Dijkstra's Algorithm
(same-source shortest path)
Salvatore Castro
Jason Roberts

Introduction

Single-source shortest-path
Applies to weighted-directed graph
$G = (V, E)$
Running time lower than Bellman-Ford
Does not run on negative weights

History

Edsger Wybe Dijkstra
May 11, 1930 – August 6, 2002
Go To Statement Considered Harmful (1968)
"Computer science is no more about computers than astronomy is about telescopes." - E. Dijkstra

Bellman-Ford

Quick Overview
$BF(G,w,s)$ if $G =$ Graph, $w =$ weight, $s =$ source
Determine Single Source($G,s$)
set Distance($s) = 0$;
Predecessor($s$) = nil;
for i <- 1 to $|V(G)| - 1$ do // |V(G)| Number of vertices in the graph
for each edge $(u,v)$ in $G$ do
if Distance($v$) > Distance($u$) + $w(u,v)$ then
set Distance($v$) = Distance($u$) + $w(u,v)$, Predecessor($v$) = $u$;
for each edge $(u,r)$ in $G$ do
if Distance($r$) > Distance($u$) + $w(u,r)$;
return false; //This means that the graph contains a cycle of negative
//weight and the shortest paths are not well defined
return true; //Lengths of shortest paths are in Distance array
Routing, Routing, and Routing

OSPF (Open shortest path first) is a well known real world implementation used in internet routing.

Assume that the function \( w : V \times V \rightarrow [0, \infty] \) describes the cost \( w(x,y) \) of moving from vertex \( x \) to vertex \( y \) (non-negative cost).

We can define the cost to be infinite for pairs of vertices that are not connected by an edge.

The cost of a path between two vertices is the sum of costs of the edges in that path. The cost of an edge can be thought of as (a generalisation of) the distance between those two vertices.

For a given pair of vertices \( s,t \) in \( V \), the algorithm finds the path from \( s \) to \( t \) with lowest cost (i.e. the shortest path).

The algorithm works by constructing a subgraph \( S \) of such that the distance of any vertex \( v' \) (in \( S \)) from \( s \) is known to be a minimum within \( G \).

Initially \( S \) is simply the single vertex \( s \), and the distance of \( s \) from itself is known to be zero.

Edges are added to \( S \) at each stage by (a) identifying all the edges \( e_i = (v_i1,v_i2) \) in \( G-S \) such that \( v_i1 \) is in \( S \) and \( v_i2 \) is in \( G \), and then (b) choosing the edge \( e_j = (v_j1,v_j2) \) in \( G-S \) which gives the minimum distance of its vertex \( v_j2 \) (in \( G \)) from \( s \) from all edges \( e_i \).

The algorithm terminates either when \( S \) becomes a spanning tree of \( G \), or when all the vertices of interest are within \( S \).

1. Make the source node a “Permanent” Node. The source node is the first working node.
2. Examine each non-permanent node adjacent to the working node. If it is not labeled, label it with the distance from the source and the name of the working node. If it is labeled, see if the cost computed using the working node is cheaper than the cost in the label; if so re-label the node as above.
3. Find the non-permanent node with the smallest label, and make it permanent. If this is the destination, then finished. Otherwise, this node is the next working node, continue from step 2.
Example

For this example we will use ‘S’ as the start node.

Example Continued

Final Result

This directed graph illustrates the shortest path from the source node to each node.

Complexity Analysis

First Case: Min-Priority Queue has ordered number of vertices numbered 1 to |V|. Each INSERT and DECREASE-KEY operation is O(1). EXTRACT-Min takes O(V) time. Total: O(V^2+E) -> O(V^2).

Second Case: Min-Priority Queue is implemented as a Fibonacci Heap. The total running time for this will be O(V lg V + E ). EXTRACT-Min will take O(lg V) and DECREASE-Key will take O(1).

Conclusion

Dijkstra’s algorithm has some similarity to both breadth-first search (BFS) and Prim’s Algorithm for computing minimum spanning trees.

BFS: Similar in that the set S corresponds to the set of black vertices in the BFS just as the vertices in S have their final shortest-path weights, so do black vertices in a BFS for their correct vertices.

Prims: Similar in that both algorithms use a min-priority queue to find the "lightest" vertex outside a given set, add this vertex into the set, and adjust the weights of the remaining vertices outside the set accordingly.

Questions?
References


http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html