Understanding Dijkstra's Algorithm

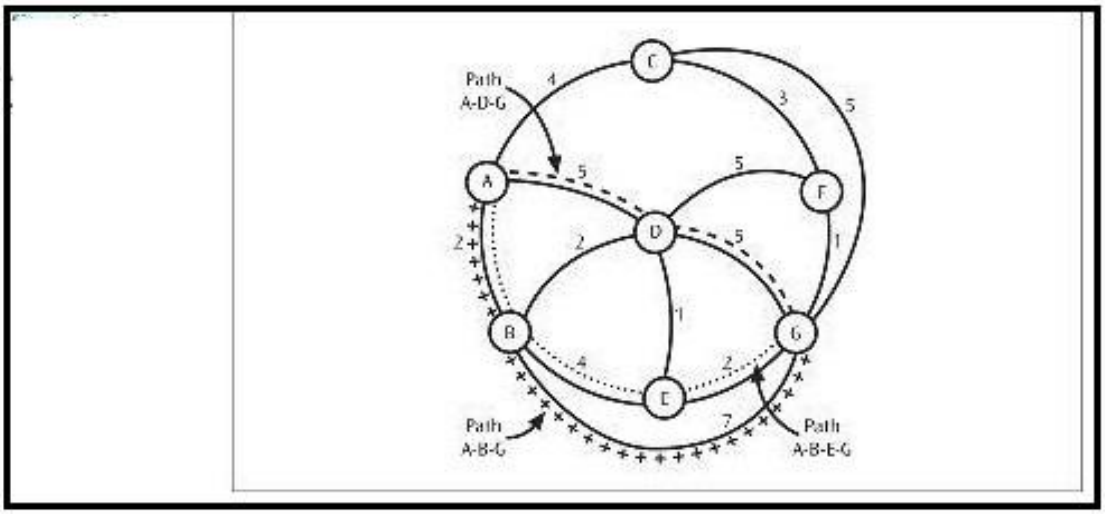
Muhammad Adeel Javaid
Member Vendor Advisory Council, CompTIA

Abstract:

Dijkstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the source) to a destination. It turns out that one can find the shortest paths from a given source to all points in a graph in the same time, hence this problem is sometimes called the single-source shortest paths problem. This paper will help you understand the underlying concepts of Dijkstra Algorithm with the help of simple and easy to understand examples and illustrations.

Introduction:

While people such as network designers and analysts need to have a thorough understanding of Dijkstra's algorithm, a simple close examination is sufficient for the rest of us. Rather than listing the algorithm in stepwise form, let's simply walk through a sample solution. The goal of our example will be to find, in Figure below, the least-cost routes from Node A to each of the other nodes.



To begin, you will first need to create a table, like Table 10-2(a) (below), with a column for each node in the network except the starting node, Node A. The table also needs to include a column called "Visited," in which you will list each node that has been visited. More precisely, when you list a node in this column, it indicates that you have gone to that node and examined all of the node's immediate neighbors. In addition, the table should include a final row called "Next" to denote the next node (but *only* the next node) that the packet should traverse after it leaves Node A. For example, if the Next value under column Node G is B, then a packet that is leaving Node A and is destined for Node G should next be transmitted to Node B. As you work through this example, you should keep in mind that the way this algorithm works is that this table, once it's complete (see Table 10-2(h) at the end of this section), shows *only* the very next hop that should be made from Node A to each of the other nodes. With respect to the example, this means that once you got to Node B (on your way to Node G), you would have to consult a different table—namely, the Dijkstra table for Node B—for the next hop.

Visited	**Node**					
-	B	C	D	E	F	G
Next						

Table 10-2(a) *Initial table for Dijkstra's algorithm*

After you've created the table, select the starting node, Node A, visit it, and add the starting node to the Visited list, as shown in Table 10-2(b). After that, locate each

immediate neighbor (a node only one link or hop away) of Node A that is not yet in the Visited list. Calculate the cost to travel from Node A to each of these neighbors, and enter these values into the table. For example, Node B is one hop away from Node A, it has not yet been visited, and it costs 2 units to travel from A to B. In this case, you should enter 2 in the column for Node B in Table 10-2(b) to indicate the cost of the path from Node A to Node B, and enter B in the Next row to note that to get to B, you go directly to B on the next hop. You can also go from A to C in one hop with a cost of 4 and a Next value of C, and from A to D with a cost of 5 and a Next value of D. These values are also recorded in Table 10-2(b). Note that we have not yet "visited" B, C, or D. We have only visited A, and we are simply examining the costs of the links that run between A and B, A and C, and A and D.

Visited	**Node**					
	B	C	D	E	F	G
A	2	4	5	-	-	-
Next	B	C	D			

Table 10-2(b) *Table for Dijkstra's algorithm after visiting Node A*

No more nodes are immediate neighbors of A, and all of Node A's immediate neighbor links have been examined, so you need to select the next node to visit. According to the algorithm, the next node to visit must be the one that has the least cost in our table thus far. Therefore, you must choose Node B. By specifying that you select the next node with the least cost, the algorithm will find the least cost in all situations. Locate the immediate

neighbors of Node B that have not yet been visited (so far only A has been visited), and determine the cost of traveling from Node A to each immediate neighbor of B via Node B. Note that Node A has been visited, so you should exclude it from being considered at this stage (no sense in going backwards). The immediate neighbors of Node B that have not yet been visited are D, E, and G. The cost of going from Node A to Node D via node B is 4 (the link from A to B costs 2, and the link from B to D costs 2). Since this cost is less than the cost of going directly from A to D (which, as can be seen in Table 10-2(b), is 5), replace the value 5 with the new value 4, to update the table. This update is highlighted in Table 10-2(c). You should also replace the D in the Next row under column D with a B, since the new least-cost path from Node A to Node D now begins with the packet going to Node B first after leaving Node A.

