Experiment No. : 3

Title: Virtual lab on Dijkstra's algorithm

Batch: A2 **Roll No.:16010421073** **Experiment No.: 3**

Aim: Explore the virtual lab on Dijkstra's algorithm to calculate single source shortest path

Resource Needed:
https://ds2-iiith.vlabs.ac.in/exp/dijkstra-algorithm/index.html

Algorithm of Dijkstra's Algorithm:

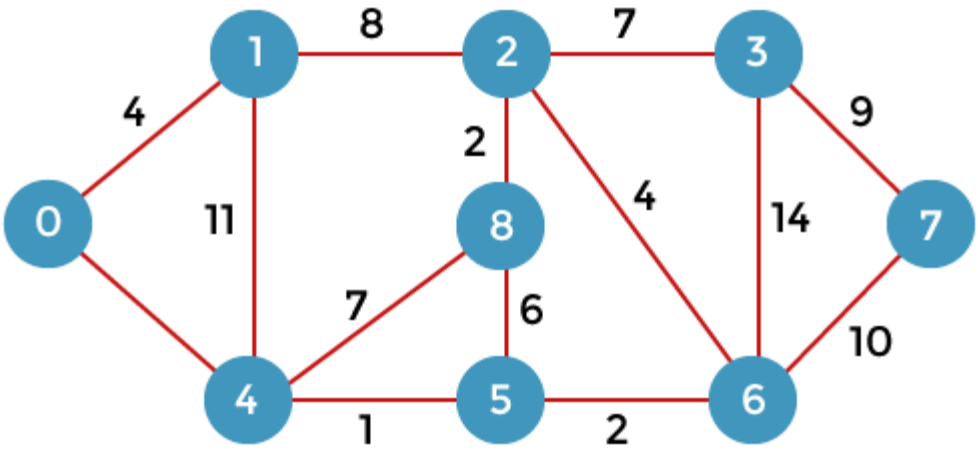
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```


Explanation and Working of Dijkstra's Algorithm:

Explanation:

This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.



Dijkstra's algorithm is the iterative algorithmic process to provide us with the shortest path from one specific starting node to all other nodes of a graph. It is different from the minimum spanning tree as the shortest distance among two vertices might not involve all the vertices of the graph.

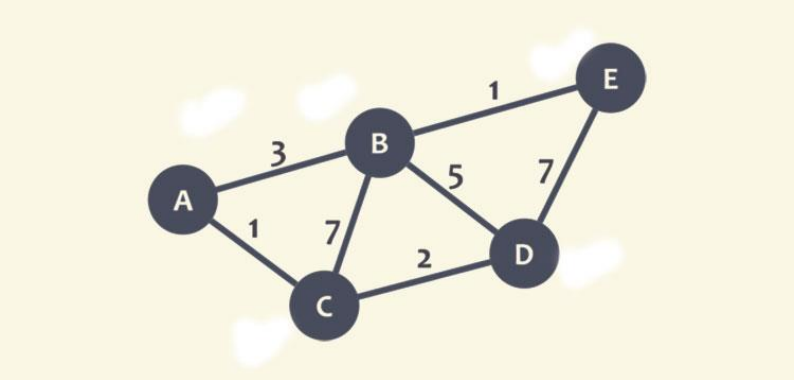
Generally, Dijkstra's algorithm works on the principle of relaxation where an approximation of the accurate distance is steadily displaced by more suitable values until the shortest distance is achieved.

Also, the estimated distance to every node is always an overvalue of the true distance and is generally substituted by the least of its previous value with the distance of a recently determined path.

Working of Dijkstra's Algorithm

In the above section, you have gained the step by step process of Dijkstra's algorithm, now let's study the algorithm with an explained example.

