Project Report on Graph Traversal

COT 6405 – Fall 2020

Ruchi Ninawe
U72686676
Guided by Dr Shaun Canavan, University of South Florida

Introduction:

The aim of this project is to model the maze into a directed graph and traverse it from start point to the end point using a graph traversal technique. Firstly, I have converted the given problem into an adjacency matrix and then used the BFS approach to reach from starting node to the ending node. The filename is *Traversal.py* which takes two arguments, an input file and name of any file to print the traversal path. Fig. 1 shows the path printed in the output file for the 8*8 matrix.

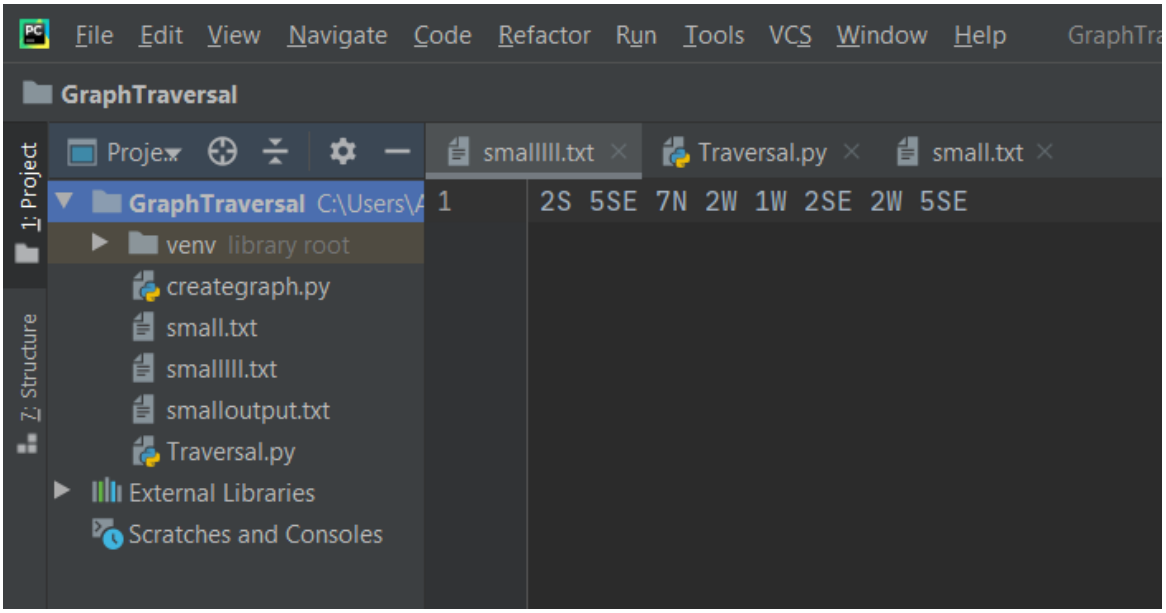

Fig. 1 Path printed in the output file

Implementation:

I have created an adjacency matrix from the given problem by the use of nested list and dictionary. The list *adjajencymatrix* holds a list of dictionaries to access the nodes of the graph. The dictionary stores color, direction, flagging for visited node and the edge as a tuple. Also I have implemented a queue using list to store the path of traversal and at the end I use the queue to perform BFS on the graph. The *Path*() prints the path in the output file by taking nodes from *adjajencymatrix*. The *CreateGraph()* creates a graph from the *adjajencymatrix* by accessing keys from the dictionary. The *addnodestomatrix()* adds nodes to the *adjajencymatrix* by determining the path using color and directions given in the nodes. The vertices are keys in the dictionary *dict* ad the edges are *connectededge* in the dictionary. The total time complexity to implement the given problem is **O(Vertices + Edges)**.

Puesdocode for implementation of BFS:

```
def BFS:
append first node to queuelement
while true:
Adjajencymatrix[first element]["visited"] = 1  #set the first node as visited
Call function addnodestomatrix()
Pop first node from queueelement
if (queueelement is empty):
break
```

Psuedocode for addnodestomatrix() implementation:

```
def addnodestomatrix():

    store color from dictionary key in nodeclr
    store index column from dictionary key in nodecol
    store index row from dictionary key in noderow
    store direction from dictionary key in nodedir
    i, j =0

  if (nodedir == directionofCurrentNode):
    i = noderow
      Loop till i >= 0
      if (adjajencymatrix[i][nodecol]["color"] != nodeclr and adjajencymatrix[i][nodecol]["visited"]
      == 0):
            mark it visited and append the connectededge tuple to queueelement
            mark it visited and append the connectededge tuple to queueelement
```

Psuedocode for creating Adjacency Matrix:

```
def Path():
    loop for elements in the graph:
        create listforgraph to store elements of graph
        create an empty list
        loop for all elements in listforgraph
            check is nodecolor is R or B
```

set mydict[‘color’] to listforgraph[i]
set mydict[‘visited’] to 0
set mydict[‘connectededges’] to (0,0)
i++
store row in empty list
append emptylist in adjajencymatrix