Visited	**Node**					
	B	C	D	E	F	G
A	2	4	5	-	-	-
A B	2	4	4	-	-	-
Next	B	C	B			

Table 10-2(c) *Table for Dijkstra's algorithm after visiting Nodes A and B*

The cost of going from A to E via B is 6 (2 + 4), and the cost of going from A to G via B is 9 (2 + 7). Enter the values 6 and 9 in the E and G columns, respectively, as shown in Table 10-2(d). B is also the Next value for both E and G.

Visited	Node					
	B	C	D	E	F	G
A	2	4	5	-	-	-
A B	2	4	4	6	-	9
Next	B	C	B	B		B

Table 10-2(d) *Table for Dijkstra's algorithm after visiting Nodes A and B, continued*

Let's visit Node C next since, as you can see in Table 10-2(d), it has the next smallest cost. The immediate neighbors of C that have not yet been visited are F and G. The cost of going from A to F via C is 7 (4 + 3). Enter the value 7 in the F column and the value C in the Next row, as shown in Table 10-2(e). The cost of traveling from Node A to G via C is 9 (4 + 5). Since this new value, 9, is *not* less than the current value (also 9) in Table 10-2(d), there is no need to update the table in this case.

Visited	Node					
	B	C	D	E	F	G
A	2	4	5	-	-	-
A B	2	4	4	6	-	9
A B C	2	4	4	6	7	9
Next	B	C	B	B	C	B

Table 10-2(e) *Table for Dijkstra's algorithm after visiting Nodes A, B, and C*

Let's visit Node D next, since it has the next smallest cost. The immediate neighbors of D that have not yet been visited are E, F, and G. The cost of going from A to E via D (via B) is 5 (4 + 1). Since this value is less than the current cost from A to E (less than 6), update the table by entering 5 in the E column (see Table 10-2(f) for reference). We still get to E by first going to B after leaving A, so the value B in the Next row does not change. The cost of going from A to F via D is 10 (5 + 5). The cost of going from A to G via D is also 10. Because the values already entered in the F and G columns are less than 10 (in other words, the table already reflects the least-cost path for those nodes), you do not update the table.

Visited	**Node**					
	B	C	D	E	F	G
A	2	4	5	-	-	-
A B	2	4	4	6	-	9
A B C	2	4	4	6	7	9
A B C D	2	4	4	5	7	9
Next	B	C	B	B	C	B

Table 10-2(f) *Table for Dijkstra's algorithm after visiting Nodes A, B, C, and D*

The next node to visit is E. The immediate neighbor of E that has not yet been visited is G. The cost of traveling from Node A to Node G via Node E (via D via B) is 7 (2 + 2 + 1 + 2). The cost of this path, 7, is smaller than the value already entered in Column G, so you should replace the current value in the table with this new, smaller value, as is shown in Table 10-2(g).

Visited	Node						
	B	C	D	E	F	G	
A	2	4	5	-	-	-	
A B	2	4	4	6	-	9	
A B C	2	4	4	6	7	9	
A B C D	2	4	4	5	7	9	
A B C D E		2	4	4	5	7	7
Next	B	C	B	B	C	B	

Table 10-2(g) *Table for Dijkstra's algorithm after visiting Nodes A, B, C, D, and E*

The next node to visit is F. The only immediate neighbor of F that has not yet been visited is G. The cost of traveling from Node A to Node G via F (via C) is 8. This cost is not less than the current value for F, so do not update the table.

The final node to visit is G. There are, however, no immediate neighbors of G that have not already been visited, so we are finished.

Table 10-2(h) shows the final results. From this table, you can now easily look up the least-cost path from Node A to any other node. If a data packet originates from Node A and is destined for Node *x*, the software in the router will simply consult Column *x* of the table to determine where the data packet should go Next. To find the least-cost route starting from another node, you would need to apply Dijkstra's algorithm again. For example, if you wished to find the least-cost path from, say, Node C to any other node, you would generate a new table by repeating the least-cost algorithm with Node C as the starting position.

Visited	Node						
	B	C	D	E	F	G	
A	2	4	5	-	-	-	
A B	2	4	4	6	-	9	
A B C	2	4	4	6	7	9	
A B C D	2	4	4	5	7	9	
A B C D E		2	4	4	5	7	7
A B C D E F		2	4	4	5	7	7
A B C D E F G	2	4	4	5	7	7	
Next	B	C	B	B	C	B	

Table 10-2(h) *The results of Dijkstra's algorithm applied to a seven-node sub-network starting from Node A*

Dijkstra's Algoritm assigns to every node j a pair of labels (p_j, d_j), where p_j is the node preceding node j in the existing shortest path from 1 to j, d_j is the length of this shortest path. Some of the labels are called temporary, i.e. they could change at a future step; some labels are called permanent, i.e. they are fixed and the shortest path from 1 to a node that is permanently labeled has been found.

We denote by d_{jk} the length of arc (j,k).

Step 1. Label node 1 with the permanent labels $(\emptyset,0)$. Label every node j, such that $(1,j)$ is an arc in the graph, with temporary labels $(1, d_{1j})$. Label all other nodes in the graph with temporary labels $(\emptyset,\infty)$.

Step 2. Let j be a temporarily labeled node with the minimum label d_j, i.e.

d_j=min{d_l: node l is temporarily labeled}.

For every node k, such that (j,k) is in the graph, if $d_k > d_j + d_{jk}$ then relabel k as follows:

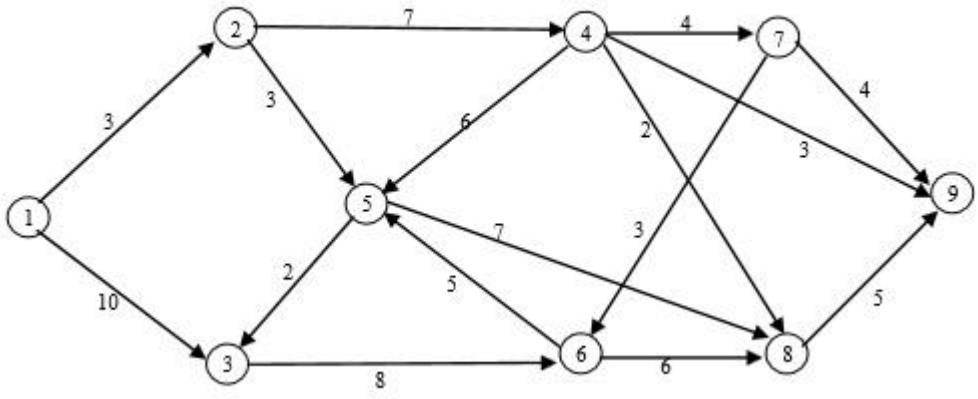
$$p_k = j,\ d_k = d_j + d_{jk.}$$

Consider the labels of node j to be permanent.

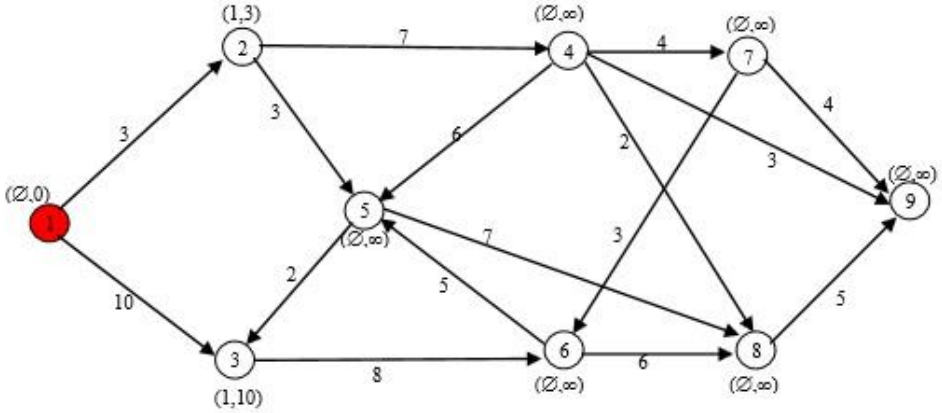
Step 3. Repeat step 2 until all nodes in the graph are permanently labeled.

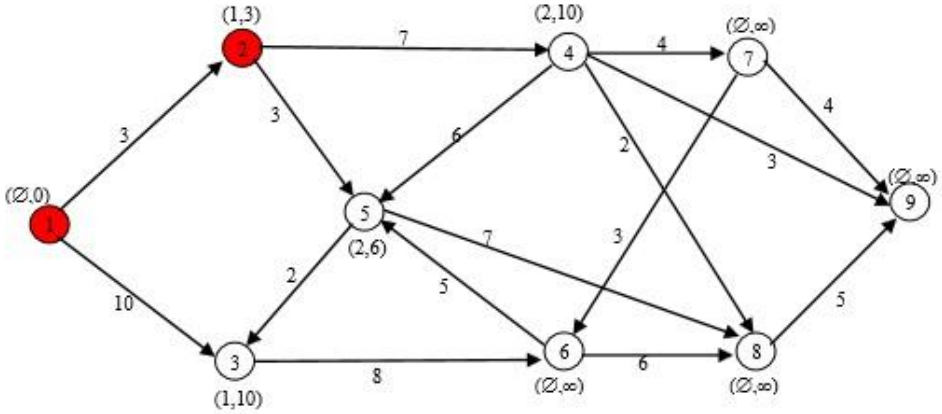
The shortest paths can be found by reading labels p_j.

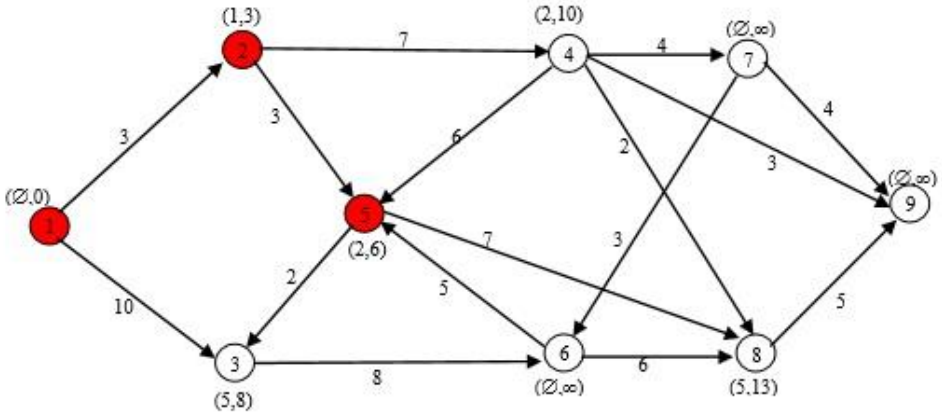
Example. Find the shortest paths from node 1 to all other nodes.