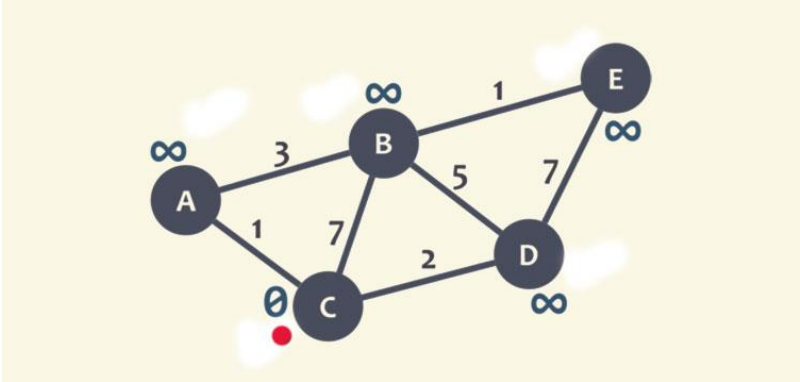
We will calculate the shortest path between node C and the other nodes in the graph.



Example of Dijkstra's Algorithm

1. During the execution of the algorithm, each node will be marked with its minimum distance to node C as we have selected node C.

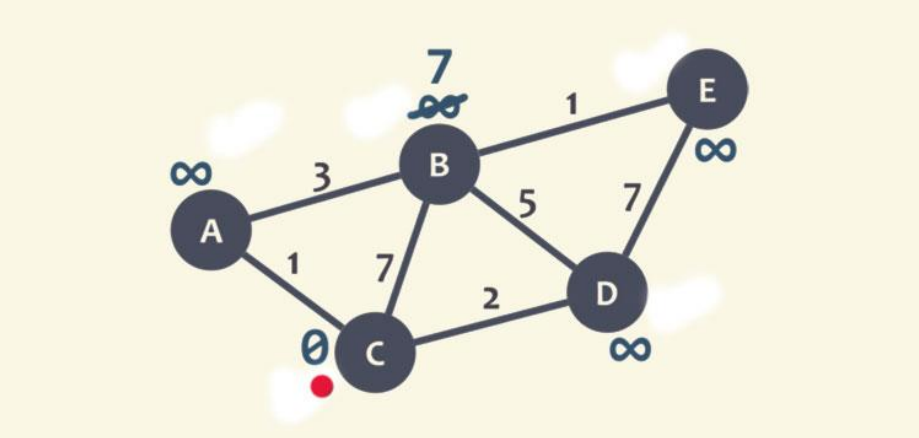
   In this case, the minimum distance is 0 for node C. Also, for the rest of the nodes, as we don't know this distance, they will be marked as infinity (∞), except node C (currently marked as red dot).



Graphical Representation of Node C as Current Node

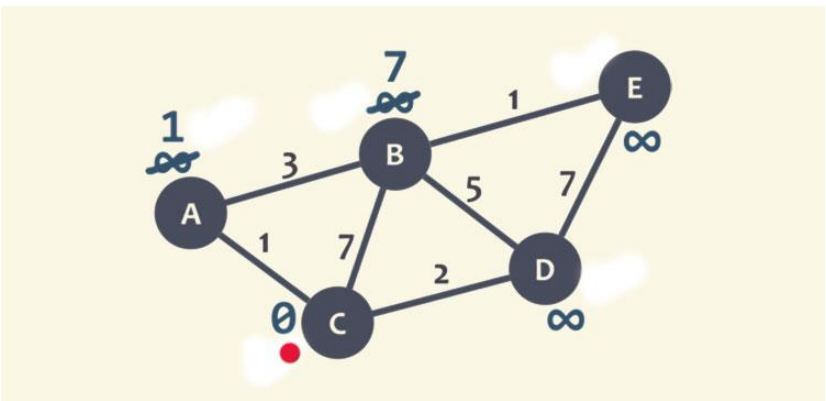
2. Now the neighbours of node C will be checked, i.e, node A, B, and D. We start with B, here we will add the minimum distance of current node (0) with the weight of the edge (7) that linked the node C to node B and get 0+ 7= 7.

   Now, this value will be compared with the minimum distance of B (infinity), the least value is the one that remains the minimum distance of B, like in this case, 7 is less than infinity, and marks the least value to node B.



