Code and Test Documentation File

There are mainly three folders
ClassfilesJavafilesand Makefile - contains class files , java files and makefile
TestAndOutputfiles - contains the generated graphs and the respective output results in four subfolders

In java files, tcss543.java is the main java file that calls all the three algorithms in order to calculate maximum flow of the network.
Ford Fulkerson Code Structure
Fordfulkerson.java s the main file that implements the Ford Fulkerson algorithm. It has following methods

- createVertexIndexMap(Graph G) method - assigns an index to each vertex for easy access of vertices in the path;
- createResidualGraph(Graph G, HashMap<String, Integer> vertexIndexMap) method - builds a residual graph as an adjacency matrix. Though it O(V^2) space, this makes it easy to access edges and edge capacities in O(1).
- fordFulkersonInit(Graph G) method - called to initialize the implementation of Ford Fulkerson.
- fordFulkersonCore(residual Graph, source, sink) method - used to Implement the ford fulkerson algorithm on the given residual graph (adjacency matrix) and return max flow. A standard breadth first search to check vertices if they are visited or not and to find the augment path in each iteration.

Scaling MaxFlow Code Structure
scalingmaxFlow.java file is the main file that implements the scaling max flow algorithm. It has three methods - scalingStarter(), scalingMaxFlow() and claculateDelta().

- scalingStarter()- takes in one input -SimpleGraph. This function finds out the source node s and sink node t for the input graph and gives a call to scalingMaxFlow()

- scalingMaxFlow() - takes in SimpleGraph, source node and sink node. It calculates the maximum capacity in the graph and also keeps track of all the vertices that are visited. To keep track of visited and unvisited vertices vertexInfo.java is used. To keep track and set the edge capacity and edge flow edgeInfo.java is used. It ensures that positive integer value flows. After nodes are marked as visited and unvisited, the function then calls the calculateDelta() function. There are two while loops in scalingMaxFlow() function which forms the heart of the algorithm. The outer loop run until the delta value is equal to or greater than 1. When the delta value is equal to 1, the residual graph obtained has the maximum flow. Each time a residual graph is returned with some flow value, the delta value is halved. The inner loop calculates the augmenting path as mentioned above.

- calculateDelta()- takes in the source node and calculate the delta value. Delta is the largest power of 2 which is equal to less than or maximum value of capacity out of source node. It traverses through the edges stored in linkedlist and incident to source s to compare the capacities and set the delta value to the largest capacity value found in the linkedlist which is greatest power of 2. While the delta value is greater than 1, the loop continues to retrieve residual graph. It generates the augmenting path using another java file augumentPath.java.

- augmentPath.java uses vertexInfo and edgeInfo to find the augment path and stores the residual graph i,e. SimpleGraph object.

Preflow Push Relabel Algorithm Code Structure

PreflowPushRelabel.java is the main file that has the required java code to implement the algorithm. In that java file,

- PreflowPushRelabel Class has the following code structure
- VertexData class represents the height and excess flow for each vertex
- EdgeData class represents the flow and capacity values
- The constructor of PreflowPushRelabel takes a SimpleGraph as arguement and calls the initialize method
- initialize() method perform takes the graph argument and performs the initialization step of the algorithm
- preflowPush() method is the starting point for the algorithm
- getVertexWithExcessFlow() returns the vertex other than source or sink with excess flow.
- getEdgeToPushFlow() return the edge where push is possible based on height constraints for a given vertex with excess flow.
- push() performs the push operation given the vertex and edge to push flow
- relabel() performs the relabel operation given the vertex.
- updateResidualGraph() updates the residual graph after every push to create reverse edges given the edge to be updated and value of flow pushed on that edge.

Test file Documents:

a) **Random Graphs:**

For Random Graphs We generated 70 files and executed the algorithms by taking these files as input.

INPUT Files :

The text files generated are in folder “Random”

Folder 1: “Random/VaryingCapacity/FixedVertices=100/”

It consist of 10 files which has different minimum capacity ranging from 5 to 90 .

Constant maximum capacity equal to 1000

fixed vertices = 100
fixed dense = 1000.

Folder 2: “Random/VaryingCapacity/FixedVertices=500/”
It consist of 10 files which has different minimum capacity ranging from 5 to 90 .
Constant maximum capacity equal to 1000
fixed vertices = 500
fixed dense = 1000.

Folder 3: “Random/VaryingCapacity/vertices100_more capacity/”
It consist of 10 files which has different minimum capacity ranging from 50 to 900 .
Constant maximum capacity equal to 1000
fixed vertices = 100
fixed dense = 1000.

