Experiment No. : 3

Title: Virtual lab on Dijkstra’s algorithm

Batch: B2 **Roll No.: 16010421119**
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  $S = \emptyset$
3  $Q = G.V$
4  while $Q \neq \emptyset$
5      $u$ = EXTRACT-MIN($Q$)
6      $S = S \cup \{u\}$
7      for each vertex $v \in G.Adj[u]$
8          RELAX($u, v, w$)
```

Explanation and Working of Dijkstra's Algorithm:

- Set the distance to the source to 0 and the distance to the remaining vertices to infinity.
- Set the current vertex to the source.
- Flag the current vertex as visited.
- For all vertices adjacent to the current vertex, set the distance from the source to the adjacent vertex equal to the minimum of its present distance and the sum of the weight of the edge from the current vertex to the adjacent vertex and the distance from the source to the current vertex.
- From the set of unvisited vertices, arbitrarily set one as the new current vertex, provided that there exists an edge to it such that it is the minimum of all edges from a vertex in the set of visited vertices to a vertex in the set of unvisited vertices. To reiterate: The new current vertex must be unvisited and have a minimum weight edges from a visited vertex to it. This can be done trivially by looping through all visited vertices and all adjacent unvisited vertices to those visited vertices, keeping the vertex with the minimum weight edge connecting it.