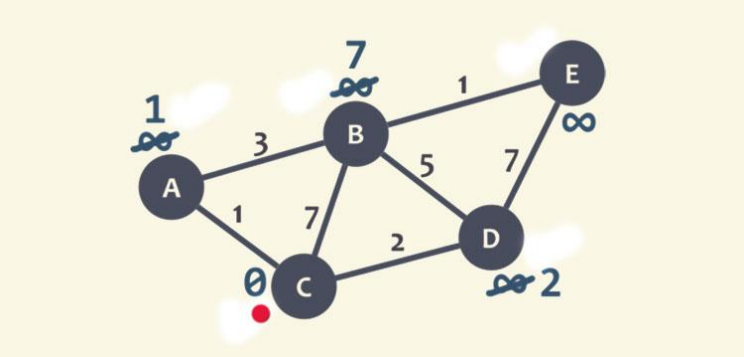
Assign Node B a minimum distance value

3. Now, the same process is checked with neighbour A. We add 0 with 1 (weight of edge that connects node C to A), and get 1. Again, 1 is compared with the minimum distance of A (infinity), and marks the lowest value.



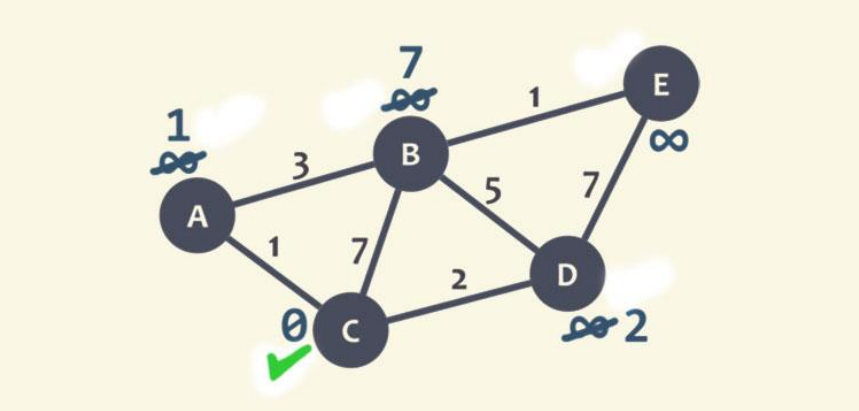
Assign Node A a minimum distance value

The same is repeated with node D, and marked 2 as lowest value at D.



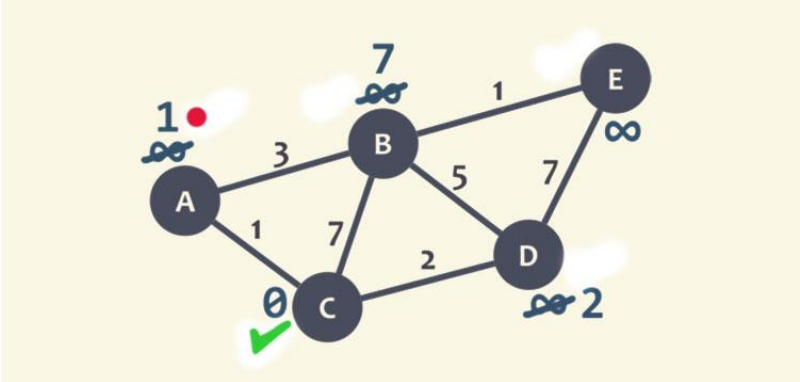
Assign Node D a minimum distance value

Since, all the neighbours of node C have checked, so node C is marked as visited with a green check mark.



Marked Node C as visited

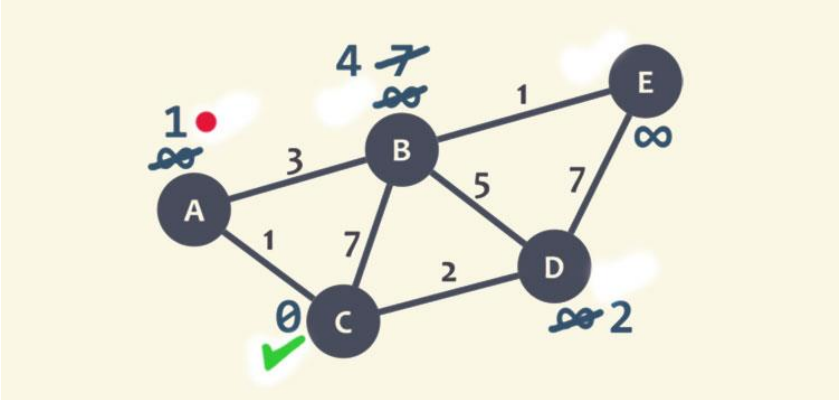
4. Now, we will select the new current node such that the node must be unvisited with the lowest minimum distance, or the node with the least number and no check mark. Here, node A is the unvisited with minimum distance 1, marked as current node with red dot.