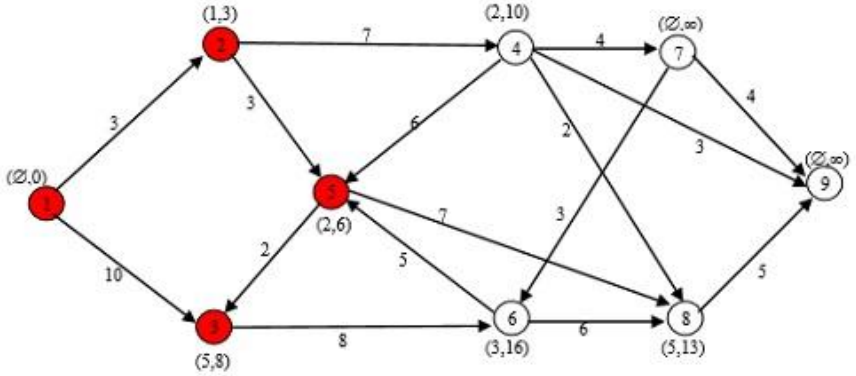


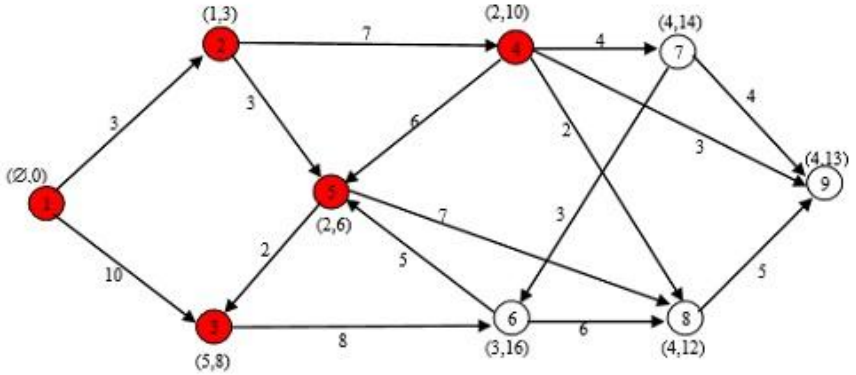
We color the nodes with permanent labels red.

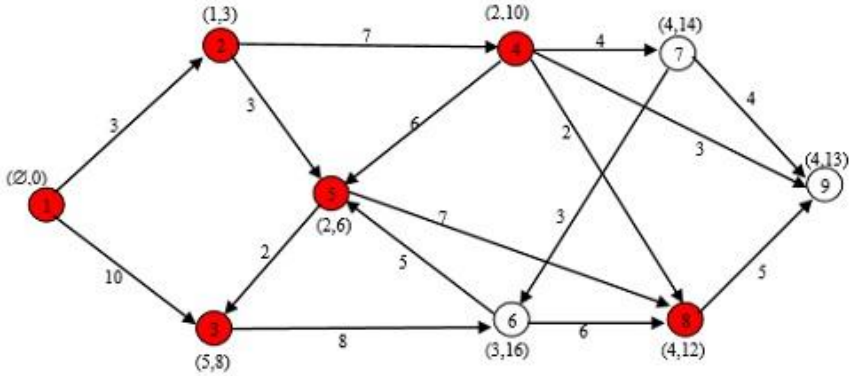

(1,3)
(∅,∞)
(∅,∞)
(∅,∞)
(∅,0)
(∅,∞)
(∅,∞)
(1,10)
(∅,∞)
(∅,∞)
7
4
3
3
6
2
3
4
7
5
3
2
5
10
8
6
5
8
9
1
2
3
4
5
6
7
8
9

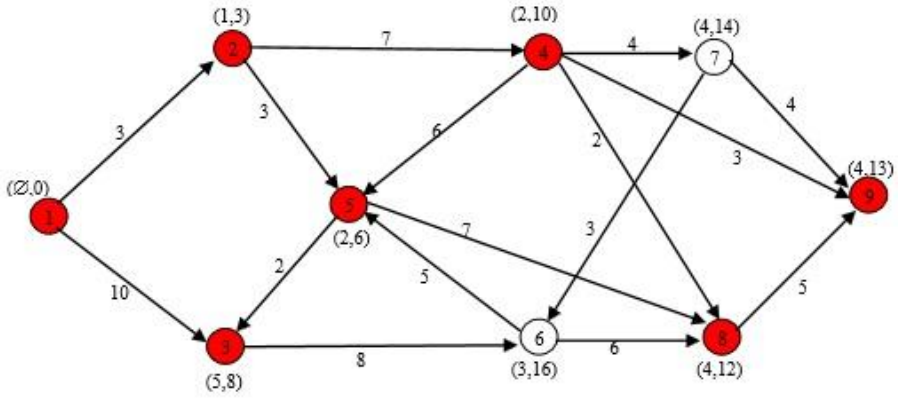

(1,3)
(2,10)
(∅,∞)
(∅,∞)
(2,6)
(∅,∞)
(∅,∞)
(1,10)
(∅,∞)
(∅,∞)
7
4
3
3
6
2
3
4
7
5
3
2
5
10
8
6
5
8
9
1
2
3
4
5
6
7
8
9