Folder 4: “Random/VaryingDense/FixedVertices=100/”
It consist of 10 files which has different Dense ranging from 10 to 100
.
Constant maximum capacity equal to 100
Constant minimum capacity = 5
fixed vertices = 100

Folder 5: “Random/VaryingDense/FixedVertices=500/”
It consist of 10 files which has different Dense ranging from 100 to 1000 .
Constant maximum capacity equal to 100
Constant minimum capacity = 5
fixed vertices = 500

Folder 6: “Random/VaryingVertices/n5-100/”
It consist of 10 files which has different Vertices ranging from 5 to 100 .
Constant maximum capacity equal to 100
Constant minimum capacity = 5
fixed Dense = 1000

Folder 7: “Random/VaryingVertices/n20-500/”
It consist of 10 files which has different Vertices ranging from 20 to 500 .
Constant maximum capacity equal to 100
Constant minimum capacity = 5
fixed Dense = 1000

OUTPUT Files :
The output observed for each text files is recorded in excel sheet "Random/RandomGraphOutput.xlsx"

b) **Bipartite Graphs:**

Total Files: 56 .txt files are generated for bipartite graphs. These files are categorized into different folders and subfolders based on how these files are generated (Varying parameters like capacities ranges, number of vertices, probabilities).

Files location:The main folder is " **Charts and Excel for Bipartite/Bipartite"**

Inside **Bipartite** folder the following subfolders are present:

1. **/varying_capacity** (Varying minimum capacity)
   Maximum Capacity-1000, Probability-1, Capacities varied from 100 to 900
   a)/**high** (number of vertices=900 (m=400,n=500))
   9 files are present.
   b)/ **low** (number of vertices=300 (m=100,n=200))
   9 files are present.

2. /**Varying_n_m** (Varying number of vertices)
   Minimum Capacity-0, Maximum Capacity-30, Probability-1
   a)**/high** (number of vertices varied from 90 to 990)
   9 files are present.
   b) **/low** (number of vertices varied from 20 to 200))
   10 files are present.

3. /**varyingP** (Varying Probability)
   Minimum Capacity-0 Maximum Capacity-1000, Probability-1
   a)**/high** (number of vertices=900 (m=400,n=500),
   Probabilities varied from 0.1 to 1)
   10 files are present
   b) **/low** (number of vertices=300 (m=100,n=200)
   Probabilities varied from 0.2 to 1)
   9 files are present.

Output File: The analysis of these generated files are found in excel sheet present in "**Charts and Excel for Bipartite/Bipartite Graphs.xlsx**"

c) **Fixed Graphs:** For fixed degree graphs the folder named "**fixed degree input graph"** has four categories of graphs that were used to test. Below is the folder structure :
Fixed Degree Input Graph

1) **var_edges** (varying edge)

(a) fd_100v_e_20c_50c (contains input graphs which have 100 vertices, 20 minimum capacity and 50 maximum capacity)
(b) fd_100v_e_30c_600c (contains input graphs which have 100 vertices, 30 minimum capacity and 600 maximum capacity)

2) **var_max_cap** (varying maximum capacity)
(a) fd_100v_30e_90c_c (contains input graphs which have 100 vertices, 30 edges and 90 minimum capacity)
(b) fd_500v_30e_90c_c (contains input graphs which have 100 vertices, 30 edges and 90 minimum capacity)

3) **var_min_cap** (varying minimum capacity)
(a) fd_100v_30e_c_100c (contains input graphs which have 100 vertices, 30 edges and 100 maximum capacity)
(b) fd_500v_30e_c_100c (contains input graphs which have 500 vertices, 30 edges and 100 maximum capacity)

4) **var_vertices** (varying vertices)
(a) fd_v_10e_5c_500c (contains input graphs which have 10 edges, 5 minimum capacity and 500 maximum capacity)
(b) fd_v_20e_5c_500c (contains input graphs which have 20 edges, 5 minimum capacity and 500 maximum capacity)

A **"Fixed Degree_Read Me.txt "** file has the detailed folder structure. The analysis of all these test cases can be found in excel sheet name **"Fixed Degree Analysis.xlsx"**

d) Mesh Graphs:

For Random Graphs We generated 126 files and executed the algorithms by taking these files as input.

INPUT Files :
The text files generated are in folder "MeshGraphTestandOutputfiles"
Folder 1:
"TestAndOutputfiles\MeshGraphTestandOutputfiles\Graphgenerated\"

OUTPUT Files :
The output observed for each text files is recorded in excel sheet
"TestAndOutputfiles\MeshGraphTestandOutputfiles\Meshoutputresults.xlsx
\"