UNIVERSITY INSTITUTE OF COMPUTING

MASTER OF COMPUTER APPLICATIONS

DESIGN AND ANALYSIS OF ALGORITHMS

24CAT-611

UNIT-2

DISCOVER . LEARN . EMPOWER

DESIGN AND ANALYSIS OF ALGORITHMS

Course Outcome

CO Number	Title	Level
CO3	Apply and analyze important algorithmic design paradigms and their applications	Understand
CO4	Implement the major graph algorithms to model engineering problems	Understand

- **Divide and Conquer:** General method, Binary search, Advantages and disadvantages of divide and conquer, Decrease and conquer approach: Topological sort

Topics to be covered

- Dijkstra’s Algorithm
- Huffman Trees and Codes
- Heaps and Heap Sort

Introduction

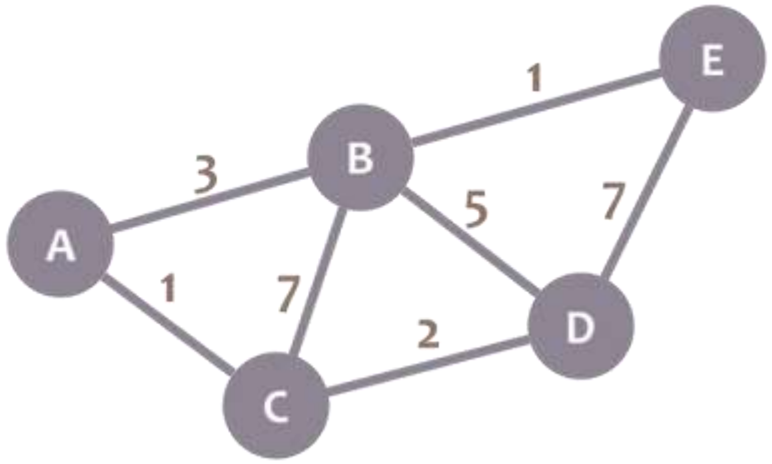
- Dijkstra Algorithm is a very famous greedy algorithm.
- It is used for solving the single source shortest path problem.
- It computes the shortest path from one particular source node to all other remaining nodes of the graph.

Conditions

- It is important to note the following points regarding Dijkstra Algorithm-
- Dijkstra algorithm works only for connected graphs.
- Dijkstra algorithm works only for those graphs that do not contain any negative weight edge.
- The actual Dijkstra algorithm does not output the shortest paths.
- It only provides the value or cost of the shortest paths.
- By making minor modifications in the actual algorithm, the shortest paths can be easily obtained.
- Dijkstra algorithm works for directed as well as undirected graphs.

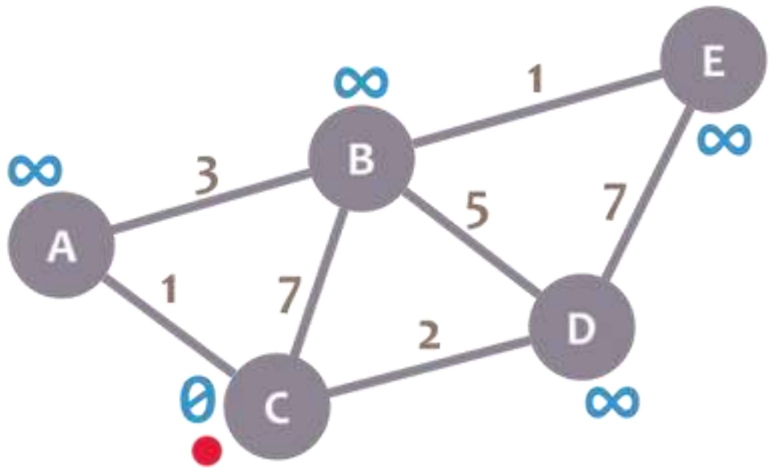
Dijkstra's Algorithm

- Dijkstra's Algorithm allows you to calculate the shortest path between one node (you pick which one) and *every other node in the graph*. You'll find a description of the algorithm at the end of this page, but, let's study the algorithm with an explained example! Let's calculate the shortest path between node C and the other nodes in our graph:



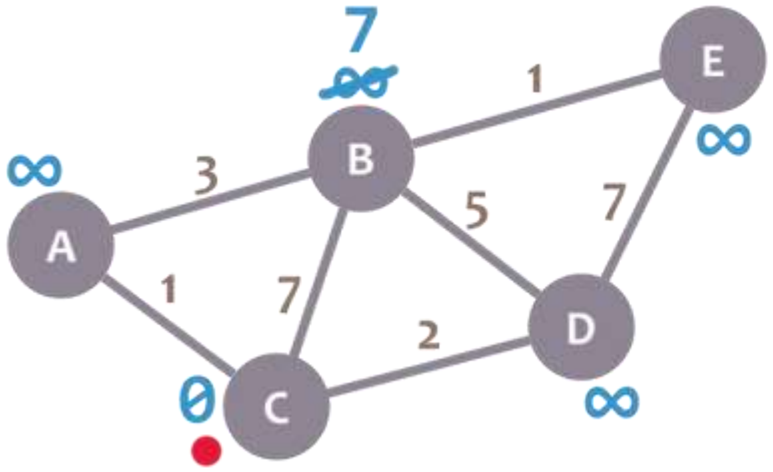
Steps

- During the algorithm execution, we'll mark every node with its *minimum distance* to node C (our selected node). For node C, this distance is 0. For the rest of nodes, as we still don't know that minimum distance, it starts being infinity (∞):



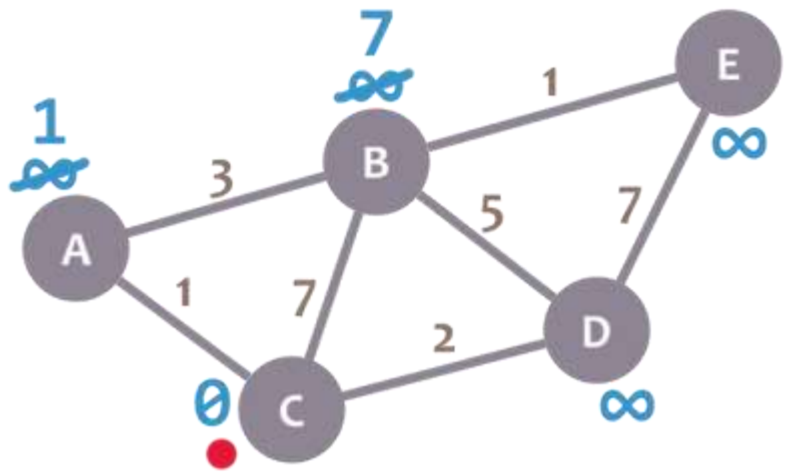
Next Step

- We'll also have a *current node*. Initially, we set it to C (our selected node). In the image, we mark the current node with a red dot.
- Now, we check the neighbours of our current node (A, B and D) in no specific order. Let's begin with B. We add the minimum distance of the current node (in this case, 0) with the weight of the edge that connects our current node with B (in this case, 7), and we obtain 0 + 7 = 7. We compare that value with the minimum distance of B (infinity); the lowest value is the one that remains as the minimum distance of B (in this case, 7 is less than infinity):