- Repeat steps 3-5 until all vertices are flagged as visited.

Pseudocode: -

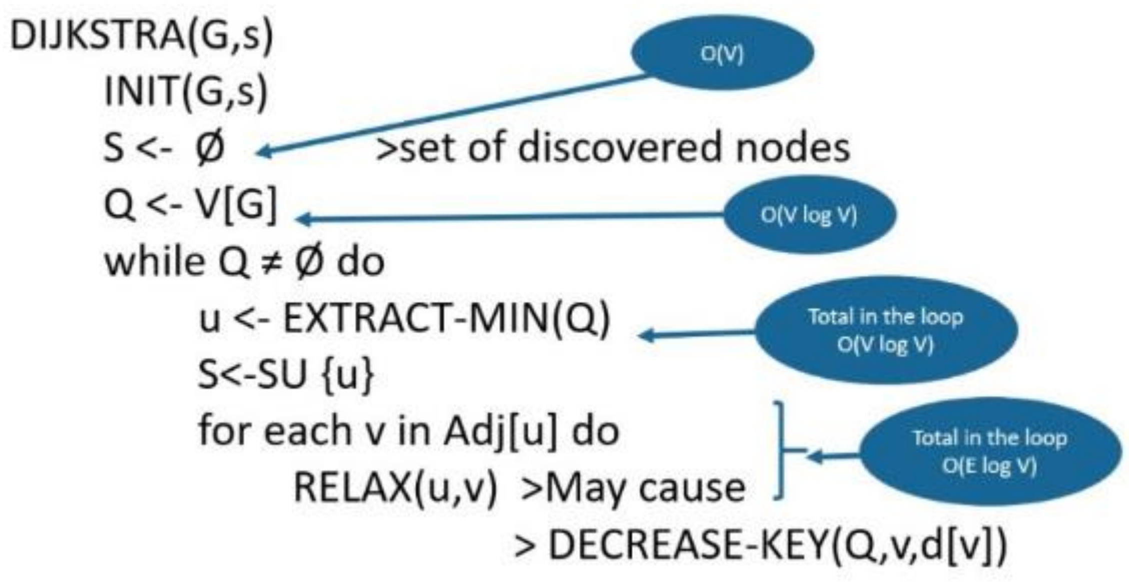
```
function Dijkstra(Graph, source):
dist[source] ← 0 // Initialization
create vertex set Q
for each vertex v in Graph:
if v ≠ source
dist[v] ← INFINITY // Unknown distance from source to v
prev[v] ← UNDEFINED // Predecessor of v
Q.add_with_priority(v, dist[v])

while Q is not empty: // The main loop
u ← Q.extract_min() // Remove and return best vertex
for each neighbor v of u: // only v that are still in Q
alt ← dist[u] + length(u, v)
if alt < dist[v]
dist[v] ← alt
prev[v] ← u
Q.decrease_priority(v, alt)
return dist, prev
```

Time complexity and derivation of Dijkstra's Algorithm:

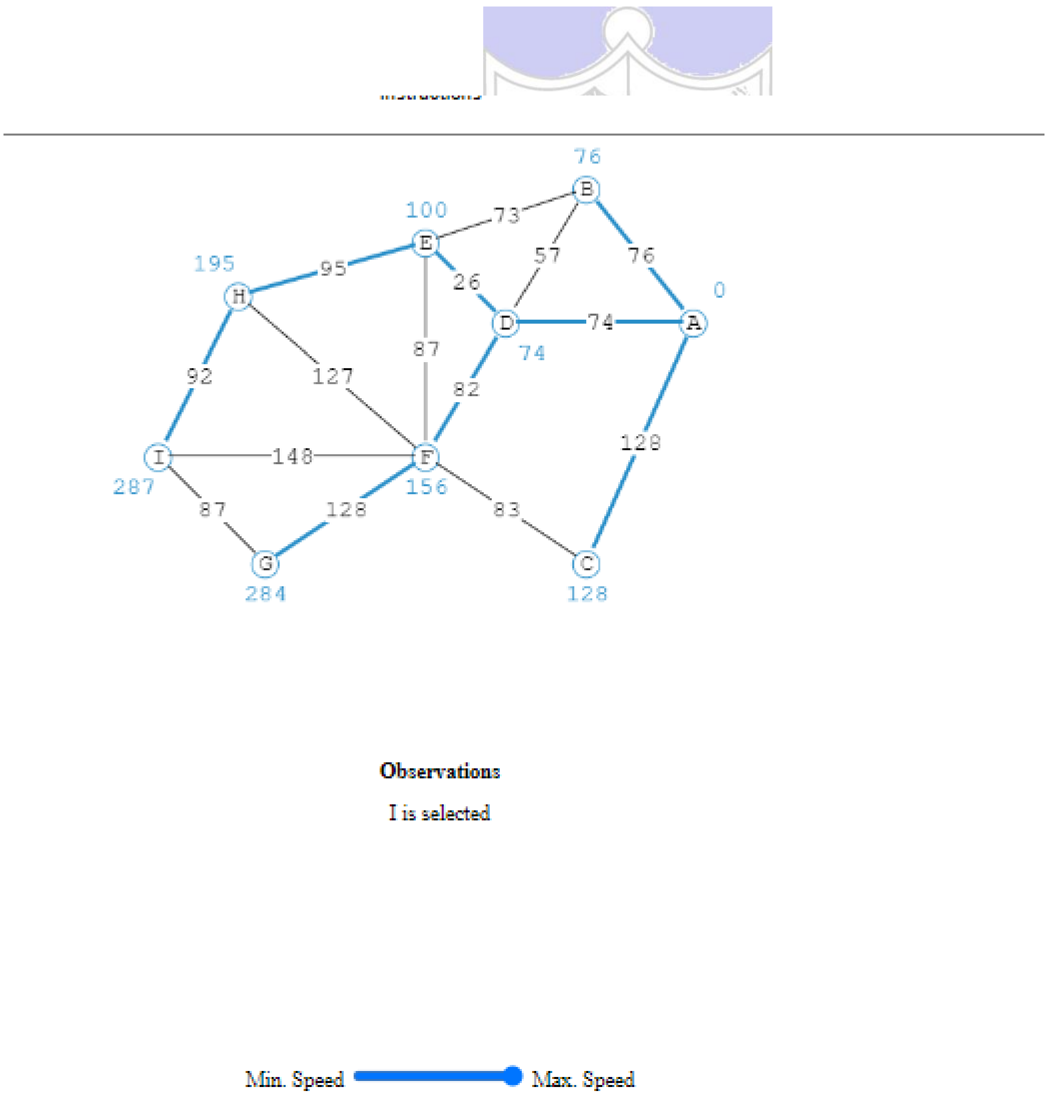
The time complexity of the given code/algorithm looks O(V^2) as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS). The inner loop has decreaseKey() operation which takes O(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV) Note that the above code uses Binary Heap for Priority Queue implementation. Time complexity can be reduced to O(E + VLogV) using Fibonacci Heap. The reason is, Fibonacci Heap takes O(1) time for decrease-key operation while Binary Heap takes O(Logn) time.

Space complexity of Dijkstra's algorithm is O(V2) where V denotes the number of vertices (or nodes) in the graph.



