Shortest Path Algorithm – Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Algorithm

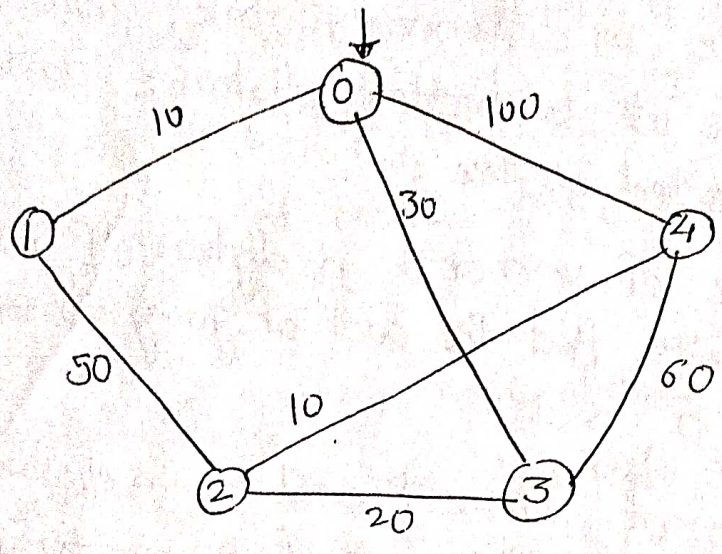
Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
3. For the current node, consider all of its neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3

Shortest Path Algorithm

1) Dijkstra's Algorithm

Show the working of the Dijkstra's Algorithm on the graph given below. (Source vertex is taken as 0).



Solution

Step 1 :- Cost Matrix

	0	1	2	3	4
0	∞	10	∞	30	100
1	10	∞	50	∞	∞
2	∞	50	∞	20	10
3	30	∞	20	∞	60
4	100	∞	10	60	∞

distance matrix

0	1	2	3	4
0	∞	∞	∞	∞

visited

0	1	2	3	4
0	0	0	0	0

Iteration 1

a) Select vertex 0 (at minimum distance).

b) Mark visited[0] as 1

c) Re-adjust distances of vertices not marked as 1 in visited[].

- Distance in distance matrix should be altered if there is a better distance path through the selected vertex 0.
- Distance of vertex 1 in distance[] matrix is infinity. But the cost of going to vertex 1 from vertex 0 is 10.
- Distance of vertex 3 and 4 in distance[] matrix are ∞. But the cost of going to ~~vertex~~ vertices 3 and 4 from vertex 0 is 30 and 100 respectively.
- Distance ~~fo~~ of vertices 1, 3 and 4 should be modified.

	0	1	2	3	4
distance	0	10	∞	30	100

	0	1	2	3	4
visited	1	0	0	0	0

Iteration 2

- Select vertex 1 (at minimum distance)
- Mark visited[1] as 1.
- Re-adjust distances.
- Cost of going to vertex 2 from vertex 0, via the selected vertex 1 is given by

  distance[1] + cost[1][2] = 10 + 50 = 60

  Distance of 60 is better than the existing distance of ∞

- Cost of going to vertex 3 from the source vertex 0, via the selected vertex 1 is

  distance[1] + cost[1][2] = 10 + ∞ = ∞

  which is worse than the existing distance of 30.

- Similarly cost of going to vertex 4 from the source, via selected vertex is 1 is ∞. which is worse than existing distance of 100.

distance

0	1	2	3	4
0	10	60	30	100

visited

0	1	2	3	4
1	1	0	0	0

Iteration 3 :-

Vertex selected = 3

Cost of going to vertex 2 via vertex 3

= distance[3] + cost[3][2]

= 30 + 20

= 50

Cost of going to vertex 4 via vertex 3.

= distance[3] + cost[3][4]

= 30 + 60

= 90

Distances ~~between~~ of vertices 2 & 4 should be changed.