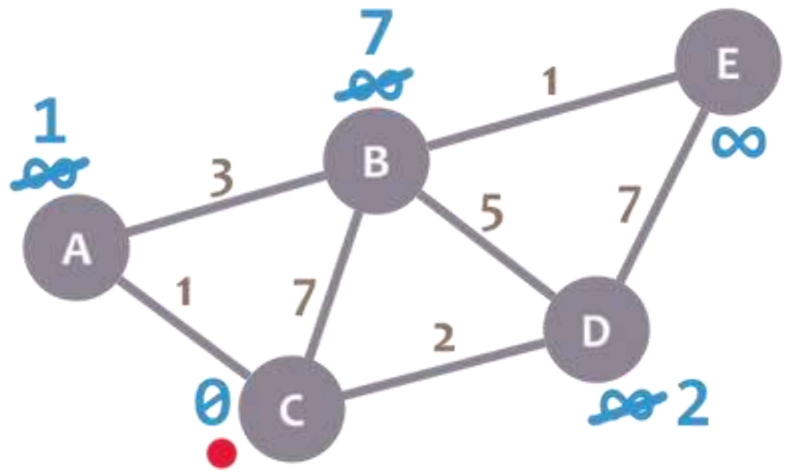
Next Step

- So far, so good. Now, let's check neighbour A. We add 0 (the minimum distance of C, our current node) with 1 (the weight of the edge connecting our current node with A) to obtain 1. We compare that 1 with the minimum distance of A (infinity), and leave the smallest value:



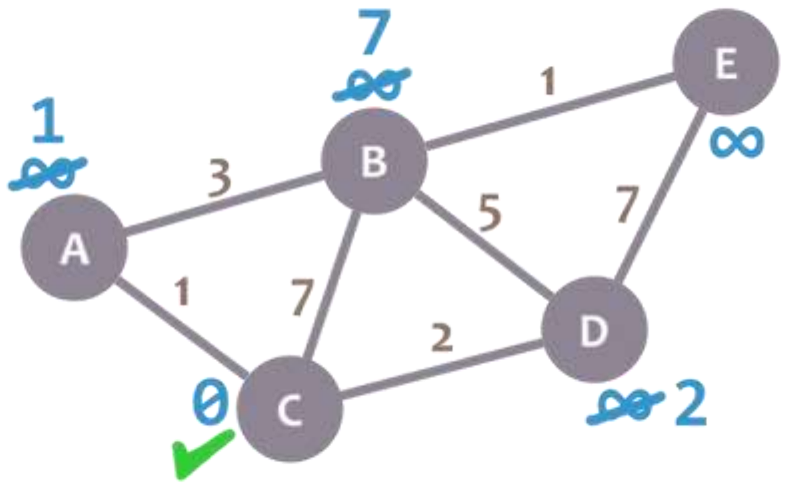
Next Step

OK. Repeat the same procedure for D:



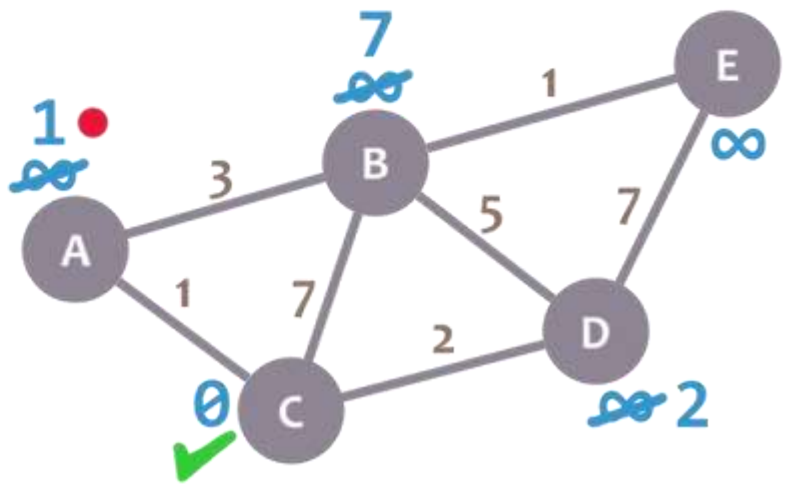
Next Step

- Great. We have checked all the neighbours of C. Because of that, we mark it as *visited*. Let's represent visited nodes with a green check mark:



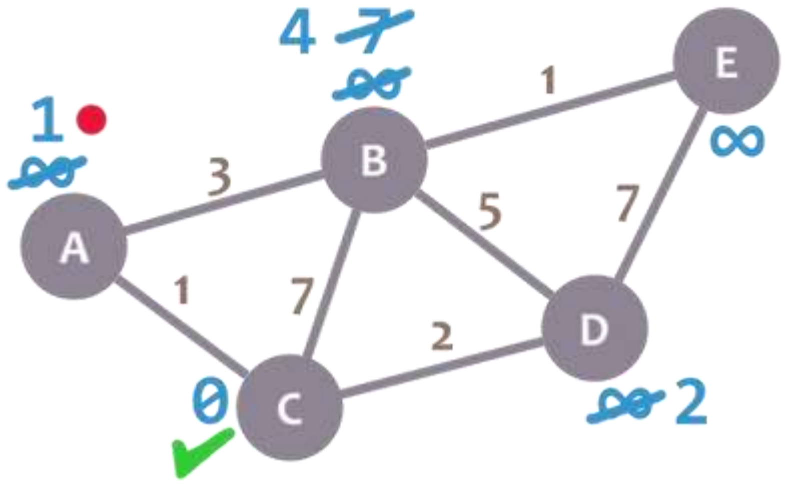
Next Step

- We now need to pick a new *current node*. That node must be the unvisited node with the smallest minimum distance (so, the node with the smallest number and no check mark). That's A. Let's mark it with the red dot:



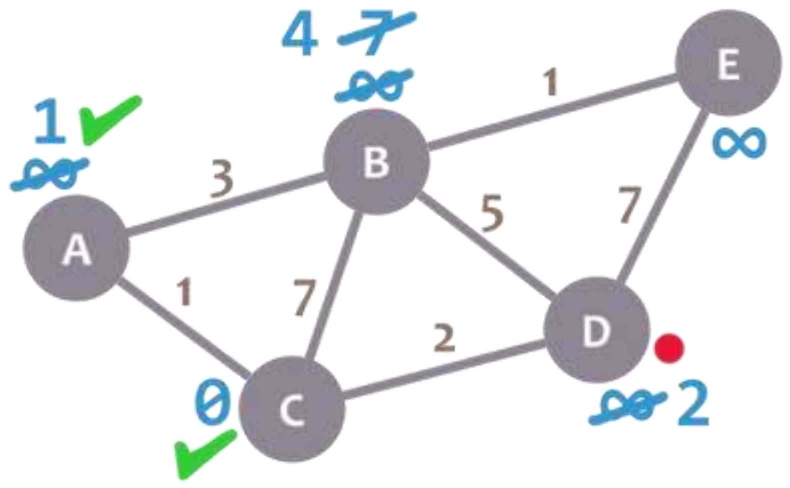
Next Step

- And now we repeat the algorithm. We check the neighbours of our current node, ignoring the visited nodes. This means we only check B.
- For B, we add 1 (the minimum distance of A, our current node) with 3 (the weight of the edge connecting A and B) to obtain 4. We compare that 4 with the minimum distance of B (7) and leave the smallest value: 4.



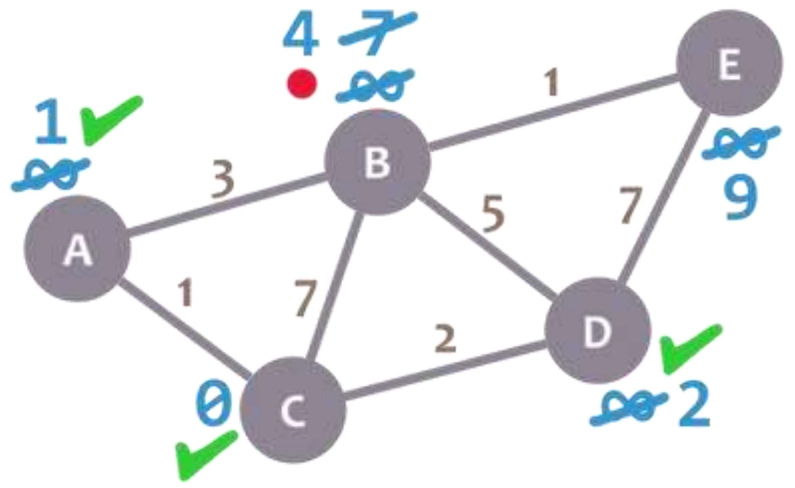
Next Step

- Afterwards, we mark A as visited and pick a new current node: D, which is the non-visited node with the smallest current distance.



