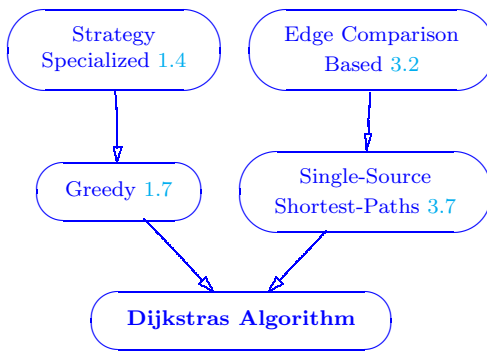
2.6.1. Dijkstra's Algorithm

Section authors: Hamilton Clower, David Scott, Ian Dundore.



Refinement of: Single-Source Shortest-Paths (§3.7), Greedy Algorithm (§1.7).

Dijkstra's Algorithm locates the shortest paths to all vertices in a graph. It is based on a greedy strategy, similar to Prim's algorithm (§3.3) for finding minimum spanning trees, and it is a key component in Johnson's all-pairs shortest-paths algorithm.

Prototype:

```
template <typename VertexListGraph,
 typename P, typename T, typename R>

void
 dijkstra_shortest_paths(VertexListGraph& g,
 typename
  graph_traits<VertexListGraph>
   ::vertex_descriptor s,
 const bgl_named_params<P, T, R>& params);
```

Input:

Dijkstra's algorithm requires a directed, acyclic graph with no negative-weight edges. In terms of the Boost Graph Library, the inputs are a Graph object, a vertex descriptor object which names the source vertex (s), and a parameter list object.

Output:

Dijkstra's algorithm finds the shortest path to all Vertices in a given graph leading from a single source vertex. The length between vertices is defined by the $CompareFunction$ object passed into the parameter list.

Effects:

- After the algorithm has run, the parent of each vertex has been set in the $predecessor\_map$ passed in via the $params$

object. If, for some vertex u, $p[u]$ is equal to u, then u is either the source vertex, or unreachable from the source.

- After the algorithm has run, the distance from the source to each vertex has been set in the $distance\_map$ passed in via the params object. If, for some vertex u, $d[u]$ is zero, then u is the source vertex. If u is unreachable from the source, $d[u]$ will be set to an $infinity$ value. (This is defined as the maximum value for whatever type distances are defined as, e.g. if the type is long, then the infinity value is MAX_LONG).

Complexity:

- This algorithm is $O(E \log V)$ if all vertices are reachable from the named source vertex s.
- In the worst case, where there are some vertices unreachable from the source vertex, which results in a $O((V + E) \log V)$ time bound.
- *Note:* In the Boost Graph Library, optimizations to the algorithm prevent the dijkstra_shortest_paths from examining any vertices not connected to the source vertex. This does not change the asymptotic running time of the algorithm, but it does change the circumstances in which the worst-case running time occurs. With these optimizations, the worst-case times occur when all nodes are reachable from the source. For an illustration of this, see the pair of charts below.

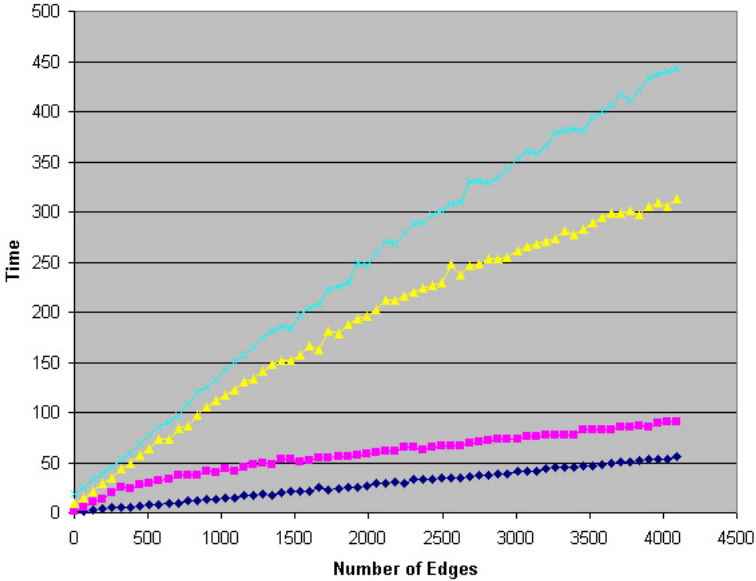
Running Time:

Note: All times are in thousandths of a second.
The data for the following chart was generated completely randomly, with no guarantee as to whether the graphs were connected or not.

	Edges					
Vertices	**4**	**16**	**64**	**256**	**1024**	**4096**
4	0	0	1	4	13	54
16	0	1	2	4	15	55
64	1	0	1	9	21	62
256	2	2	2	1	43	94
1024	5	5	5	5	5	197
4096	20	20	19	19	20	20

The data for the following chart was generated by a different algorithm. This alternate method of generating graphs generated each edge of the graph from a vertex reachable from the source vertex to a random vertex. While this will not ensure that all vertices are reachable from the start vertex, it does ensure that all edges are reachable from the start vertex.

	Edges					
Vertices	**4**	**16**	**64**	**256**	**1024**	**4096**
4	0	0	1	4	14	55
16	0	1	2	5	15	54
64	1	1	4	9	20	62
256	2	2	6	19	43	89
1024	6	5	11	28	95	201
4096	21	21	24	48	135	446



Running Time Chart:

The above chart shows comparative run-times of graphs generated by the all-edges-reachable algorithm with variable numbers of edges and vertices. The horizontal scale shows the number of edges, while each line represents a constant number of vertices. The cyan line represents 4096 vertices, the yellow line represents 256 vertices, the magenta line represents 64 vertices and the black line represents 16 vertices.

Animation:

A short animation of Dijkstra's algorithm in action has been prepared and presented on the Web.

Pseudocode:

From the Boost Graph Library.

(Q is a min-priority queue that supports the **DECREASE-KEY** operation.)

```
DIJKSTRA(G, s, w)
 for each vertex u in V
  d[u] := infinity
   p[u] := u
   color[u] := WHITE
 end for
 color[s] := GRAY
 d[s] := 0
 INSERT(Q, s)
 while (Q != )
  u := EXTRACT-MIN(Q)
  S := S U { u }
  for each vertex v in Adj[u]
   if (w(u,v) + d[u] < d[v])
    d[v] := w(u,v) + d[u]
    p[v] := u
    if (color[v] = WHITE)
```

```
      color[v] := GRAY
     INSERT(Q, v)
     else if (color[v] = GRAY)
      DECREASE-KEY(Q, v)
     end if
    end if
   end for
   color[u] := BLACK
 end while
 return (d, p)
End DIJKSTRA
```