Graphical Representation of Node A as Current Node

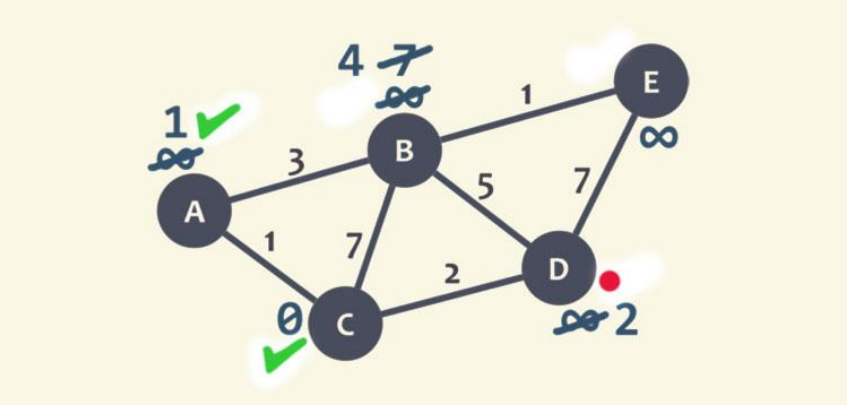
We repeat the algorithm, checking the neighbour of the current node while ignoring the visited node, so only node B will be checked.

For node B, we add 1 with 3 (weight of the edge connecting node A to B) and obtain 4. This value, 4, will be compared with the minimum distance of B, 7, and mark the lowest value at B as 4.



*Assign Node **B** a minimum distance value*

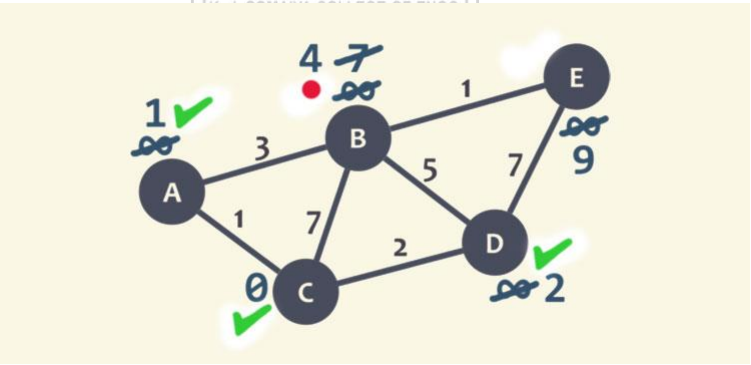
5. After this, node A marked as visited with a green check mark. The current node is selected as node D, it is unvisited and has a smallest recent distance. We repeat the algorithm and check for node B and E.



Graphical Representation of Node D as Current Node

For node B, we add 2 to 5, get 7 and compare it with the minimum distance value of B, since 7>4, so leave the smallest distance value at node B as 4.

For node E, we obtain 2+ 7= 9, and compare it with the minimum distance of E which is infinity, and mark the smallest value as node E as 9. The node D is marked as visited with a green check mark.

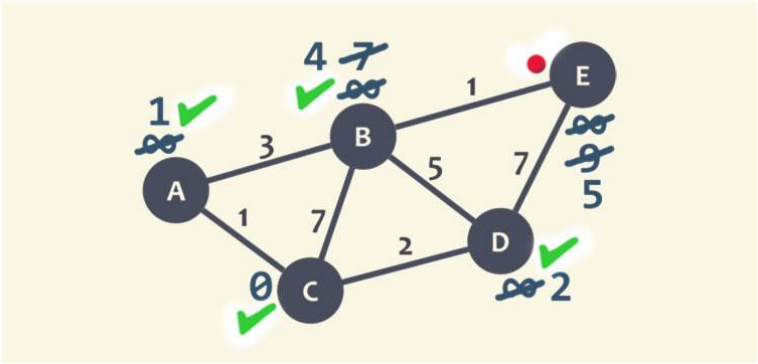


Marked Node D as visited

6. The current node is set as node B, here we need to check only node E as it is unvisited and the node D is visited. We obtain 4+ 1=5, compare it with the minimum distance of the node.

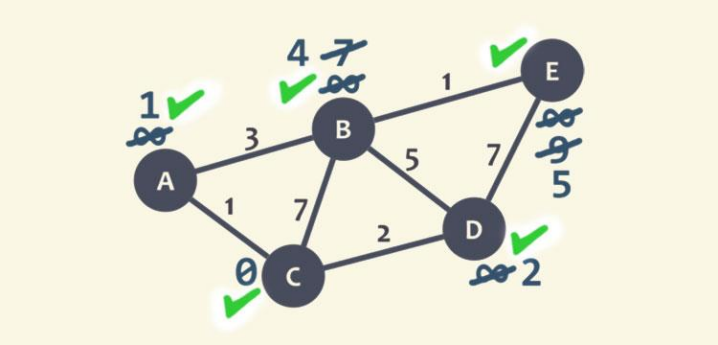
As 9 > 5, leave the smallest value at node node E as 5.

We mark D as visited node with a green check mark, and node E is set as the current node.



Marked Node B as visited

7. Since it doesn't have any unvisited neighbours, so there is not any requirement to check anything. Node E is marked as a visited node with a green mark.



Marked Node E as visited

So, we are done as no unvisited node is left. The minimum distance of each node is now representing the minimum distance of that node from node C.

Time complexity and derivation of Dijkstra's Algorithm:

Time Complexity derivation
Dijkstra Algorithm

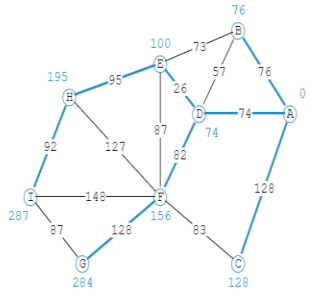
Cost		
$O(V)$	1)	Intitialize Single Source (G, s)
$O(1)$	2)	$S = \phi$
$O(V)$	3)	$Q = G.V$
	4)	while $Q \neq \phi$
$O(\log V)$ ←	5)	$u =$ Extract-Min (Q)
	6)	$S = S \cup \{u\}$
$O(V \log V)$	7)	for each vertex $v \in G.Adj[u]$
	8)	Relax (u, v, w)
		⇓ Decrease Key
$= O(V^2 \log V)$	$V.(V \log V)$	

Total time complexity:

$$\Rightarrow O(V) + O(1) + O(V) + O(V \log V) + O(V^2 \log V)$$

$$\therefore O(V^2 \log V) \text{ or } O((V^2 + V) \log V).$$

Observations from Simulation:



Unvisited Nodes

Visited Nodes

Edge traversal options

Traversed edge

Observations

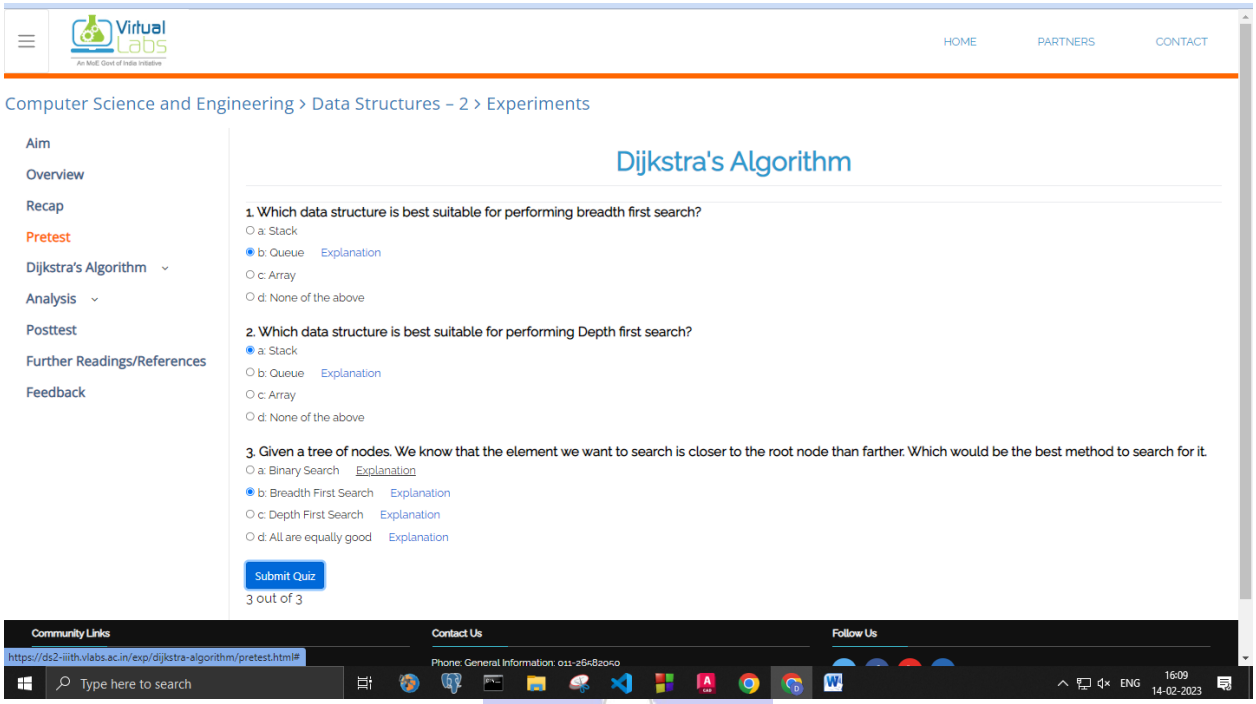
I is selected

Min. Speed Max. Speed

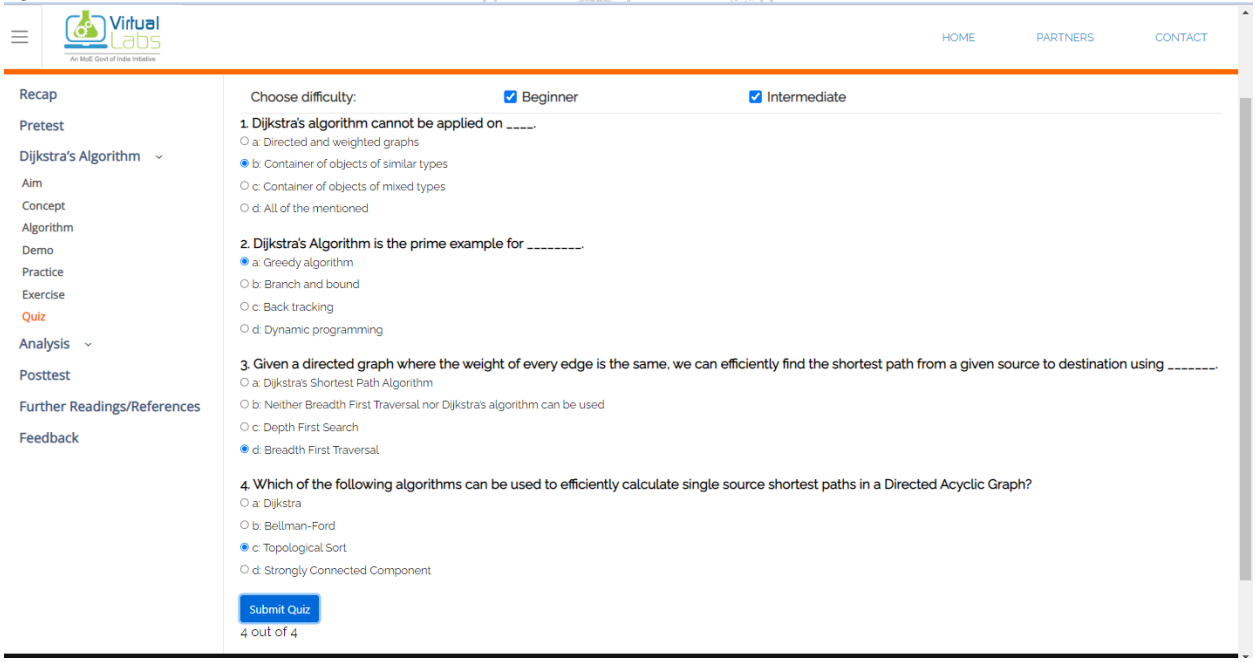
Next Reset Play

Self-evaluation: Solve both, Dijkstra's Algorithm Quiz and Analysis Quiz, and display the result of your first attempt.

