]. Here is a web-based ¡A HREF="http://www.deakin.edu.au/tildaroonieа example of the workings of the algorithm ¡/A¿ (http://www.deakin.edu.au/tildaroonieagoodman/graph/ex1rd1.htm) (where there is no stated destination node).

- Given two nodes x and y in a simple, connected, positively weighted graph. We seek the shortest path from x to y (assume a non-directed graph).
- Represent the graph by its adjacency matrix (with distances between non-adjacent nodes set to ∞) — meaning that they are non-adjacent.
- A *settled* node is one whose minimal distance from x is known. Initially, the set of settled nodes is only $IN = \{x\}$.
- We grow IN by adding in the next nearest node to x via those already settled. When y falls into IN, we're done.

We keep track of two arrays, indexed by the nodes of graph G:

- an array d indexed by the nodes z, of distances of z to x; and
- an array s indexed by the nodes z, of the node adjacent to a given node z on the shortest path from z to x (so far).

[1] Dijkstra was the one who said that "the quality of programmers is a decreasing function of the density of GO TO statements in the programs they produce." (from a letter to the editor of *Communications of the ACM*, circa 1968)

When y enters IN, we can use s to trace the shortest path.

Let's look at an example:

Example: #4, p. 441/507 We'll need the adjacency matrix: from 4 to 7

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}\begin{bmatrix} \infty & 3 & 5 & \infty & 8 & 1 & \infty & \infty \\ 3 & \infty & 2 & \infty & \infty & \infty & 1 & \infty \\ 5 & 2 & \infty & 1 & \infty & \infty & \infty & 2 \\ \infty & \infty & 1 & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & 4 & \infty & 6 & \infty & 1 \\ 1 & \infty & \infty & \infty & 6 & \infty & 5 & \infty \\ \infty & 1 & \infty & \infty & \infty & 5 & \infty & 1 \\ \infty & \infty & 2 & \infty & 1 & \infty & 1 & \infty \end{bmatrix}$$

$IN = \{4\}$

	1	2	3	4	5	6	7	8
d	∞	∞	1	0	4	∞	∞	∞
s	4	4	4	-	4	4	4	4

$IN = \{4, 3\}$

	6	3	1	0	4	∞	∞	3
s	3	3	4	-	4	4	4	3

$IN = \{4, 3, 8\}$

	6	3	1	0	4	∞	4	3
s	3	3	4	-	4	4	8	3

$IN = \{4, 3, 8, 2\}$

	6	3	1	0	4	∞	4	3
s	3	3	4	-	4	4	8	3

$IN = \{4, 3, 8, 2, 7\}$

$7 \leftarrow 8 \leftarrow 3 \leftarrow 4$

1.2 Other shortest path algorithms

The Bellman-Ford Algorithm (AnotherShortestPath, p. 442/508) operates in a fashion similar to Dijkstra's algorithm, only it finds the shortest distance from x to every other node as described in the book (one could add a termination step, of course).

Each node keeps an eye on its adjacent nodes:

- if no neighbor changes, no change in node; and
- if they change, reevaluate based on the weight of that arc.

Example: #4, p. 443/507 Again, we'll need the adjacency matrix:

$$\begin{bmatrix} 0 & 3 & 5 & \infty & 8 & 1 & \infty & \infty \\ 3 & 0 & 2 & \infty & \infty & \infty & 1 & \infty \\ 5 & 2 & 0 & 1 & \infty & \infty & \infty & 2 \\ \infty & \infty & 1 & 0 & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & 4 & 0 & 6 & \infty & 1 \\ 1 & \infty & \infty & \infty & 6 & 0 & 5 & \infty \\ \infty & 1 & \infty & \infty & \infty & 5 & 0 & 1 \\ \infty & \infty & 2 & \infty & 1 & \infty & 1 & 0 \end{bmatrix}$$

	1	2	3	4	5	6	7	8
d	∞	∞	1	0	4	∞	∞	∞
s	4	4	4	–	4	4	4	4

d	6	3	1	0	4	10	∞	3
s	3	3	4	–	4	5	4	3

d	6	3	1	0	4	7	4	3
s	3	3	4	–	4	1	2	3

etc. until "convergence" – no change in d.

Floyd's algorithm (the algorithm AllPairsShortestPath, p. 444/510) is simpler, and relatively stupid, but has the advantage that it produces the shortest distance between **any** two nodes in the graph (however it does not produce the path itself!). Sometimes this is desired, rather than the distance between any special pair. It too works with the adjacency matrix representation of the graph (modified to contain ∞ off the diagonal).

It simply uses brute force to compare direct paths between a pair and indirect paths between the same pair: we compare

$$A(i,k) + A(k,j)$$

to $A(i,j)$, to see if it's shorter to go from i to j via k. If so, it replaces the element $A(i,j)$ with this shorter distance.

Example: #15, p. 444/510.

2 Minimal Spanning Trees

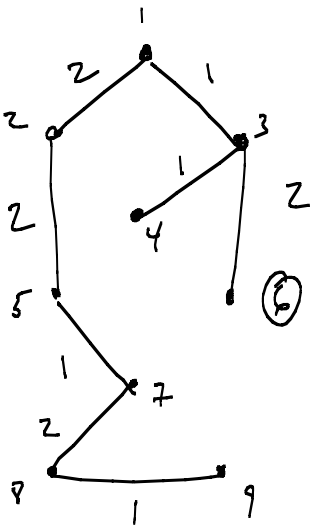
Definition: A **spanning tree** for a connected graph G is a non-rooted tree (basically a connected graph without cycles) containing the nodes of the graph and a subset of the arcs of G. A **minimal** spanning tree is a spanning tree of least weight of a simple, weighted, connected graph G.

Prim's algorithm is a simple one for constructing a minimal spanning tree (these may not be unique!):

- Pick a node at random (they all must figure in the spanning tree, and it doesn't matter where you start). This node is the start of your tree T.

- Follow the arc from the tree T to the nearest adjacent node, and incorporate that arc and node into T (there may be ties - pick one).
- Iterate!

Example: #20, p. 445/511

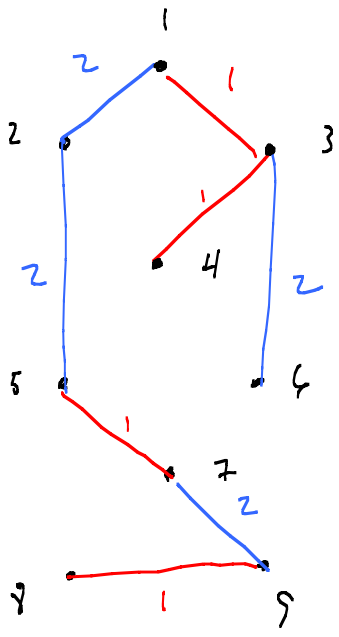


Cost = 12

Kruskal's algorithm is an alternative method for generating a minimal spanning tree. It works by building up a spanning tree from the arcs, ordered from smallest in weight to largest. The only reason to reject a smaller arc over a larger is if it creates a cycle.

Example: #~~23~~ 24, p. 445/511

Weight	
1	1-3, 3-4, 5-7, 8-9
2	1-2, 2-5, 3-6, 7-9, 7-8
3	2-4, 4-6, 6-7,
4	4-5



Cost = 12