CMPE 343 HW 2 REPORT

Information:

4865115490
Fatih Mehmet Yiğitel
Section 02
HW2

Problem Statement and Code Design:

Task1:

In the problem we have some initial configuration for the password and with changing digits we must reach the destination password. There are some digits that starting from 0 and ending at 9. With turning wheels digits can be changed. But there are some forbidden cases that we cannot passes. We are asked to find how much steps (least) is required to reach destination password. To do that we implemented an undirected Graph API.

Implementing necessary functions and constructors for the graphs

Implementing BreadthFirstPaths class

In the main Adding all the vertices and edges to the graph

Using BFS for finding the shortest path for source to destination

Task2:

In the task 2 we are asked to create a maze that includes cycles it's inside. To create a maze we have to get the input from files. Files includes maze size information at the top and the slashes that we design and implement to the graphs. Aim was finding the longest cycle in the maze. To do that I use BreadthFirstPaths and Cycle.java classes. After the creating maze I use the hasSelfLoop methods.

Implementation and Functionality

- **Task1:**

I used BreadthFirstPaths and Graph class from the Sedgewick Algorithms book.

Graph class:

public Graph (int v);

In the constructer we set number of vertices v and assign 0 to Edge. We use bag structure. And initialize empty bag in the constructor.

public int V ();

Returns the number of vertices in this graph.

public int E ();

Returns the number of edges in this graph

private void validateVertex(int v);

Checks the vertex's value is proper for the limits. Throw an IllegalArgumentException unless { 0 <= v < V}

public void addEdge ();

It adds the undirected edge v-w to this graph. We increment number of E.

public int Degree();

It returns the degree of vertex

public Iterable<Integer>adj(int v);

It returns the vertices adjacent to vertex

public String toString();

It returns a string representation of this graph.

Queue class:

BFS uses Queue data structure to find the shortest path.

BreadthFirstPaths class:

BreadthFirstPaths (Graph G, int s)

We have Boolean marked array to keep the marked vertices, int edgeTo array to keep previous edge on the shortest path, int distTo array to keep number of edges shortest path. Also, we use dfs function in the constructor. Computes the shortest path between any one of the source vertices and every other vertex in graph.

private void bfs(Graph G int s);

It visited every vertex from source vertex and mark them. After it finds the shortest path.

It continues until there is no unmarked vertex. (traversal)

public int distTo (int v);

It returns the number of edges in a shortest path between the source vertex with parameter vertex.

public boolean hasPathTo(int v);

It checks is there a path between the source vertex and parameter vertex.

public Iterable<Integer>pathTo(int v);

It returns a shortest path between the source vertex with parameter vertex(destination)

Main class:

In the main class take the input from file and assign the source and destination values to integer and add the forbidden configuration to the forbiddenNumbersArr array.After that add the vertices and connect to edges between them in the graph. Except forbidden cases. Bond the related digits correctly.

- **Task2:**

I use the BreadthFirstPaths, Bag, Cycle classes to the implement a maze.

Cycle class can detect a cycle in the maze. Using this class I check there is a cycle or not.

Using BFS I try to find the shortest path from source vertex to source vertex.

Testing:

In the testing we are asked for reading input from file. I use File and Scanner for the reading input file. I use arrays to keep the inputs for both tasks. I tested the both tasks from given inputs. For the task1 I reach the correct results but in the task 2 I couldn't because I couldn't implement it correctly.

Final Assessments:

At the adding edge part to the graph was hard. I couldn't create a correct main class for the task2. The hardest part was creating a maze using graphs.

Thinking about implementation was abstract and it was the most challenging part of the homework.

I learned how to design maze from the input file and implement it to the graph

designing maze.

I learned Making new more resolvable problem from the problem. And solving some stufs like that.