	0	1	2	3	4
distance	0	10	50	30	90

	0	1	2	3	4
visited	1	1	0	1	0

Iteration 4 :-

vertex selected = 2

Cost of going to vertex 2

= distance[2] + cost[2][4]

= 50 + 10

= 60

Distance of vertex 4 should be changed

	0	1	2	3	4
distance	0	10	50	30	60

← Final distance

	0	1	2	3	4
visited	1	1	1	1	0

Step 2

+ Vertex selected = 0
+ Mark visited[0] as 1
+ Re-adjust distances of vertices not marked as 1 in visited[]:

0	1	2	3	4
1	0	0	0	0

$d_0 = 0$

$d_1 = 10$ ←

$d_2 = \infty$

$d_3 = 30$

$d_4 = 100$

Selected the minimum distance i.e. $d1 = 10$

So vertex 1 is selected.

Step 3

* Vertex selected = 1

Readjust distances of vertices

0	1	2	3	4
1	1	0	0	0

$d_0 = 0$

$d_1 = 10$

new $d_2 = \min(d_2, d_1 + cost[1][2])$

$= \min(\infty, 10 + 50)$

$= \min(\infty, 60)$

$= 60$

new $d_3 = \min(d_3, d_1 + cost[1][3])$

$= \min(30, 10 + \infty)$

$= \min(30, \infty)$

$= 30$

new $d_4 = \min(d_4, d_1 + cost[1][4])$

$= \min(100, 10 + \infty)$

$= \min(100, \infty) = 100$

$d_0 = 0$

$d_1 = 10$

new $d_2 = 60$

new $d_3 = 30$ ←

new $d_4 = 100$

Step 4

0	1	2	3
1	1	0	1

vertex selected = 3

Readjust distances of vertices

$d_0 = 0$

$d_1 = 10$

new $d_2 = \min(d_2, d_3 + cost[3][2])$

$= \min(60, 30+20) = \min(60, 50) = 50$

$d_3 = 30$

new $d_4 = \min(d_4, d_3 + cost[3][4])$

$= \min(100, 30+60)$

$= \min(100, 90)$

$= 90$

$d_0 = 0$

$d_1 = 10$

new $d_2 = 50$ ←

$d_3 = 30$

new $d_4 = 90$

As $d_2 = 50$, vertex 2 is selected.

Step 5

Vertex selected = 2

Readjust distances of vertices

$d_0 = 0$

$d_1 = 10$

$d_2 = 50$

$d_3 = 30$

new $d_4 = \min(d_4, d_2 + cost[2][4])$

$= \min(90, 50 + 10)$

$= \min(90, 60)$

$= 60$

$d_0 = 0$

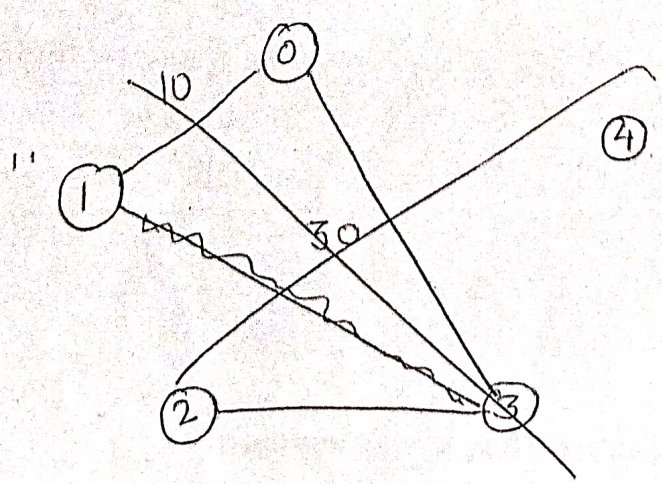
$d_1 = 10$

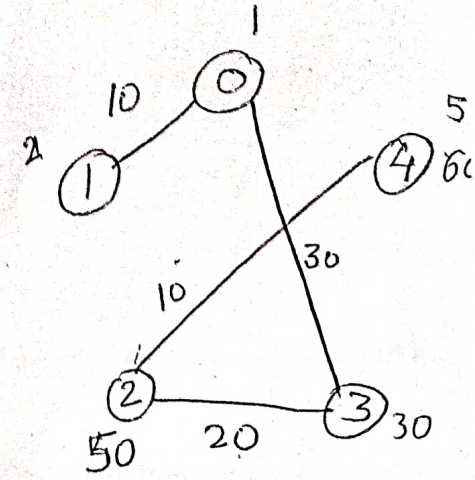
$d_2 = 50$

$d_3 = 30$

new $d_4 = 60$

Finally, shortest path for given graph is





Total cost of shortest path from 0

Example