**Question 1. All or None**

**Divyansh Mathur, 210357**

1. Depth-first traversal aims to ensure that each vertex is visited exactly once by recursively exploring unvisited neighboring vertices.

However, there is essentially no guarantee that each edge is traversed exactly once.

Our particular problem may involve situations where a vertex needs to be revisited and requires traversal of multiple edges connected to the same vertex.

1. In commonly applied breadth-first traversal, vertices are visited in order of increasing distance from the source vertex, resulting in a BFS tree in which each vertex (except the root) contains exactly one input edge.

will be formed.

Connected.

This inherent property of BFS does not provide a guarantee that he will find a path that covers each edge exactly once.

This is because the problem may require revisiting at least one vertex multiple times to achieve this.

1. To cross each road in a given diagram exactly once and return to the starting city, two basic conditions must be met: A single connected component: The diagram is One connected component must be formed.

This ensures that a path exists between each pair of cities in the graph, eliminating sections and ensuring access to the entire network.

Even vertex degree: For each city (vertex) in the graph, the number of streets (edges) connected to it must be an even number.

This condition ensures that all roads leading into a city pass through it only once, and that the road returns to the city each time it leaves the city.

In mathematical terms, this means vertices of even degree indicating an even number of visits to the circuit.

These conditions are very useful for setting up a racetrack that can reliably return to the starting city without untraveled roads or isolated components, while still being able to systematically traverse all roads in the graphic.

It is important.

1. **Algorithm Steps:**

**Initialization:**

Begin with an empty stack to track vertices during traversal.

Create an empty path to record the order of visited vertices.

**Select a Starting Vertex:**

Choose any vertex from the graph as the initial current vertex for exploration.

**Exploration Loop:**

While the current vertex has at least one adjacent neighbor, continue with the following steps:

**Step A: Push the Current Vertex:**

Push the current vertex onto the stack to maintain traversal order.

**Step B: Remove the Edge:**

Delete the edge between the current vertex and the encountered adjacent vertex. This ensures that each edge is traversed exactly once.

**Step C: Update Current Vertex:**

Set the adjacent vertex as the new current vertex for further exploration.

**Backtracking:**

If the current vertex has no remaining neighbors (all edges have been traversed), perform the following actions:

**Step A: Update the Path:**

Add the current vertex to the path to document the order of visited vertices.

**Step B: Pop from the Stack:**

Pop a vertex from the stack, signifying backtracking to explore other untraveled paths if available.

**Step C: Set the Popped Vertex as Current:**

Assign the popped vertex as the new current vertex to continue exploration.

**Repeat the Exploration:**

Continue with the exploration, revisiting steps 4 and 5 until the stack becomes empty, and the current vertex no longer has unexplored neighbors.

This algorithm systematically explores the graph, removing edges as they are traversed, ensuring that every edge is visited exactly once while constructing an Eulerian path or circuit. It offers an efficient and structured approach to solving graph traversal problems of this nature.