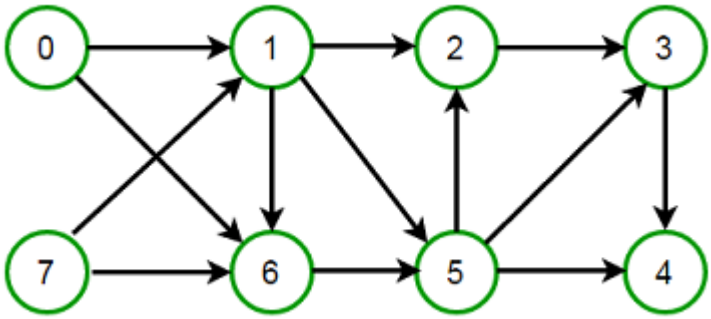
Name: _____ Reg #: _____ Section: _____

National University of Computer and Emerging Sciences, Lahore Campus

Course:	Design and Analysis of Algorithms	Course Code:	CS2009
Program:	BS(Computer Science)	Semester:	Fall 2022
Duration:	20 Minutes	Total Marks:	10
Paper Date:	28-Nov-2022	Weight	%
Section:	5A	Page(s):	3
Exam:	Quiz 4		

Q1) Given a directed graph, find the total number of routes to reach the destination from a given source with exactly m edges. For example, consider the following graph: [10 Marks]



Let source = 0, destination = 3, number of edges m = 4. The graph has 3 routes from source 0 to destination 3 with 4 edges. The solution should return the total number of routes 3. Give an efficient algorithm for solving this problem and analyze its time complexity.

0 —> 1 —> 5 —> 2 —> 3
0 —> 1 —> 6 —> 5 —> 3
0 —> 6 —> 5 —> 2 —> 3

Solution:

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`.

```
int findTotalPaths(Graph const &graph, int src, int dest, int m)
{
    // create a queue for doing BFS
    queue<Node> q;

    // enqueue source vertex
    q.push({src, 0});

    // stores number of paths from source to destination having exactly `m` edges
    int count = 0;

    // loop till queue is empty
    while (!q.empty())
    {
        // dequeue front node
        Node node = q.front();
        q.pop();

        int v = node.vertex;
        int depth = node.depth;

        // if the destination is reached and BFS depth is equal to `m`, update count
        if (v == dest && depth == m) {
            count++;
        }

        // don't consider nodes having a BFS depth more than `m`.
        // This check will result in optimized code and handle cycles
        // in the graph (otherwise, the loop will never break)
        if (depth > m) {
            break;
        }

        // do for every adjacent vertex `u` of `v`
        for (int u: graph.adjList[v])
        {
            // enqueue every vertex (discovered or undiscovered)
            q.push({u, depth + 1});
        }
    }

    // return number of paths from source to destination
    return count;
}
```

Time complexity of your algorithm: