Experiment No. : 3

Title: Virtual lab on Dijkstra’s algorithm

Batch:A2 **Roll No.:16010421059** **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.
-
- **Working:**

Dijkstra's Algorithm requires a graph and source vertex to work. The algorithm is purely based on greedy approach and thus finds the locally optimal choice(local minima in this case) at each step of the algorithm.

In this algorithm each vertex will have two properties defined for it-

- **Visited property**:-
    - This property represents whether the vertex has been visited or not.
    - We are using this property so that we don't revisit a vertex.
    - A vertex is marked visited only after the shortest path to it has been found.
- **Path property**:-
    - This property stores the value of the current minimum path to the vertex. Current minimum path means the shortest way in which we have reached this vertex till now.
    - This property is updated whenever any neighbour of the vertex is visited.
    - The path property is important as it will store the final answer for each vertex.

Initially all the vertices are marked unvisited as we have not visited any of them. Path to all the vertices is set to infinity excluding the source vertex. Path to the source vertex is set to zero(0).

Then we pick the source vertex and mark it visited.After that all the neighbours of the source vertex are accessed and **relaxation** is performed on each vertex. Relaxation is the process of trying to lower the cost of reaching a vertex using another vertex.

In relaxation, the path of each vertex is updated to the minimum value amongst the current path of the node and the sum of the path to the previous node and the path from the previous node to this node.

Assume that p[v] is the current path value for node v, p[n] is the path value upto the previously visited node n, and w is the weight of the edge between the curent node and previously visited node(edge weight between v and n)

Mathematically, relaxation can be represented as: **p[v]=minimum(p[v] , p[n] + w)**

Then in every subsequent step, an unvisited vertex with the least path value is marked visited and its neighbour's paths updated.

The above process is repeated till all the vertices in the graph are marked visited.

Whenever a vertex is added to the visited set, the path to all of its neighbouring vertices is changed according to it.

If any of the vertex is not reachable(disconnected component), its path remains infinity. If the source itself is a disconected component, then the path to all other vertices remains infinity

Time complexity and derivation of Dijkstra's Algorithm:

Complexity analysis for dijkstra's algorithm with adjacency matrix representation of graph.

Time complexity of Dijkstra's algorithm is ***O*(*V*^2)** where **V** is the number of vertices in the graph.

It can be explained as below:

1. First thing we need to do is find the unvisited vertex with the smallest path. For that we require ***O*(*V*)** time as we need check all the vertices.
2. Now for each vertex selected as above, we need to relax its neighbours which means to update each neighbours path to the smaller value between its current path or to the newly found. The time required to relax one neighbour comes out to be of order of **O(1)** (constant time).
3. For each vertex we need to relax all of its neighbours, and a vertex can have at most V-1 neighbours, so the time required to update all neighbours of a vertex comes out to be [O(V) * O(1)] = O(V)

So now following the above conditions, we get:

Time for visiting all vertices =O(V)

Time required for processing one vertex=O(V)

Time required for visiting and processing all the vertices = O(V)*O(V) =O(V^2)

So the time complexity of dijkstra's algorithm using adjacency matrix representation comes out to be ***O*(*V*2)**

Space complexity of adjacency matrix representation of graph in the algorithm is also ***O*(*V*2)** as a V*V matrix is required to store the representation of the graph. An additional array of V length will also be used by the algorithm to maintain the states of each vertex but the total space complexity will remain **O(*V*2)**.

The time complexity of dijkstra's algorithm can be reduced to **O((V+E)logV)** using adjacency list representation of the graph and a min-heap to store the unvisited vertices, where E is the number of edges in the graph and V is the number of vertices in the graph.

With this implementation, the time to visit each vertex becomes **O(V+E)** and the time required to process all the neighbours of a vertex becomes **O(logV)**.

Time for visiting all vertices =O(V+E)

Time required for processing one vertex=O(logV)

Time required for visiting and processing all the vertices = O(V+E)*O(logV) = O((V+E)logV)

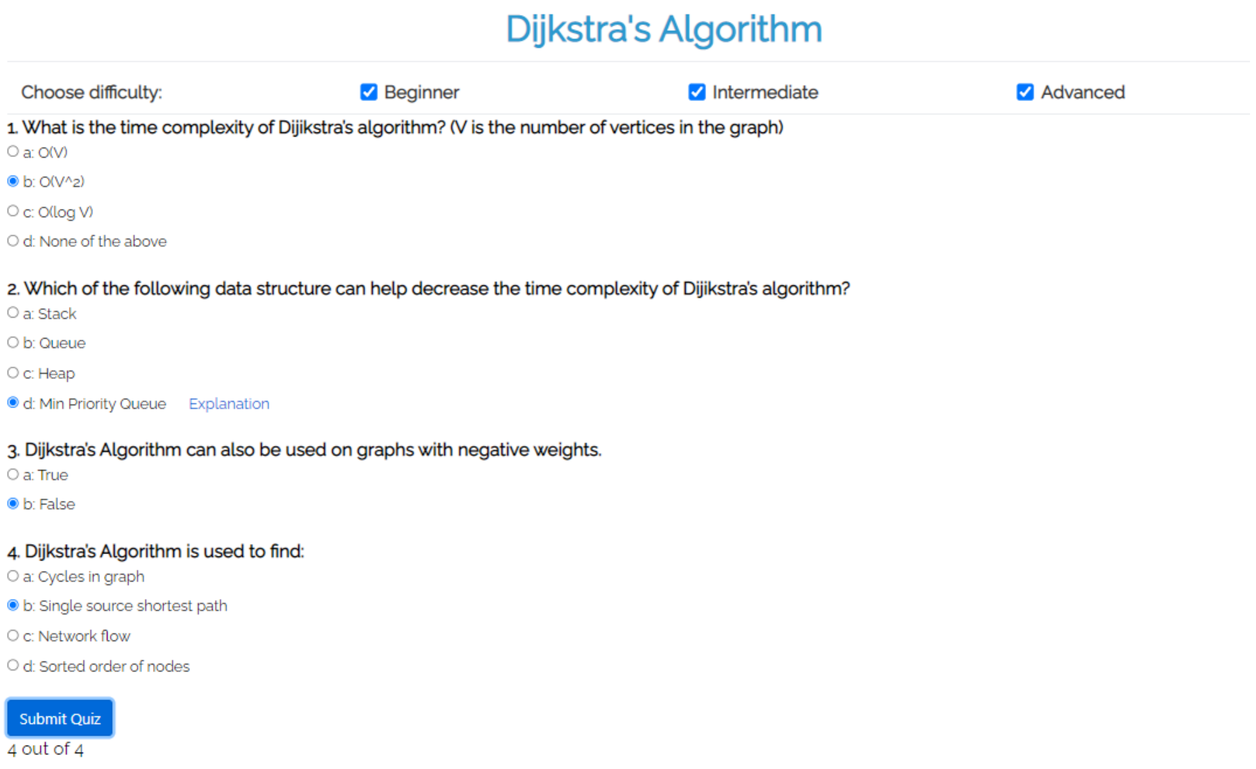
The space complexity in this case will also improve to **O(V)** as both the adjacency list and min-heap require **O(V)** space. So the total space complexity becomes

O(V)+O(V)=O(2V) = O(V)

Observations from Simulation:

Simulation not working.

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



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

Conclusion: (Based on the observations):

Simulation not working.

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.