# PATH FINDING - Dijkstra's Algorithm

Ravikiran Jaliparthi Venkat

Indiana State University

rjaliparthive@sycamores.indstate.edu

December 13, 2014

**Abstract**

This document is about Dijkstra's algorithm which is one of the most famous algorithms in computer science. Dijkstra's algorithm is a graph searching algorithm which is used to find shortest path from source node to all the remaining nodes. As far as optimal solutions (paths) are concerned, Algorithms designed for the shortest path problems should be capable of handling three cases. An optimal solution exists. No optimal solution exists because there are no feasible solutions. No optimal solution exists because the length of feasible paths from city 1 to city n is unbounded from below. It has been extended by others to create more advanced path finding algorithms such as A*. This algorithm is often used in routing and as a subroutine in other graph algorithms.

## 1 DESCRIPTION OF THE ALGORITHM

Dijkstra's Algorithm was created in 1959 by Dutch computer scientist Edsger Dijkstra.Dijkstra's Algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and other network related protocols.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

Dijkstra's algorithm works by solving the sub problem k, which computes the shortest path from the source to vertices among the k closest vertices to the source. For the dijkstra's algorithm to work it should be directed- weighted graph and the edges should be non-negative. If the edges are negative then the actual shortest path cannot be obtained. At the kth round, there will be a set called Frontier of k vertices that will consist of the vertices closest to the source and the vertices that lie outside frontier are computed and put into New Frontier. The shortest distance obtained is maintained in sDist[w]. It holds the estimate of the distance from s to w.

Dijkstra‘s algorithm finds the next closest vertex by maintaining the New Frontier vertices in a priority-min queue.
The algorithm works by keeping the shortest distance of vertex v from the source in an array, sDist. The shortest distance of the source to itself is zero. sDist for all other vertices is set to infinity to indicate that those vertices are not yet processed. After the algorithm finishes the processing of the vertices sDist will have the shortest distance of vertex w to s. two sets are maintained Frontier and New Frontier which helps in the processing of the algorithm. Frontier has k vertices which are closest to the source, will have already computed shortest distances to these vertices, for paths restricted upto k vertices. The vertices that resides outside of Frontier is put in a set called New Frontier.

# 2 PSEUDO-CODE OF THE ALGORITHM

The following pseudo-code gives a brief description of the working of the Dijkstra‘s algorithm.
Procedure Dijsktra (V: set of vertices 1... n Vertex 1 is the source
Adj [1 to n] of adjacency lists;
EdgeCost(u, w): edge cost functions;)
Var: sDist[1 to n] of path costs from source (vertex 1); sDist[j] will be equal to the length of the shortest path to j
Begin:
Initialize
Create a virtual set Frontier to store i where sDist[i] is already fully solved Create empty Priority Queue New Frontier; sDist[1]←0; The distance to the source is zero
forall vertices w in V-1 do no edges have been explored yet sDist[w]←$\infty$
end for;
Fill New Frontier with vertices w in V organized by priorities sDist[w];
endInitialize;
repeat
v←DeleteMinNew Frontier; v is the new closest; sDist[v] is already correct forall of the neighbors w in Adj[v] do if sDist[w]¿sDist[v] +EdgeCost(v,w) then sDist[w]←sDist[v] +EdgeCost(v,w)
update w in New Frontier with new priority sDist[w]
endif
endfor
until New Frontier is empty endDijkstra;

# 3 EXAMPLE

## 3.1 Step 1

Using Figure 1-1 we are going to trace the graph from vertex A to all other accessible vertices. We will consider vertex A to be our origin. Before we begin, notice that we are using a directed weighted graph. We can tell this because the edges contain arrows showing the direction of travel they will allow. The graph is weighted because each edge is assigned a non-negative numerical value. Starting at A, we look for vertices that we can reach. Here we can reach B (cost 50), C (cost 30), D (cost 100), and F (cost 10).