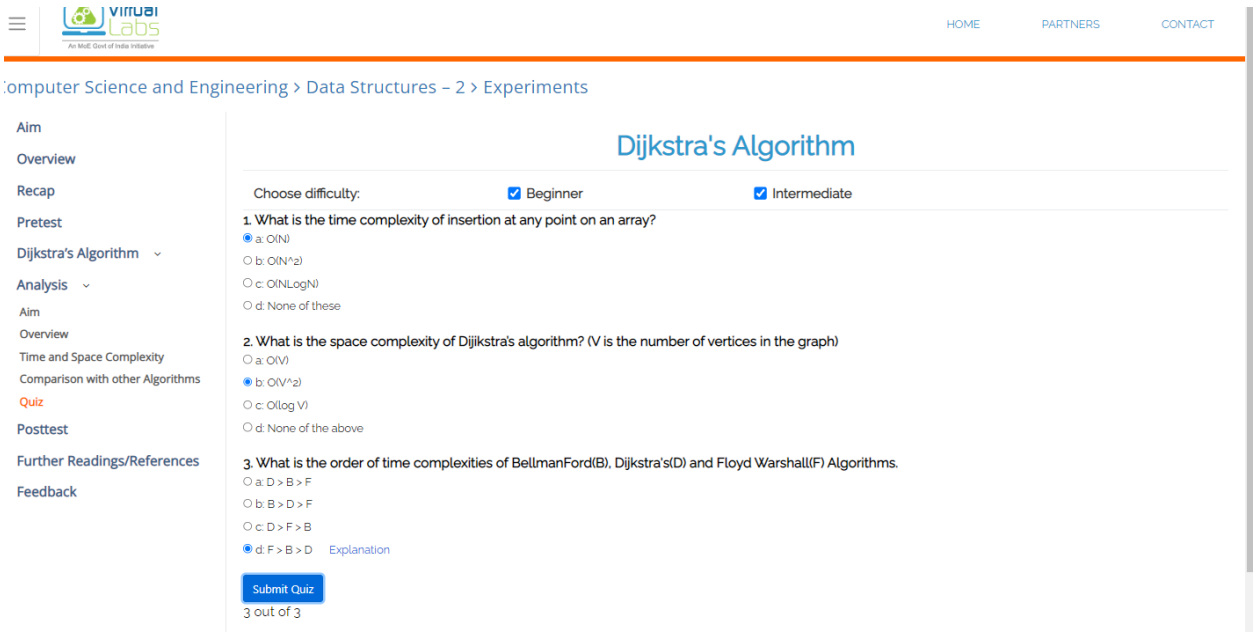
Pretest:



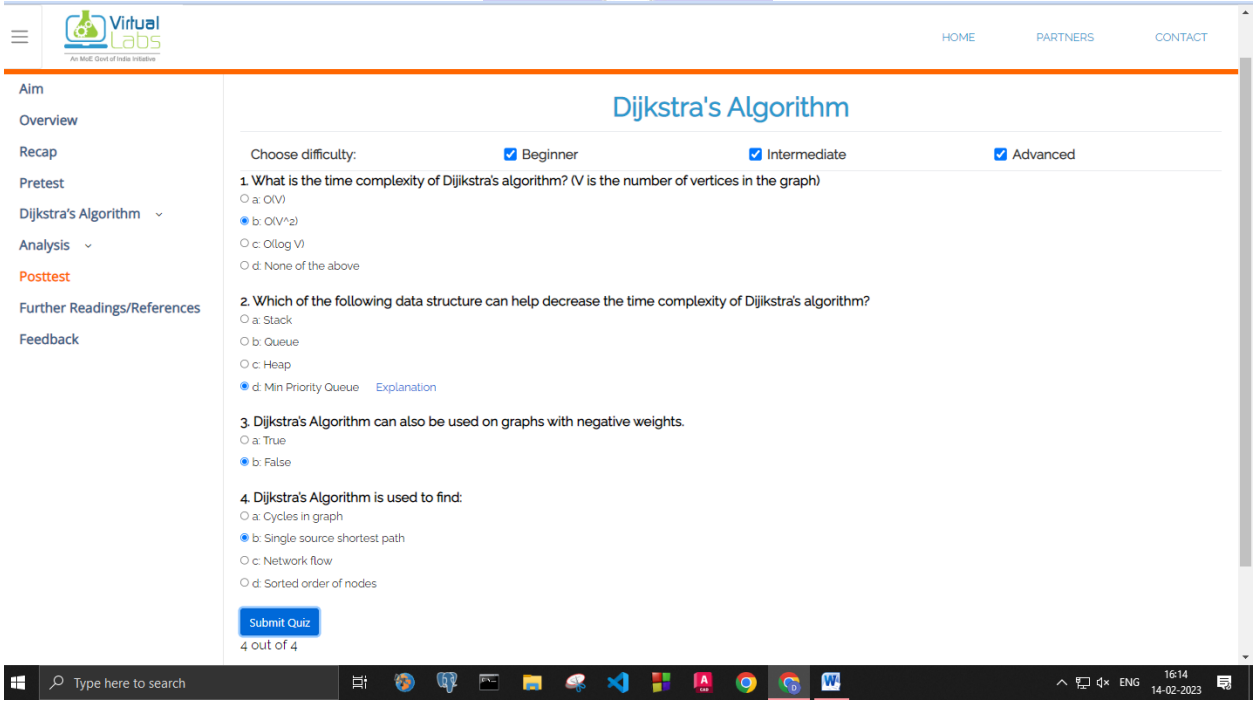
Quiz:



Time complexity quiz:


HOME
PARTNERS
CONTACT
Computer Science and Engineering > Data Structures – 2 > Experiments
Aim
Overview
Recap
Pretest
Dijkstra's Algorithm
Analysis
Aim
Overview
Time and Space Complexity
Comparison with other Algorithms
Quiz
Posttest
Further Readings/References
Feedback
Dijkstra's Algorithm
Choose difficulty:
Beginner
Intermediate
1. What is the time complexity of insertion at any point on an array?
a: O(N)
b: O(N^2)
c: O(NLogN)
d: None of these
2. What is the space complexity of Dijikstra's algorithm? (V is the number of vertices in the graph)
a: O(V)
b: O(V^2)
c: O(log V)
d: None of the above
3. What is the order of time complexities of BellmanFord(B), Dijkstra's(D) and Floyd Warshall(F) Algorithms.
a: D > B > F
b: B > D > F
c: D > F > B
d: F > B > D
Explanation
Submit Quiz
3 out of 3

Post-test:


HOME
PARTNERS
CONTACT
Aim
Overview
Recap
Pretest
Dijkstra's Algorithm
Analysis
Posttest
Further Readings/References
Feedback
Dijkstra's Algorithm
Choose difficulty:
Beginner
Intermediate
Advanced
1. What is the time complexity of Dijikstra's algorithm? (V is the number of vertices in the graph)
a: O(V)
b: O(V^2)
c: O(log V)
d: None of the above
2. Which of the following data structure can help decrease the time complexity of Dijikstra's algorithm?
a: Stack
b: Queue
c: Heap
d: Min Priority Queue
Explanation
3. Dijkstra's Algorithm can also be used on graphs with negative weights.
a: True
b: False
4. Dijkstra's Algorithm is used to find:
a: Cycles in graph
b: Single source shortest path
c: Network flow
d: Sorted order of nodes
Submit Quiz
4 out of 4
Type here to search
ENG
16:14
14-02-2023

Conclusion: (Based on the observations):

Thus we successfully simulated and understood the working of Dijkstra's algorithm in virtual lab.

Outcome:

CO2: Implement Greedy and Dynamic Programming algorithms.

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.