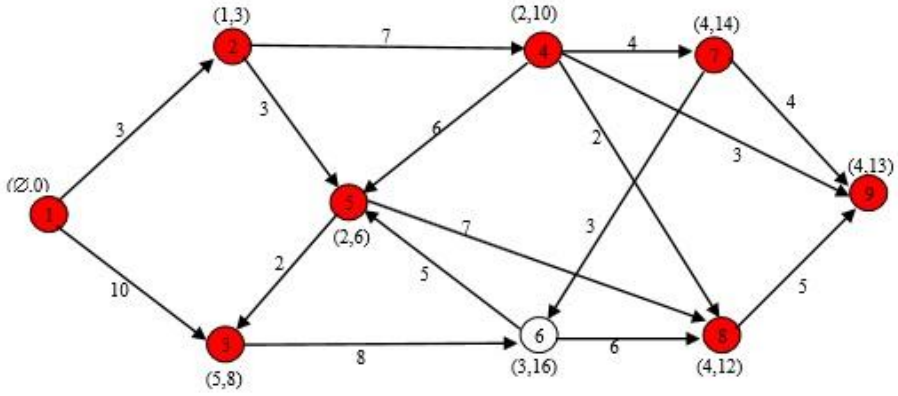

(1,3)
(2,10)
(∅,∞)
(∅,∞)
(2,6)
(∅,∞)
(5,13)
(5,8)
(∅,∞)
7
4
3
3
6
2
3
4
7
5
3
2
5
10
8
6
5
8
9
1
2
3
4
5
6
7
8
9

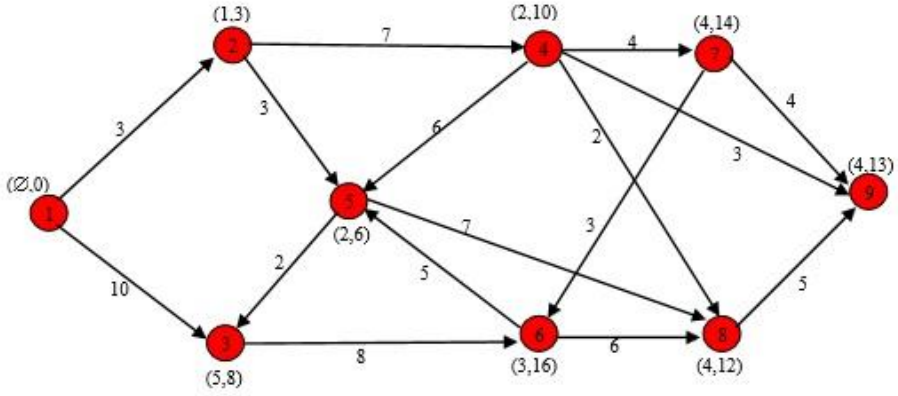

(1,3)
(2,10)
(∅,∞)
(∅,0)
(2,6)
(∅,∞)
(5,8)
(3,16)
(5,13)


(1,3)
(2,10)
(4,14)
(∅,0)
(2,6)
(4,13)
(5,8)
(3,16)
(4,12)

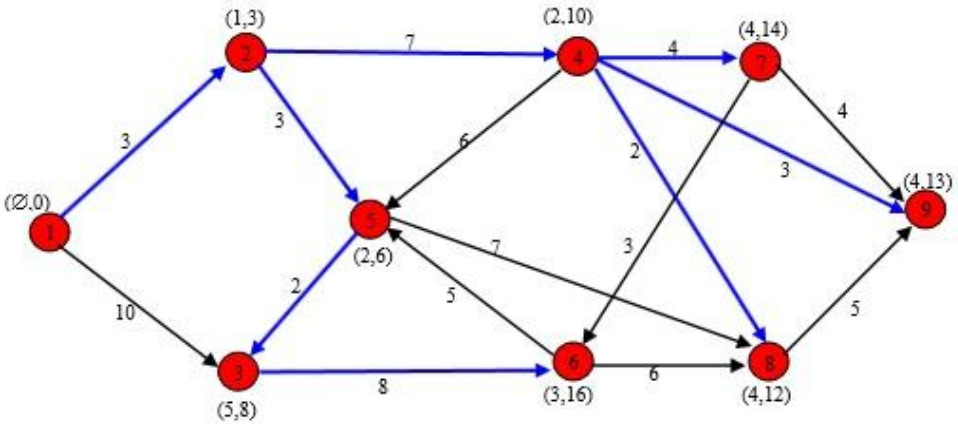

(1,3)
(2.10)
(4,14)
(∅,0)
(2,6)
(4,13)
(5,8)
(3,16)
(4,12)


(1,3)
(2,10)
(4,14)
(∅,0)
(2,6)
(4,13)
(5,8)
(3,16)
(4,12)


(1,3)
(2,10)
(4,14)
(∅,0)
(2,6)
(4,13)
(5,8)
(3,16)
(4,12)

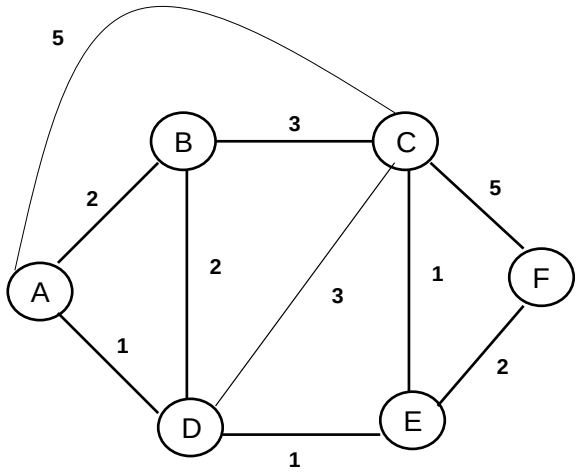

(1,3)
(2,10)
(4,14)
(∅,0)
(2,6)
(4,13)
(5,8)
(3,16)
(4,12)

The Shortest Paths from 1 to all other nodes:



Dijkstra's Algorithm Example

Given: A network topology graph, G, with 6 nodes, and the link cost between nodes as shown below:



Find: Using Dijkstra's Algorithm to find the least-cost paths for nodes A through F

Solution:

Let T = {A, B, C, D, E, F} = number of nodes in the network

s = source node = A

N = set of nodes that have been processed so far

c(i,j) = link cost from node i to node j if two nodes are directly

connected.