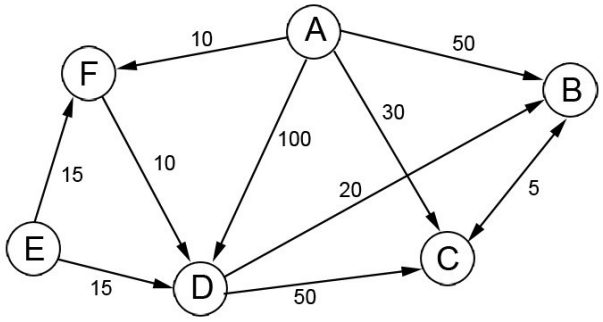
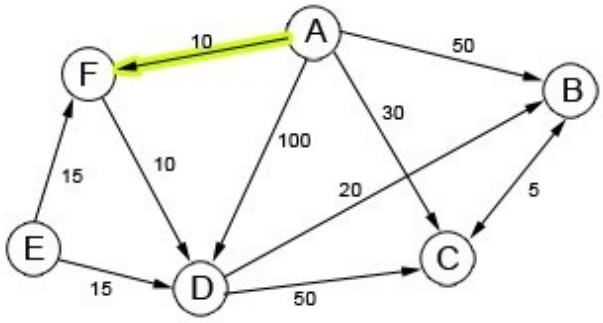
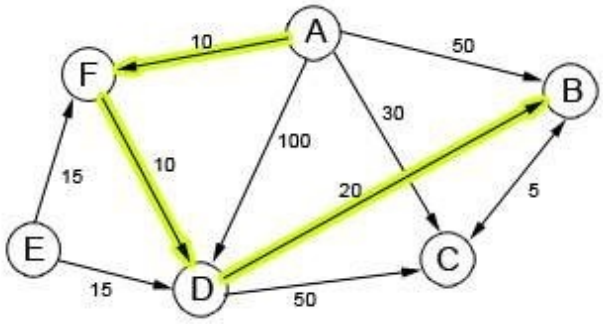


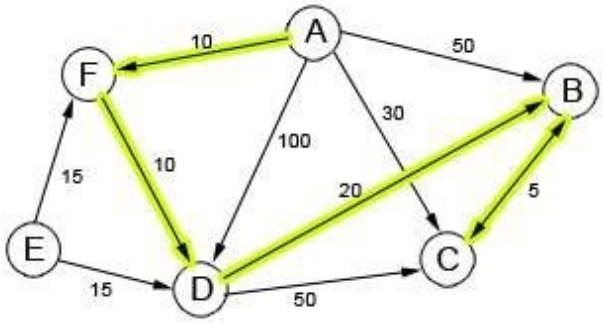
Figure 1-1

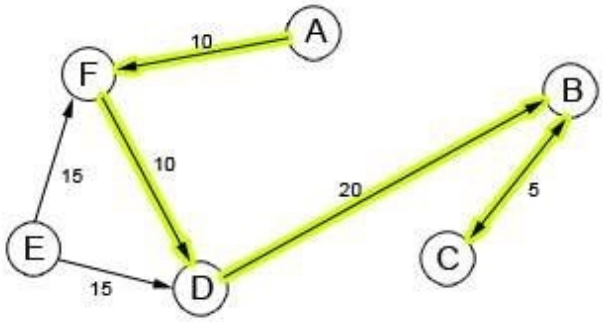



## 3.2 Step 2

In the prior illustration we were able to identify that from vertex A, vertex F was the vertex with the shortest path or least value. So now given A and F, we need to find the next path of least cost, so we evaluate the differences at F and add them to our table on the right. From F we can not get to B, however we can get to B from A as indicated from the previous step. Therefore we bring the 50/A from line 1 of the table, down to line 2. The same event will follow through for vertex C and F and have been highlighted gray for that reason. This leaves us with the only accessible vertex of D at a value of 10/F.Again, E remains at infinity.






