README FILE

System Programming 2 – EX_1

Full name: Daniel Kuris

ID: 214539397

Email: danielkuris6@gmail.com

Graph structure:

I made graph class have the number of verticals and edges as a variable.

I made checks to determine if a graph is not squared, empty, or non-zero diagonal. In which case – invalid.

I calculated the edges by the amount of non-zero in the matrix disregarding the diagonal.

Functions:

negativeCycle:

The method takes a Graph object as an argument and returns a string indicating whether the graph contains a negative cycle or not.

The method begins by getting the number of vertices in the graph and storing it in V variable.

Next, it initializes a dist vector with size V and sets all elements to INT_MAX, which represents infinity in this context. This vector will hold the shortest distance from the source vertex (which is vertex 0 in this case) to every other vertex. The distance to the source vertex itself is set to 0.

The method then performs the Bellman-Ford algorithm to find the shortest paths from the source vertex to all other vertices.

isBipartite:

It checks whether a given graph is bipartite or not. A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets such that every edge connects a vertex in the first set to one in the second set.

The function begins by initializing a vector color with the size equal to the number of vertices in the graph, and all elements set to -1, indicating that no vertex has been assigned a color yet. It also initializes a queue q and two vectors partitionA and partitionB to store the vertices of the two partitions of the graph. The function then iterates over all vertices of the graph. If a vertex has not been colored yet, it is assigned color 1, added to the queue, and added to partition A. The function then enters a while loop that continues until the queue is empty, performing BFS.

isContainCycle:

The isContainsCycle() method checks if a given graph contains a cycle. It does this by iterating over all vertices of the graph and performing a depth-first search (DFS) starting from each vertex. The DFS is performed by the helper method isContainsCycleUtil().

The isContainsCycleUtil() method takes a graph, a vertex v, a visited vector to keep track of visited vertices, a parent integer to keep track of the parent of the current vertex, a path vector to store the path of the cycle if one is found, and a start vertex.

shortestPath:

The method first checks if the start and end vertices are the same. If they are, it returns a string "Invalid request - path to itself" because a path from a vertex to itself is not meaningful in this context.

Next, it checks if the start and end vertices are within the valid range of vertices in the graph. If either of them is not, it returns a string "Invalid start or end vertex".

It creates two vectors, dist and prev, of size V. dist is initialized with the maximum integer value for all vertices, and prev is initialized with -1 for all vertices. The distance from the start vertex to itself is set to 0. Then the method uses the Bellman Frod algorithm

isConnected:

The purpose of this method is to determine if a given graph is connected or not.

The method takes a Graph object as a parameter. It first gets the number of vertices in the graph using the getNumVertices method of the Graph class and stores it in V. It then creates a boolean vector visited of size V and initializes all elements to false. This vector is used to keep track of which vertices have been visited during the BFS traversal.

Tests:

The test includes extreme cases that I chose to deal with in a certain way:

Empty graph – Invalid

Non-zero diagonal – Invalid

Cycle – when encountering multiple cycles, the function will return the first and shortest it finds. It will return the path itself with a positive cycle.

In case of negative cycles – the functions will return if one exists. If shortest path managed to capture the negative cycle, it will return that it did so.

In case of 1 vertical – shortest path is invalid, the graph is connected and isn’t bipartite.

In general – shortest path to itself is invalid.

Shortest path out of bonds is invalid as well.