**Dijkstra’s Algorithm**

**Algorithm Description**: Dijkstra’s Algorithm is a procedure to find shortest paths from a given vertex s in a graph G to all other vertices. The algorithm incrementally builds a sub-graph of G which is a tree containing shortest paths from s to every other vertex in the tree. A step of the algorithm consists of determining which vertex to add to the tree next.

Note: This is a variant of the approach in the text.

Basic structures needed:

1. The graph G = (V,E) to be analyzed.
2. The tree, actually stored as a map, T. Each time a shortest path to a new vertex is found, an entry is added to T associating that vertex name with a pair indicating the total minimum distance to that vertex and the last edge traversed to get there.
3. A priority queue in which each element is an edge (u, v) to be considered as a path from a located vertex u and a vertex v which we have not yet located. The priority is the total distance from the starting vertex s to v using the known shortest path from s to u plus the length of (u, v).

**Algorithm Pseudocode**:

T is an empty map;

PQ is an empty priority queue;

All vertices in V are marked unvisited;

Add s to T with a total distance of 0 and a null previous edge;

mark s as visited in G;

Add each edge (s,v) of G to PQ with appropriate value

while (T.size() < G.size() and PQ not empty)

do

nextEdge = PQ.remove();

until(one vertex of nextEdge is visited and the other is unvisited)

or until there are no more edges in PQ

// assume nextEdge = (v,u) where v is visited (in T) and u is

unvisited (not in T)

Add u to T; mark u as visited in G;

Add (u,v) to T;

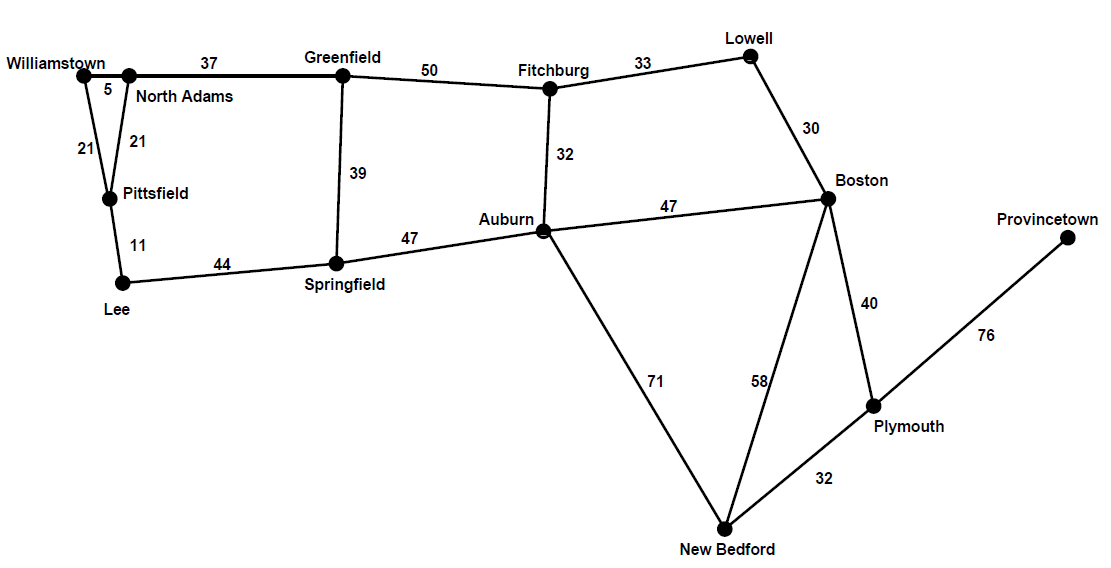
for each unvisited neighbor w of u

add (u,w) to PQ with appropriate weight

When the procedure finishes, T should contain all vertices reachable from s, along with the last edge traversed along the shortest path from s to each such vertex.

