ELL781: Assignment 3

Submission deadline: **12 November, 23:59**

Maximum Marks: 8 (+1 extra credit for single-person attempts)

1 Problem statement

This assignment involves implementing two versions of the Ford-Fulkerson maximum flow algorithm: one using BFS to find augmenting paths (*i.e.*, the Edmonds-Karp algorithm), and the other using DFS.

2 Procedure

Here are the steps you should follow for this assignment:

1. Once again, you need a `GRAPH` data structure to represent the input flow network. You can use your data structure from the previous assignments, but can also make changes if required (here you will be dealing with directed, weighted graphs). The first step is to implement BFS and DFS; but a specific version of these has to be implemented, for the purpose of pathfinding between a given pair of nodes. So for this you should have two separate standalone functions, one each for BFS and DFS, and they should be implemented as follows:

    - Each function should take three inputs: a `GRAPH` instance, the index of the source node, and the index of the sink node;
    - Each function should run the search starting from the source node, and should stop the search as soon as the sink node is encountered;
    - The output of each function should be the sequenced list of nodes on the path discovered from source to sink, both inclusive (use a `LIST` data structure for this, *i.e.*, each function should return an instance of a `LIST`). If no path is found between source and sink, then an empty list should be returned.

2. Now implement the Ford-Fulkerson algorithm (following the pseudocode given in Section 26.2 of *CLRS*). This should have 4 inputs: the flow network (as a `GRAPH` instance), the source node index, the sink node index, and a flag denoting whether BFS or DFS is to be used for finding augmenting paths. Depending on the value of this flag, you should call the respective function implemented above for pathfinding. The output of your Ford-Fulkerson function should also be a `GRAPH` instance, where the edge weights represent the flows, *i.e.*, the weight of edge (u, v) in this output graph should be set to $f(u, v)$, where $f()$ gives the maximum flow discovered.

3. Write a top-level program for creating a `GRAPH` instance corresponding to a given flow network and running both the BFS and DFS versions of your Fold-Fulkerson function on it, and printing out the respective maximum flows returned (see output format below), along with the total value of the maximum flow and the runtime (in milliseconds) for each one. The input to this top-level program should be an $n \times n$ capacity matrix for a flow network, n being the number of vertices (assume the diagonal elements and elements corresponding to

unconnected vertices will be set to 0); this can be read in from a text file. The vertex corresponding to the first row/column in the matrix will be the source, and the vertex corresponding to the last row/column will be the sink. A sample input file is provided at `http://web.iitd.ac.in/~sumeet/input_flownet.txt`), which corresponds to the flow network depicted in Figure 26.2(a) of *CLRS*.

Logically, your code should consist of the following 6 components/modules (it is desirable to have each component in a separate file):

- An implementation of a data structure which implements the `GRAPH` ADT (you may re-use your code from Assignments 1/2, but may need to make some changes to it). **[0.5]**
- An implementation of a data structure which implements the `LIST` ADT (again, you may re-use your code from Assignment 1).
- An implementation of a specific version of BFS for pathfinding as indicated, as a function which takes a `GRAPH` instance, the source node, and the sink node as input and returns a `LIST` instance as output. **[2]**
- An implementation of a specific version of DFS for pathfinding as indicated, as a function which takes a `GRAPH` instance, the source node, and the sink node as input and returns a `LIST` instance as output. **[1.5]**
- An implementation of the Ford-Fulkerson algorithm, as a function which takes a `GRAPH` instance, the source node, the sink node, and the choice of BFS/DFS as input and returns another `GRAPH` instance as output. **[3]**
- A top-level program which is a function that takes as input the capacity matrix for a flow network in the given format, creates a `GRAPH` instance using it, calls the Ford-Fulkerson function on this instance with both the BFS and DFS options and gets the `GRAPH` instances returned by each of them (whilst also keeping track of the runtime for each call), and prints out these maximum flows in the given format along with the respective total flow values and runtimes. **[1]**

2.1 Output format

Your top-level program should print out the complete flow matrices for both the BFS and DFS versions, in the same format as the input file. Each flow matrix should be preceded by a line which gives the name of the pathfinding algorithm (BFS/DFS) that was used for it, the total value of the flow, and the runtime it took your code to generate the given maximum flow. For the sample input file linked to above, the output might look as follows:

```
BFS (maximum flow value:  23; runtime:  17ms)
0 12 11 0 0 0
0 0 0 12 0 0
0 0 0 0 11 0
0 0 0 0 0 19
0 0 0 7 0 4
0 0 0 0 0 0
DFS (maximum flow value:  23; runtime:  41ms)
0 16 7 0 0 0
0 0 4 12 0 0
0 0 0 0 11 0
0 0 0 0 0 19
0 0 0 7 0 4
0 0 0 0 0 0
```

3 What to submit

You need to submit all your code, properly commented and documented for easy readability. Please package everything into a single zip/tar/rar file. You should also include a README file which explains to the user exactly how they should use your code to generate the maximum flows for a given flow network.

Submission will be via Moodle (`https://moodle.iitd.ac.in/`); the exact submission procedure will be announced there in due course. The submission deadline for this assignment will be **12 November, 23:59**. Any late submissions will be penalised, and may not be evaluated at all, depending on the extent of delay.

4 Collaboration policy

You are free to discuss any aspect of the assignment with others; however, your code must be entirely written by you, without any copying from anywhere (other than your own code from the two preceding assignments). We will be using plagiarism-detection tools to ensure that this is the case, and any violations will lead to the loss of all marks for the assignment, plus further disciplinary action depending on the severity of the offence.