= ∞ if there is no link between i and j

D(v) = cost of the shortest path (least-cost path) from node s to node v that is currently known so far. When the algorithm terminates, D(v) is the cost of the shortest path from source s to destination v in the network.

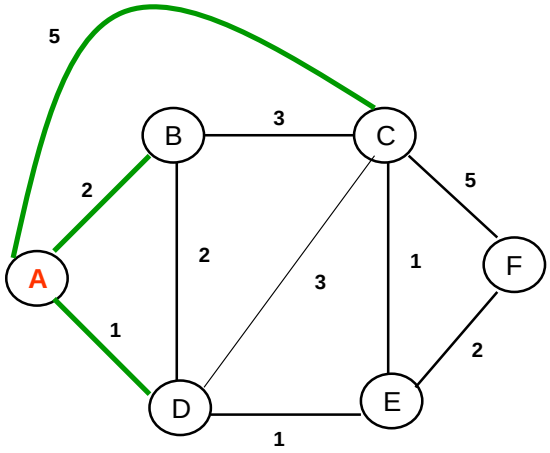
P(v): predecessor node along path from source s to v, that is next v

1) [Initialization]

- N = {**A**} initialize set N contains source node A
- Compute D(v) for v is in set T but not in set N.

  Need to Compute D(v) for v = **B, C, D, E, F**

v	**D(v) = c(s,v) = c(A,v)**	**P(v)**
B	2	A
C	5	A
D	1	A
E	∞	No path
F	∞	No path

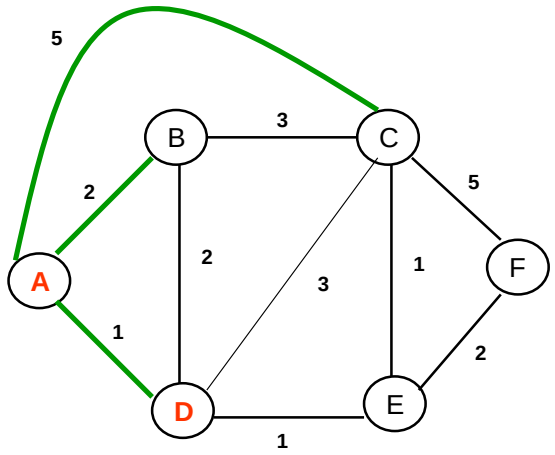


2) [Get Next Node]

- Find w not in N such that D(w) is the minimum. That is the smallest D(w) in the previous step add that node, w, to set N. If they are equal, randomly pick one of them.

  In this case, D(D) = 1 is the smallest, so we add node w = D to set N.

  N = { A, **D** }

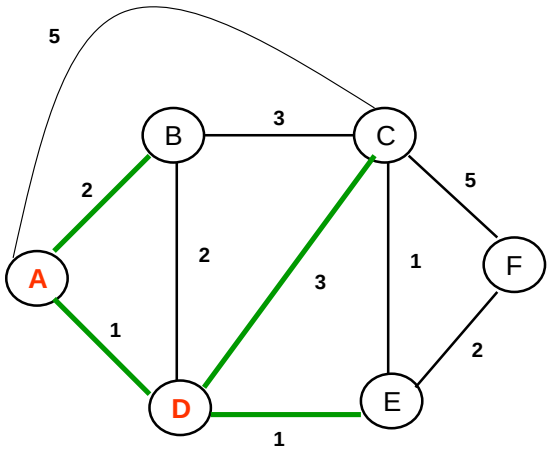


3) [Update Cost for the Shortest Paths]

D(v) = min [D(v), D(w) + c(w,v)] for all nodes v not in T

= min [D(v), D(D) + c(D,v)] for node v = B, C, E, F

v	**D(v) = min [D(v), D(D) + c(D,v)]**	**D(v)**	**P(v)**
B	D(B) = min[D(B), 1 + c(D,B)] = min[2, 1+2]	2	A
C	D(C) = min[D(C), 1 + c(D,C)] = min[5, 1+3]	**4 New**	**D**
E	D(E) = min[D(E), 1 + c(D,E)] = min[∞, 1+1]	**2 New**	**D**
F	D(F) = min[D(F), 1 + c(D,F)] = min[∞, 1+∞]	∞	No path

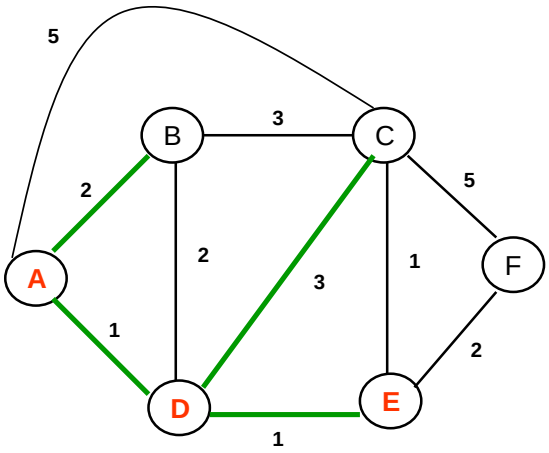


4) [Get Next Node]

- Find w not in N such that D(w) is the minimum. That is the smallest D(w) in the previous step add that node, w, to set N. If they are equal, randomly pick one of them.

