Documentation

Software:

Code Editor: Visual Studio Code (Version : 1.63.0)
OS: Windows 11 Home
Libraries: pandas, matplotlib

System Configuration:

Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
Installed RAM: 16.0 GB
System type: 64-bit operating system, x64-based processor

Code Structure:

Ford Fulkerson:

fordFulkerson.py contains the following functions:

1. getallvariables() - the main function which calls the FF(). The time to execute FF() is calculated in this function.
2. FF() - the Ford Fulkerson algorithm is implemented here.The flows are updated here.
3. search() - finds s-t path using BFS traversal.
4. searchDFS() - finds s-t path using DFS traversal.
5. graphConv() - converts the data in .txt input graph file to a suitable adjacency matrix.

Scaling Ford Fulkerson:

scalingFordFulkerson.py implements the following functionalities:

1. Calls the scalingFordFulkerson().
2. The execution time is measured here.

scalingFordFulkersonHelp.py contains the following functions:

1. scalingFordFulkerson() - the main function which implements the Scaling Ford Fulkerson algorithm. The flows and the delta value is updated over here.
2. scalingFordFulkersonHelper() - finds s-t path using BFS traversal. The traversal also makes sure that the path has a capacity >= delta value.

3. scalingFordFulkersonHelperDFS() - the same functionality as the above function but uses DFS traversal instead of BFS traversal.
4. setMax() - return the maximum capacity of the graph passed as an argument to the function.
5. largestPowerofTwo() - as the name suggests, it finds the largest power of two which is less than or equal to n.
6. graphConv() - converts the data in .txt input graph file to a suitable adjacency matrix.

Preflow Push:

preflow-push.py has the following functionalities and functions:

1. Execution time is measured in this file.
2. solve_max_flow(graph,s,t) - comes up with actual flow from a preflow edge by edge basis.
3. relabel(node) - updates the height of node to 1+ min_height of neighbors.
4. push_flow(node) - it will push the excess flow from node to neighboring node if the outgoing edges haven't exhausted the capacity and also height of neighboring node is exactly 1 less than height of node.
5. has_active_node(graph,s,t) - will return true if a node with excess > 0 exists in the graph otherwise false.
6. get_active_node(graph,s,t) - will return a node with excess > 0.
7. It has 3 classes, Graph, Node and Edge to represent the graphs, nodes and edges. All classes have helper functions.

Plot Visualization:

Visualization folder contains graph.py which visualizes the data.

Steps to run the code:

*****Download and install matplotlib and pandas.*

Ford Fulkerson BFS:

Steps:

1. Open Ford Fulkerson folder
2. Make sure the input graph .txt files are in the same folder.
3. Open this folder in Visual Studio Code

4. Click on fordFulkerson.py
5. Click on Run - > Run Without Debugging
6. Enter the name of the input graph .txt file in the terminal when the code prompts the user.

Ford Fulkerson DFS:

Steps:

1. Follow the above steps 1-4.
2. In FF(), change search() to searchDFS().
3. Click on Run - > Run Without Debugging.
4. Enter the name of the input graph .txt file in the terminal when the code prompts the user.

Scaling Ford Fulkerson BFS:

Steps:

1. Open Scaling Ford Fulkerson Folder.
2. Make sure the input graph .txt files are in the same folder.
3. Open this folder in Visual Studio Code.
4. Click on scalingFordFulkerson.py.
5. Click on Run - > Run Without Debugging
6. Enter the name of the input graph .txt file in the terminal when the code prompts the user.

Scaling Ford Fulkerson DFS:

Steps:

1. Follow the above steps 1-4.
2. Open scalingFordFulkersonHelp.py.
3. Change scalingFordFulkersonHelper() to scalingFordFulkersonHelperDFS() in scalingFordFulkerson().
4. Click on Run - > Run Without Debugging (scalingFordFulkerson.py).
5. Enter the name of the input graph .txt file in the terminal when the code prompts the user.

Preflow-Push:

Steps:

1. Open Preflow Push Folder.
2. Make sure the input graph .txt files are in the same folder.
3. Open this folder in Visual Studio Code.
4. Click on preflow-pus.py
5. Enter the filename of the input graph .txt file in the input_graphs list.
6. Click on Run - > Run Without Debugging.

Visualization:

Steps:

1. Manually record the execution time for the various graph input files.
2. Save it as .csv files (Sample files are in the 'Visualization Complete Graph Data ' subfolder.
3. Open the Visualization folder in Visual Studio Code.
4. Click on graph.py.
5. Make appropriate changes to the labels based on the input file.
6. Click on Run - > Run Without Debugging.

Input Graphs (Generated with Java starter code):

Bipartite Graphs - Parameters - #source nodes, #sink nodes, max probability, minimum capacity, maximum capacity:

b1.txt: 50,50,0.5,200,400
B2.txt: 100,100,0.5,200,400
B3.txt: 150,150,0.5,200,400
B4.txt: 200,200,0.5,200,400
b5.txt: 250,250,0.5,200,400

b12.txt: 300,300,0.5,100,200
b22.txt: 300,300,0.5,100,300
b32.txt: 300,300,0.5,100,400
b42.txt: 300,300,0.5,100,500
b52.txt: 300,300,0.5,100,600

Mesh Graphs - Parameters- #rows, #columns, capacity limit:

m1.txt: 20,30,200
m2.txt: 20,30,300

m3.txt: 20,30,400
m4.txt: 20,30,500
m5.txt: 20,30,600

m6.txt: 10,10,600
m7.txt: 20,10,600
m8.txt: 30,10,600
m9.txt: 40,10,600
m10.txt: 50,10,600

Random Graphs - Parameters - vertices, edges, min capacity, max capacity:

r-n-200.txt: 200,100,200,400
r-n-300.txt: 300,100,200,400
r-n-400.txt: 400,100,200,400
r-n-500.txt: 500,100,200,400
r-n-600.txt: 600,100,200,400

r-c-100.txt: 400,100,100,200
r-c-200.txt: 400,100,100,300
r-c-300.txt: 400,100,100,400
r-c-400.txt: 400,100,100,500
r-c-500.txt: 400,100,100,600

r-e-100.txt: 400,100,200,400
r-e-150.txt: 400,150,200,400
r-e-200.txt: 400,200,200,400
r-e-250.txt: 400,250,200,400
r-e-300.txt: 400,300,200,400

Code Output:

Max Flow:

b1.txt: 15159
b2.txt: 29782
b3.txt: 44951
b4.txt: 59987
b5.txt: 74520

b12.txt: 44495
b22.txt: 60492
b32.txt: 72914
b42.txt: 90632
b52.txt: 105858

m1.txt: 6000
m2.txt: 12000
m3.txt: 18000
m4.txt: 24000
m5.txt: 30000

m6.txt: 4000
m7.txt: 6000
m8.txt: 8000
m9.txt: 10000
m10.txt: 12000

r-n-200.txt: 30230
r-n-300.txt: 29653
r-n-400.txt: 28690
r-n-500.txt: 29218
r-n-600.txt: 29531

r-c-100.txt: 14800
r-c-200.txt: 19700
r-c-300.txt: 24838
r-c-400.txt: 29029
r-c-500.txt: 34415

r-e-100.txt: 29546
r-e-150.txt: 45646
r-e-200.txt: 59182
r-e-250.txt: 73335
r-e-300.txt: 88805