Cosc-320

Single-Source Minimum Path in Directed Graphs (Dijkstra's Algorithm)

November 30, 2003

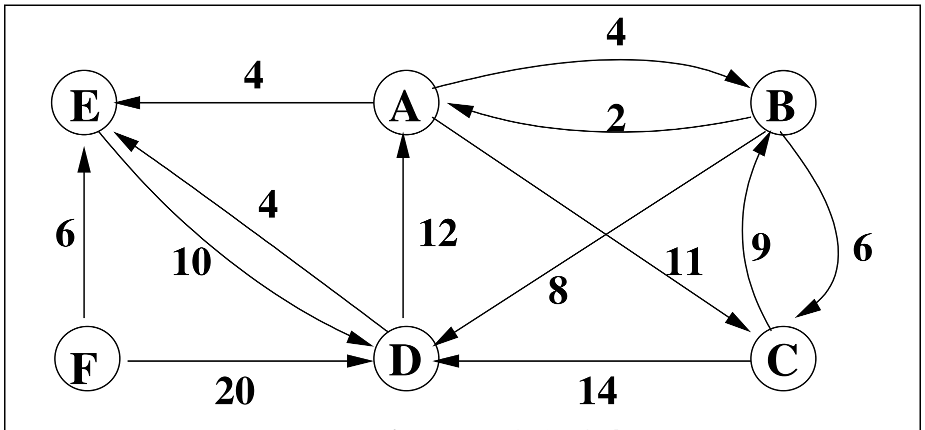
Introduction

These notes on the minimum path (Dijkstra) algorithm expound on Chapter 16 of "Data Structures with C++ using STL 2nd Edition," William Ford and William Topp, Prentice-Hall, 2002, Section 16.6). The treatment here is not as detailed as that in the text, but is the same idea. It is simplified in an attempt to expose the underlying algorithm at the cost of glossing over a number of important implementation details. You should read the text after reading these notes.

◇ **Definition**: The *single-source minimum path* problem is to find the path with minimum total weight from a given starting vertex, V_s, to every other vertex in a weighted directed graph $G(V, E)$.

• Note: The text calls this *minimum-path* leaving off the "single-source" part.

• For an example, we use Figure 16-24 from the text, reproduced here.



Text Figure 16–24

The Algorithm

First, construct a two-dimensional table, T. The rows of T are indexed by the vertices in the graph. T has two columns, one called `DIST` and the other called `VERT`.

$T[V_i]$[DIST] holds the distance (sum of weights of edges) from V_s to V_i.
$T[V_i]$[VERT] holds the vertex from which the algorithm arrived at V_i.

Initially, for each vertex, V_i, $T[V_i]$[DIST] $= \infty$ (except for the starting vertex, $T[V_s]$[DIST] $= 0$) and $T[V_i]$[VERT] is undefined (except $T[V_s]$[VERT] $= V_s$). Construct an empty set of vertices, S and insert V_s into it. Complete the initialization of the table by running the following loop:

```
  for (each vertex Vi in V-S) // i.e., all except Vs
    if (there is an edge from Vs to Vi)
      {
        T[Vi][DIST] = C(Vs,Vi) // weight of edge (Vs,Vi)
        T[Vi][VERT] = Vs
      }
```

After choosing $V_s = A$ in the example graph, the initialized table will be:

Vertex	DIST	VERT
A	0	-
B	4	A
C	11	A
D	∞	-
E	4	A
F	∞	-

and $S = \{A\}$

Finally, run the following loop:

```
while (V-S is not empty) // all vertices less those in S
{
   W = the vertex in (V-S) for which T[W][DIST] is minimal.
   Add W to set S (effectively removing it from V-S)
   for (each vertex v in V-S)
     if (there is an edge from w to v)
       if (T[w][DIST] + C(w,v) < T[v][DIST])
         {
           T[v][DIST] = T[w][DIST] + C(w,v);
           T[v][VERT] = w;
         }
}
```

After running the algorithm on the graph in Figure 16-24 from the text, table T will be:

Vertex	DIST	VERT
A	0	-
B	4	A
C	10	B
D	12	B
E	4	A
F	∞	-

The length of the minimum path from V_s (namely vertex A) to a vertex V_i is found in $T[V_i]$[DIST]. Thus, the minimum path from vertex A to vertex C has a weight of 10. The minimum path from A to E has weight of 4.

The table can also be used to determine the path in each case. To find the minimum path from V_s to V_i, start at $T[V_i]$[VERT] and work toward V_s. To reconstruct the minimum path from A to C, observe that we got to C from B; and to B from A. Thus the minimum path is A-B-C.

Performance

Setting up the table and queue is in $O(|V|)$. Accessing table elements is in $O(1)$ and is done a maximum of $|V|$ times. The loop is done $O(|E|)$ times, but each loop requires finding the minimum weight edge. This is done by keeping the edges in a priority queue, so each access will be in $O(\lg |E|)$. Thus, the Dijkstra algorithm performance is in $O(|V| + |E| \lg |E|)$.