```
DIJKSTRA(G,s)
    INIT(G,s)
    S <- Ø          >set of discovered nodes
    Q <- V[G]
    while Q ≠ Ø do
        u <- EXTRACT-MIN(Q)
        S<-SU {u}
        for each v in Adj[u] do
            RELAX(u,v)  >May cause
                        > DECREASE-KEY(Q,v,d[v])
```

Observations from Simulation:

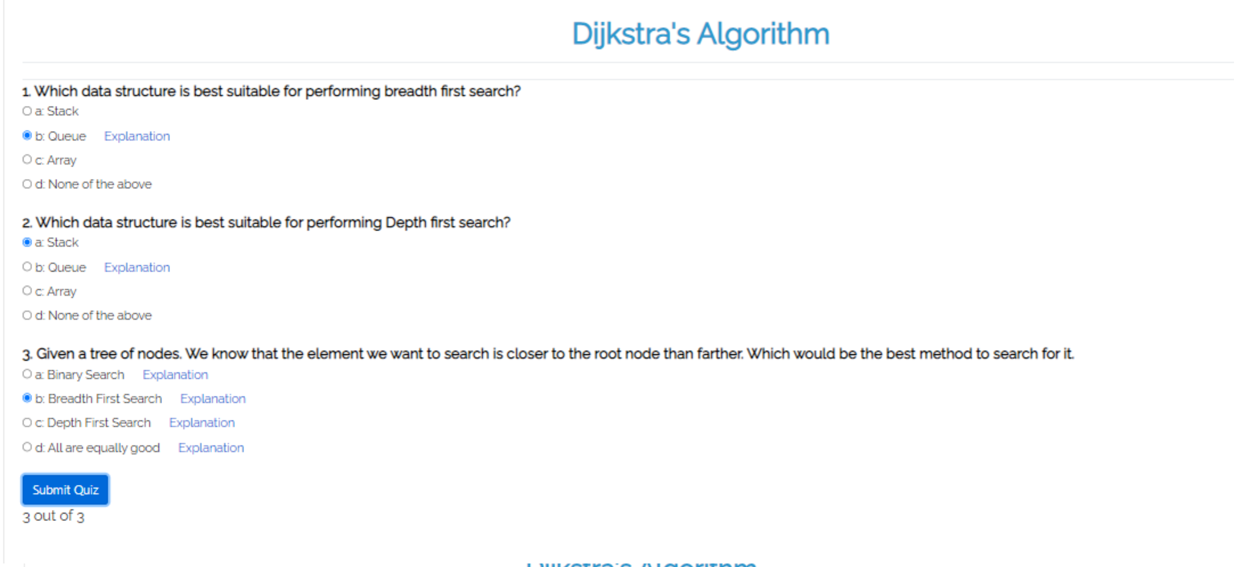


Observations

I is selected

Min. Speed Max. Speed

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



Dijkstra's Algorithm