  In this case, both D(E) and D(B) have the same smallest cost of 2, we randomly pick one. We pick node w = E to set N.

  N = { A, D, **E** }

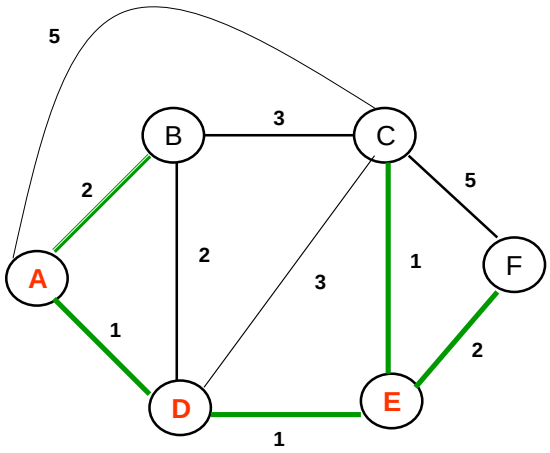


5) [Update Cost for the Shortest Paths]

D(v) = min [D(v), D(w) + c(w,v)] for all nodes v not in N

= min [D(v), D(E) + c(E,v)] for node v = **B, C, F**

v	**D(v) = min [D(v), D(E) + c(E,v)]**	**D(v)**	**P(v)**
B	D(B) = min[D(B), 2 + c(E,B)] = min[2, 2+∞]	2	A
C	D(C) = min[D(C), 2 + c(E,C)] = min[4, 2+1]	**3 New**	**E**
F	D(F) = min[D(F), 2 + c(E,F)] = min[∞, 2+2]	**4 New**	**E**

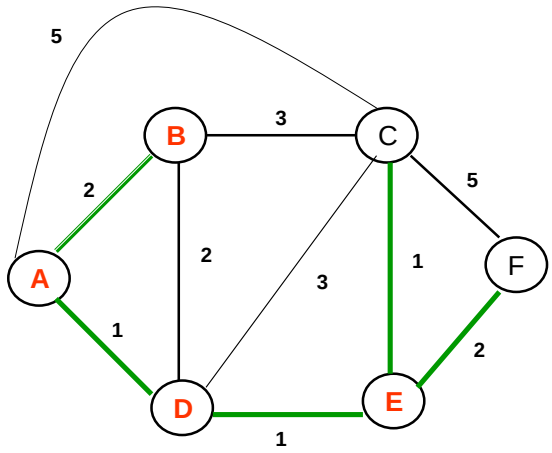


6) [Get Next Node]

- Find the smallest D(v) in the previous step add that node to N. If they are equal, randomly pick one of them.

  D(B) has the lowest cost = 2. We pick node w = B to set N.

  N = { A, D, E, **B** }

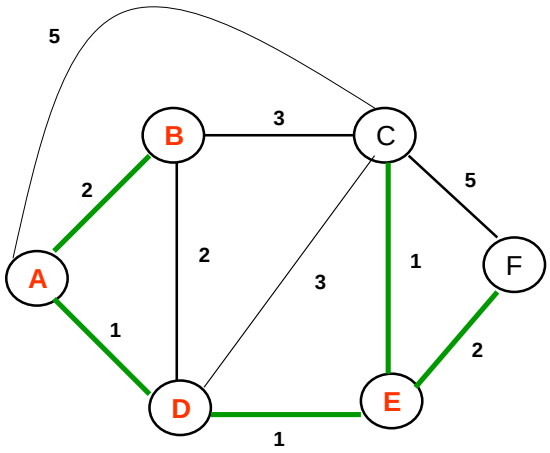


7) [Update Cost for the Shortest Paths]

D(v) = min [D(v), D(w) + c(w,v)] for all nodes v not in N

= min [D(v), D(B) + c(B,v)] for node v = **C, F**

v	**D(v) = min [D(v), D(B) + c(B,v)]**	**D(v)**	**P(v)**
C	D(C) = min[D(C), 2 + c(B,C)] = min[3, 2+3]	3	E
F	D(F) = min[D(F), 2 + c(B,F)] = min[4, 2+∞]	4	E

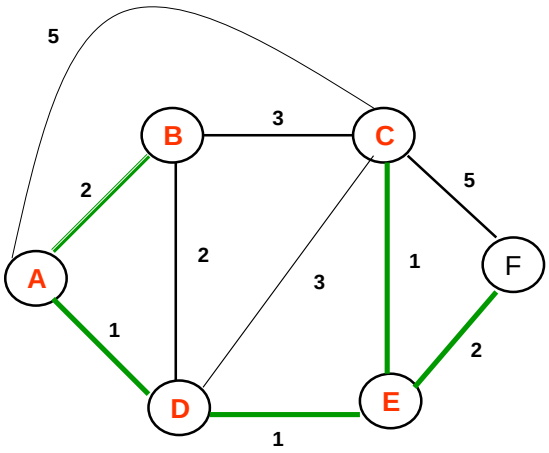


8) [Get Next Node]

- Find the smallest D(v) in the previous step add that node to N. If they are equal, pick any one of them.

  D(C) is the smallest, so we add node w = C to set N.