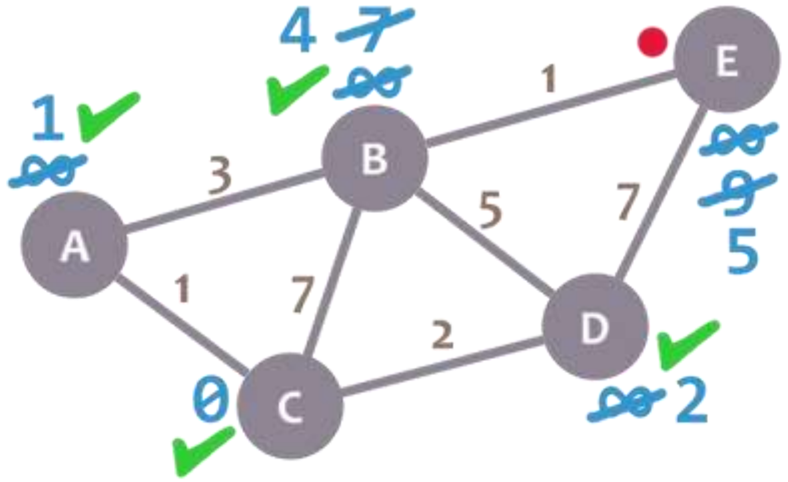
Next Step

- We repeat the algorithm again. This time, we check B and E.
- For B, we obtain 2 + 5 = 7. We compare that value with B's minimum distance (4) and leave the smallest value (4). For E, we obtain 2 + 7 = 9, compare it with the minimum distance of E (infinity) and leave the smallest one (9).
- We mark D as visited and set our current node to B.



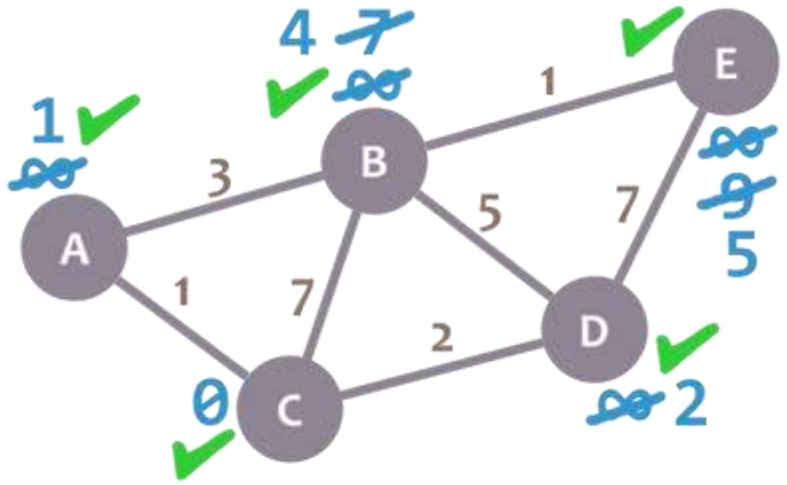
Next Step

- Almost there. We only need to check E. 4 + 1 = 5, which is less than E's minimum distance (9), so we leave the 5. Then, we mark B as visited and set E as the current node.



Next Step

- E doesn't have any non-visited neighbours, so we don't need to check anything. We mark it as visited.



Next Step

- As there are not unvisited nodes, we're done! The minimum distance of each node now actually represents the minimum distance from that node to node C (the node we picked as our initial node)!

References

- https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/dijkstras-algorithm

Books:

1. Introduction to Algorithms by Coreman, Leiserson, Rivest, Stein.
2. Fundamentals of Algorithms by Ellis Horwitz, Sartaj Sahni, Sanguthevar Rajasekaran

NAAC GRADE A+

THANK YOU