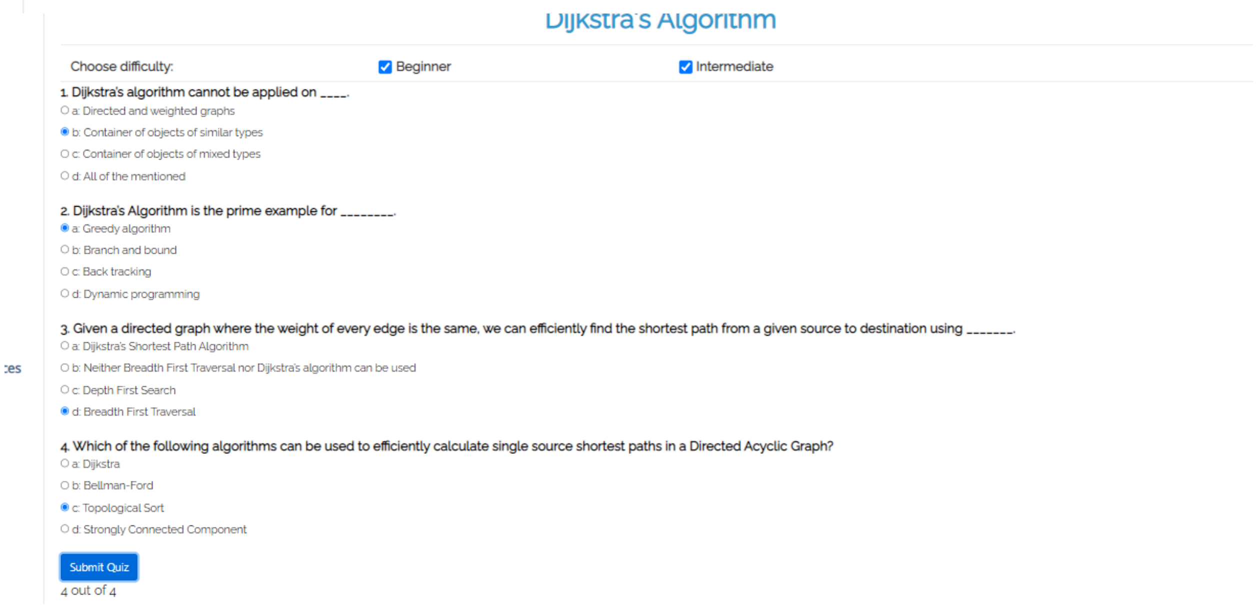
1. Which data structure is best suitable for performing breadth first search?
- a: Stack
- [✓ b: Queue] Explanation
- c: Array
- d: None of the above

2. Which data structure is best suitable for performing Depth first search?
- [✓ a: Stack]
- b: Queue Explanation
- c: Array
- d: None of the above

3. Given a tree of nodes. We know that the element we want to search is closer to the root node than farther. Which would be the best method to search for it.
- a: Binary Search Explanation
- [✓ b: Breadth First Search] Explanation
- c: Depth First Search Explanation
- d: All are equally good Explanation

Submit Quiz

3 out of 3



Dijkstra's Algorithm

Choose difficulty: ✓ Beginner ✓ Intermediate

1. Dijkstra's algorithm cannot be applied on ____.
- a: Directed and weighted graphs
- [✓ b: Container of objects of similar types]
- c: Container of objects of mixed types
- d: All of the mentioned

2. Dijkstra's Algorithm is the prime example for ________.
- [✓ a: Greedy algorithm]
- b: Branch and bound
- c: Back tracking
- d: Dynamic programming

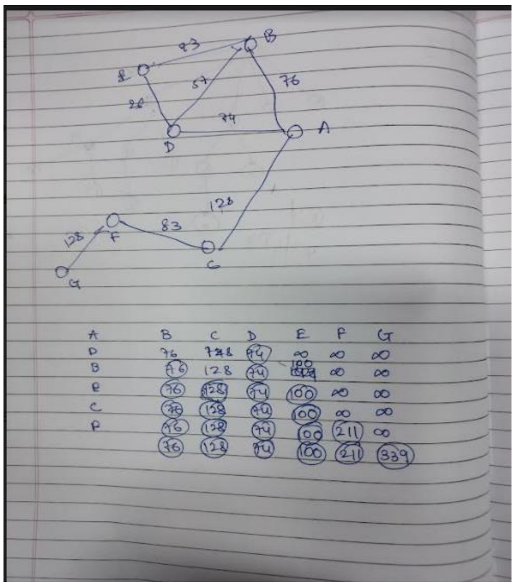
3. Given a directed graph where the weight of every edge is the same, we can efficiently find the shortest path from a given source to destination using _______.
- a: Dijkstra's Shortest Path Algorithm
- b: Neither Breadth First Traversal nor Dijkstra's algorithm can be used
- c: Depth First Search
- [✓ d: Breadth First Traversal]

4. Which of the following algorithms can be used to efficiently calculate single source shortest paths in a Directed Acyclic Graph?
- a: Dijkstra
- b: Bellman-Ford
- [✓ c: Topological Sort]
- d: Strongly Connected Component

Submit Quiz

4 out of 4

Solution for the Given Question:



A	B	C	D	E	F	G
D	76	128	74	∞	∞	∞
B	76	128	74	100	∞	∞
E	76	128	74	100	∞	∞
C	76	128	74	100	∞	∞
F	76	128	74	100	211	∞
	76	128	74	100	211	339

Conclusion: (Based on the observations):

We can conclude that we have learnt about Djikstra's algorithm .

Outcome:

CO2: - Implementing 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.