  N = { A, D, E, B, **C**}

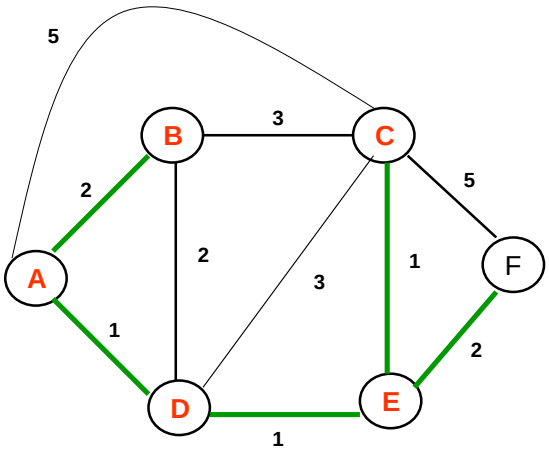


9) Update Cost for the Shortest Paths]

D(v) = min [D(v), D(w) + c(w,v)] for all node n not in N

= min [D(v), D(C) + c(C,v)] for node **n = F**

v	**D(v) = min [D(v), D(C) + c(C,v)]**	**D(v)**	**P(v)**
F	D(F) = min[D(F), 3 + c(C,F)] = min[4, 3+5]	4	E



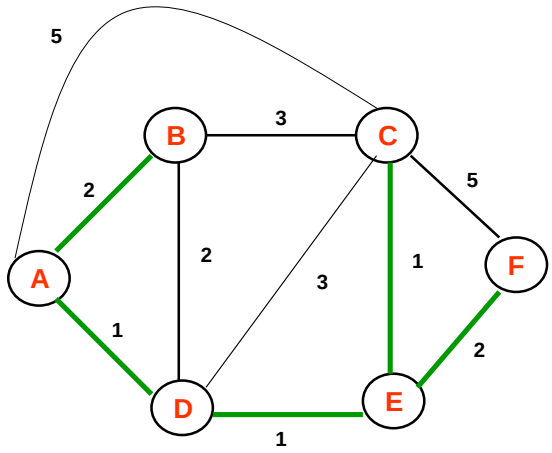
10)

[Get Next Node]

- Find the smallest D(v) in the previous step add that node to N. If they are equal, pick any one of them.

  D(F) is the only one, so we add node w = F to set N.

  N = { A, D, E, B, C, **F** }

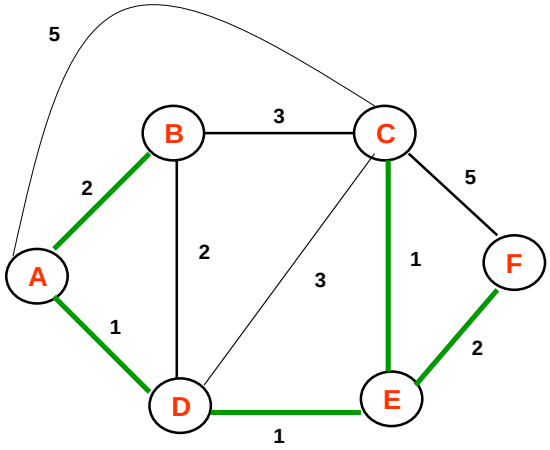


N = T, so the algorithm terminate.

11) Now, we have to write down the shortest path from node A to all other nodes. Start from backwards since those are the least cost for the shortest path, and then work towards source node A. The column D(v) is the cost for the shortest path from source A to v.

v	D(v)	Path
F	**4**	**A-D-E-F**
D	**1**	**A-D**
C	**3**	**A-D-E-C**

E	**2**	**A-D-E**
B	**2**	**A-B**



References:

- Ahuja, R.K., Magnanti, T.L. and Orlin, J.B. (1993), Network Flow Theory, Algorithms, and Applications, Prentice-Hall, Englewood-Cliffs, NJ.
- Bellman, R. (1957), Dynamic Programming, Princeton University Press, Princeton, NJ.
- Brassard, G. and Bratley, P. (1988) Algorithmics,Prentice-Hall, Englewood Cliffs, NJ.
- Daellenbach, H.G., George, J.A. and D.C. McNickle, (1983), Introduction to Operations Research Techniques, 2nd Edition, Allyn and Bacon, Boston.
- Dantzig, G.B., (1963), Linear Programming and Extensions, Princeton University Press, Princeton, NJ.
- Dantzig, G.B. and N.M Thapa, (2003), Linear Programming 2: Theory and Extensions, Springer Verlag, Berlin.
- Denardo, E.V. (2003), Dynamic Programming, Dover, Mineola, NY.
- Dijkstra, E.W. (1959), A note on Two Problems in Connexion with Graphs, Numerische mathematik, 1, 269-271.
- Dreyfus, S. (1969), An appraisal of some shortest-path algorithms Operations Research, 17, 395-412.
- Evans, J.R. and Minieka, E. (1992), Optimization Algorithms for Networks and Graphs, Marcel Dekker, NY.
- Gass, S.I. and Harris, C.M. (1996), Encyclopedia of Operations Research and management Science, Kluwer, Boston, Mass.
- Hillier, F.S. and Lieberman, G.J. (1990) Introduction to Operations Research, 5th Edition, Holden -Day, Oakland, CA.
- Lawler, E.L. (1976), Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Whinston, NY.
- Markland, R.E. and J.R. Sweigart, (1987), Quantitative Methods: Applications to Managerial Decision Making, John Wiley, NY.
- Microsoft Shortest Path Algorithms Project: research.microsoft.com/research/sv/SPA/ex.html.
- Moore, E.F. (1959), The shortest path through a maze, pp. 285-292 in Proceedings of an International Synposium on the Theory of Switching (Cambridge, Massachusetts, 2-5 April, 1957), Harvard University Press, Cambridge.