### 3.3 Step 3

Now that vertices A,F, and D have been identified as the path with least cost, we now look for the next vertex from D or F,A and B if available as well. From D, we can reach vertices B and C with costs of 20 and 50 respectively. Therefore we will choose B since its cost is lower than C.

### 3.4 Step 4

So far we have identified vertices A, F, D, and B as the shortest path. We now need to account for vertex C. We could have traveled to C from A several steps earlier, but the cost of going to C was far greater than the cost to travel to F at that time. Since A, D, and B are in our generated path, and each of these vertices can go to D, we need to look at each individual edge and the cost assigned. A to C has a cost of 30, D to C has a cost of 50, and B to C has a cost of 5. As a result we will choose edge BC at a cost of 5.

### 3.5 Step 5

Our final graph of Dijkstra*lq*s Algorithm will look like the data shown above. The final traceable path from origin A is A-F-D-B-C with E remaining at infinity since the only edges that originate from E point away at either direction making it impossible to reach E.

## 4 EFFICIENCY

The complexity/efficiency can be expressed in terms of Big-O Notation. Big-O gives another way of talking about the way input affects the algorithms running time. It gives an upper bound

of the running time.

In Dijkstra‘s algorithm, the efficiency varies depending on |V|=n DeleteMins and |E| updates for priority queues that were used.

If a Fibonacci heap was used then the complexity is O( | E | + | V | log | V | ) , which is the best bound. The DeleteMins operation takes O(log|V|).

If a standard binary heap is used then the complexity is O( | E |log |E|),| E | log |E| term comes from|E|updates for the standard heap .

If the set used is a priority queue then the complexity is O(|E|+|V|2). O(V|2) term comes from |V| scans of the unordered set New Frontier to find the vertex with the least sDist value.

# 5 ADVANTAGES OF DIJKSTRA‘S ALGORITHM

• Once it has been carried out you can find the least weight path to all permanently labelled nodes.
• You don’t need a new diagram for each pass.
• Dijkstra’s algorithm has an order of n2 so it is efficient enough to use for relatively large problems.

# 6 DISADVANTAGES OF DIJKSTRA‘S ALGORITHM

• The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.
• Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

# 7 APPLICATIONS

• Traffic information systems use Dijkstra‘s algorithm in order to track the source and destinations from a given particular source and destination
• In the Link-state routing protocol approach, using Dijkstra‘s algorithm each router calculates the shortest path to each network and enters this information into the route table.
• OSPF- Open Shortest Path First, used in Internet routing. It uses a link-state in the individual areas that make up the hierarchy. The computation is based on Dijkstra‘s algorithm which is used to calculate the shortest path tree inside each area of the network.

# 8 RELATED ALGORITHMS

A* algorithm is a graph/tree search algorithm that finds a path from a given initial node to a given goal node It employs a ”heuristic estimate” h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. It follows the approach of best first search.

The Bellman Ford algorithm computes single-source shortest paths in a weighted digraph. It uses the same concept as that of Dijkstra‘s algorithm but can handle negative edges as well. It has a better running time than that of Dijkstra‘s algorithm.

Prims algorithm finds a minimum spanning tree for a connected weighted graph. It implies that it finds a subset of edges that form a tree where the total weight of all the edges in the tree is minimized. it is sometimes called the DJP algorithm or Jarnik algorithm.

# 9 REFERENCES


- http://www.cs.nyu.edu/courses/summer07/G22.2340-001/Presentations/Puthuparampil.pdf
- http://shawnrutter.com/pdf/dijkstra_algo.pdf
- http://math.mit.edu/ rothvoss/18.304.3PM/Presentations/1-Melissa.pdf
- http://www.ijaiem.org/volume2issue7/IJAIEM-2013-07-23-079.pdf
- http://www.academypublisher.com/proc/iscsct10/papers/iscsct10p48.pdf
- http://mat.uab.cat/ alseda/MasterOpt/MyL09.pdf