Graphs-2

Assignment Solutions

PW SKILLS

Q1. ASS_Code1.java

- When we do a Depth–first search (DFS) from any vertex v in an undirected graph, we may encounter a back-edge that points to one of the ancestors of the current vertex v in the DFS tree.
- Each "back edge" defines a cycle in an undirected graph. If the back edge is x —> y, then since y is the ancestor of node x, we have a path from y to x.
- So, we can say that we have a path y ~~ x ~ y that forms a cycle. (Here, ~~ represents one more edge in the path, and ~ represents a direct edge).

Q2. ASS_Code2.java

Approach:
We can use Depth–first search (DFS) to solve this problem. The idea is to find if any back-edge is present in the graph or not. A digraph is a DAG if there is no back-edge present in the graph. Recall that a back-edge is an edge from a vertex to one of its ancestors in the DFS tree.

Fact: For an edge u —> v in a directed graph, an edge is a back edge if departure[u] < departure[v].

Q3. ASS_Code3.java

Whenever we see the term shortest, the first thing we should think about is doing a BFS traversal. So, here also, we start BFS traversal from the given source vertex. Usually, BFS doesn't explore already discovered vertices again, but here we do the opposite. To cover all possible paths from source to destination, remove this check from BFS. But what if the graph contains a cycle? Removing this check will cause the program to go into an infinite loop. We can easily handle that if we don't consider nodes having a BFS depth of more than m.
The solution below maintains the following information in a BFS queue node:

- The current vertex number.
- The current depth of BFS (i.e., how far the current node is from the source?).
- The cost of the current path chosen so far.

Whenever we encounter any node whose cost of a path is more and required BFS depth is reached, update the result. The BFS will terminate when we have explored every path in the given graph or BFS depth exceeds m.

Q4. ASS_Code4.java

The idea is to do a BFS traversal from the given source vertex. BFS is generally used to find the shortest paths in graphs/matrices, but we can modify normal BFS to meet our requirements. Usually, BFS doesn't explore already discovered vertices again, but here we do the opposite. To cover all possible paths from source to destination, remove this check from BFS. But if the graph contains a cycle, removing this check will cause the program to go into an infinite loop. We can easily handle that if we don't consider nodes having a BFS depth of more than m.
Basically, we maintain two things in the BFS queue node:

- The current vertex number.
- The current depth of BFS (i.e., how far away from the current node is from the source?).

So, whenever the destination vertex is reached and BFS depth is equal to m, we update the result. The BFS will terminate when we have explored every path in the given graph or BFS depth exceeds m.

Q5. ASS_Code5.java

A directed graph has an Eulerian path if and only if the following conditions are satisfied:
At most one vertex in the graph has out-degree = 1 + in-degree, and at most one vertex in the graph has in-degree = 1 + out-degree. All the remaining vertices have in-degree == out-degree.
All vertices with a non-zero degree belong to a single connected component of the underlying undirected graph. In other words, if one removes the “directness” of edges, then the graph without isolated vertices should be connected.
For example, in the above graph, the only vertex 0 has out-degree one more than the in-degree, and vertex 4 has in-degree one more than the out-degree. Every other vertex has equal in-degree and out-degree, and the non-isolated vertices are weakly connected.