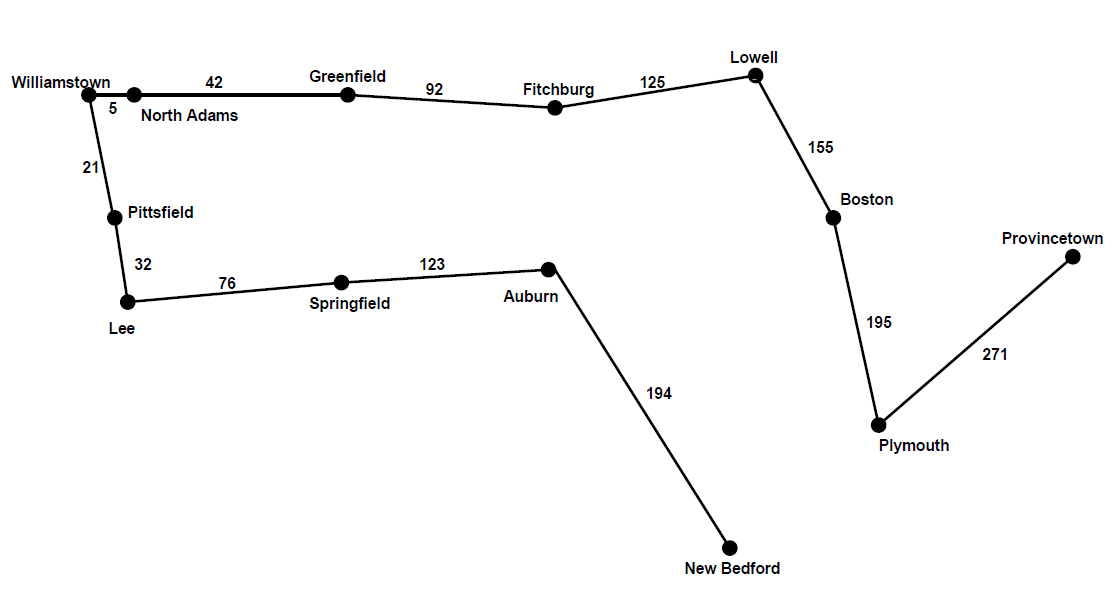
*Disclaimer*: Many details still need to be considered, but this is the essential information needed to implement the algorithm.

Consider the following graph:



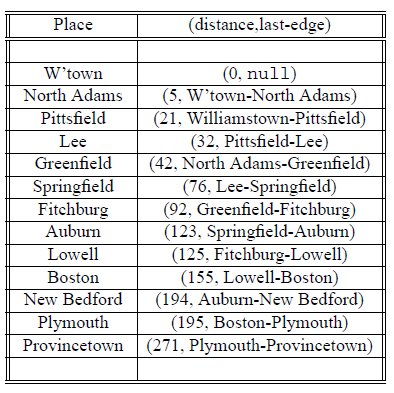
From that graph, the algorithm would construct the following tree for a start node of Williamstown.

Costs on edges indicate total cost from the root.



We obtain this by filling in the following table, a map which has place names as keys and pairs indicating the distance from Williamstown and the last edge traversed on that shortest route as values.

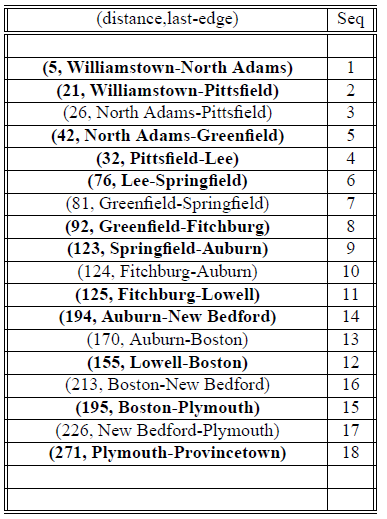
It is easiest to specify edges by the labels of their endpoints rather than the edge label itself.



The table below shows the evolution of the priority queue. To make it easier to see how we arrived at the solution, entries are not erased when removed from the queue, just marked with a number in the “Seq”

column of the table entry to indicate the sequence in which the values were removed from the queue.

Those which indicate the first (and thereby, shortest) paths to a city are shown in bold.



From the table, we can find the shortest path by tracing back from the desired destination until we work our way back to the source.