Assignment 1

Problem 1

Thought Process and Comments

For this problem we needed to create a recursive and stack based implementation of bfs and dfs where the graph to be searched is either implemented as a vertex list of an adjacency matrix. That's 2 implementations of 2 algorithms for 2 graph representations, meaning we need 8 algorithms. Each of these algorithms need to be run on two graph representations, G1 and G2, leading to a total of 16 experiments.

DFS was pretty straightforward, however BFS was more interesting to implement. First, I assumed that rather than "Perform BFS using stack" it actually meant to perform BFS using a Queue since that's what was gone over in class. Following suit, I couldn't find any obvious or clever implementation of BFS being done recursively, so I came up with an original recursive BFS algorithm that recurses "layer by layer" away from the starting node.

Results and Running

formated results directly from the command line for this problem can be found in /output/Problem1.txt.
This is generated on windows using: `python ./source/P1.py > ./output/Problem1.txt`

Answers

(For answers proving I get these results when varying graph format and implementation method see /output/Problem1.txt)
Where S is the start state and G is the goal state:

```
G1 DFS:
    States Expanded: ['S', 'd', 'b', 'a', 'c', 'f', 'r', 'e', 'h', 'p', 'q', 'G']
    Path Returned:   ['S', 'd', 'b', 'a', 'c', 'f', 'G']
G1 BFS:
    States Expanded: ['S', 'd', 'e', 'p', 'b', 'c', 'h', 'r', 'q', 'a', 'f', 'G']
    Path Returned:   ['S', 'd', 'c', 'f', 'G']
G2 DFS:
    States Expanded: ['S', 'd', 'b', 'a', 'c', 'e', 'h', 'p', 'q', 'r', 'f', 'G']
    Path Returned:   ['S', 'd', 'e', 'r', 'f', 'G']
G2 BFS:
    States Expanded: ['S', 'd', 'e', 'p', 'b', 'c', 'h', 'r', 'q', 'a', 'f', 'G']
    Path Returned:   ['S', 'e', 'r', 'f', 'G']
```

packages / libraries

1. `deque`: (from collections) used for BFS Queue Implementation
2. `numpy`: used to create the adjacency matrix in the shape I want

Problem 2

Thought Process and Comments

For this problem we needed to implement 1 algorithm for 2 graphs with 2 representations, leading to a total of 4 experiments.

The implementation of UFS was overall ok but I ran into issues using the algorithm provided in the slides as it mentioned nothing about a visited array to make sure you don't loop back. This was the biggest issue I had in the beginning, however, this was easy to see through and implement a fix for.

Results and Running

formated results directly from the command line for this problem can be found in /output/Problem2.txt.
This is generated on windows using: `python ./source/P1.py > ./output/Problem1.txt`

Answers

(For answers proving I get these results when varying graph format and implementation method see /output/Problem1.txt)
Where S is the start state and G is the goal state:

```
G3:
    States Expanded: ['S', 'd', 'e', 'p', 'h', 'q', 'b', 'c', 'a', 'r', 'f', 'G']
    Path Returned:   ['S', 'd', 'b', 'a', 'c', 'f', 'G']
G4:
    States Expanded: ['S', 'd', 'e', 'p', 'q', 'b', 'c', 'a', 'h', 'r', 'f', 'G']
    Path Returned:   ['S', 'd', 'e', 'r', 'f', 'G']
```

packages / libraries

1. `queue`: (Python 3.0+) used for UCS Priority Queue Implementation
2. `numpy`: used to create the adjacency matrix in the shape I want