COMP482 Project 1: Multiple Paths in a Graph via BFS

Due: 2355 June 13, 2023
Points: 30 points possible

Overview: For many applications of graphs, finding a path from a node x to a node y is essential, short paths are often better than long ones, and having multiple (edge) disjoint paths is strongly desired. For example, if a graph represents a computer network then computer x and y may need to communicate, the communication is better when it requires few hops, and since hardware is never 100% reliable having redundancy is a very nice property.

Details: You program will be provided with a file called input.txt which consists of one line that tells you the number of nodes n (nodes will be numbered $1, 2, 3, \ldots, n$ and n additional lines each with n entries which give the adjacency matrix.

You will use BFS to determine whether there is a path from node 1 to node n. If there is a path, your program will print it out and remove the edges used in the path. This process will repeat until there is no path from 1 to n and you print out the number of paths found. See the example input below.

Picky, but required specifications: Your project must:

- be submitted via canvas.
- consist of 1 or more dot-java files (no class files, zip files, input files, or other files should be submitted).
- have each file begin with a comment containing your name and the project number.
- not be placed into any package.
- have one file called Project1.java.
- compile with the command 'javac Project1.java'.
- run using the command 'java Project1'.
- accept input from a file called input.txt in the same directory as the java file(s) formatted precisely as described above.
- accomplishes the goal of the project. In other words, the output should be the correct answer formatted correctly.
- be submitted on time (early and multiple times is fine - do not be concerned if Canvas renames your file(s) by appending a hyphen and digit).

For each listed item you fail to follow, expect to lose at least 5 points. However, submitting via email will guarantee you a zero.

Examples:

If input.txt contains:

```
10
0 1 1 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1
1 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1
1 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 0
```

then the output would be

```
1 2 10
1 3 4 10
1 5 6 7 10
3 paths
```

because the first time running BFS you should find the path 1, 2, 10. Deleting these 2 edges and running BFS again you should find the path 1, 3, 4, 10. Deleting these 3 edges and running BFS again should find the path 1, 5, 6, 7, 10. Deleting these 4 edges and running BFS again results in n being unreachable.

If input.txt contains:

```
4
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
```

then the output would be:

```
1 2 4
1 3 4
2 paths
```

If input.txt contains:

```
15
0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
0 0 0 1 1 1 0 1 1 1 0 1 1 1 0
1 1 1 0 1 0 1 1 0 1 1 1 0 1 1
0 1 1 1 0 1 1 0 1 1 0 1 1 0 1
1 0 1 0 1 0 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 0 1 1 1 1 1 1 1 0
1 1 1 1 0 1 1 0 1 0 0 0 1 1 1
0 1 1 0 1 1 1 1 0 1 0 1 0 0 1
1 0 1 1 1 1 1 0 1 0 1 1 0 1 1
0 0 0 1 0 1 1 0 0 1 0 1 0 0 0
1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
0 1 1 0 1 0 1 1 0 0 0 1 0 1 0
1 0 1 1 0 0 1 1 0 1 0 1 1 0 1
1 0 0 1 1 0 0 1 1 1 0 1 0 1 0
```

then the output would be

```
1 15
1 4 15
1 8 15
1 10 15
1 12 15
1 14 15
1 2 5 15
1 6 9 15
8 paths
```