CODE STRUCTURE

By-
Swati Garg
Jessica Goel
Surabhi Agrawal

To conduct the empirical study for the network flow, we implemented the three network flow algorithms: Ford Fulkerson, Scaling of Ford Fulkerson and PreFlow-Push algorithms in Java.
For this we coded the class tcss543.java. This is the main class which executes all the three algorithms implementations and takes the average running time.

tcss543.java

Main():
Reads the input from the console and copies it into a file as the console input is the redirected input.
It then calls the function runAlgo() to execute each algorithm with the created input file.

Run Algo():
Performs the dummy run of the algorithm. Then executes the algorithm three times and takes the average of the times taken for the executions of the algorithms. This outputs the maxflow and the average time taken by each algorithm.

FordFulkerson.java

This is the implementation of the FordFulkerson algorithm. In this implementation we used adjacency matrix to store the graph.

Functions used:

1) **void loadFile(String filename)**: This function load the graph from the file.
2) **int maxFlow()**: This is the implementation of the FordFulkerson algorithm to calculate the max flow. In this the capacities are stored in the 2-D array such that the capacities[i][j] is the capacity of the edge between vertices i and j

3) **int findBottleneck(int[][] residualGraph, int[] path)**: This function finds the bottleneck of the path given in the residual graph. The function takes the residual graph and the augmenting path as an input.
4) **boolean findPath(int[][] residualGraph, int[] path)**: This function find whether there is any augmenting path from source to sink in the residual graph. The path is found by using breadth for search.

<u>FordFulkersonAlt.java:</u>

This is the alternative implementation of the FordFulkerson algorithm. In this implementation we used adjacency list to store the graph.

Function used:

class Vertex : This class represents the vertices of the graph

class Edge: This class represents the edges of the graph

1) **Vertex getOrAddVertex(HashMap<String, Vertex> vertices, String idVertex)**: This function checks whether the vertice exists, if it exists it gets the vertice id from the hashmap else it add the vertice id in the hashmap
2) **String getEdgeKey(String idVertexStart, String idVertexEnd)**: This function generates the key for storing the edge.
3) **Edge addEdge(String idVertexStart, String idVertexEnd, int capacity)**: This function check whether the key for the edge exists or not. If not it adds the edge in the hashmap
4) **addEdgeResidual(Edge edge)**: This function adds the edges in the residual graph
5) **run(String filename)**: This is the entry function to load the graph from the file provided and run the algorithm and return the maxflow.
6) **loadFile(String filename)**: The function loads the graph from the file
7) **int maxFlow()**: This is the implementation of the FordFulkerson algorithm to calculate the max flow. In this the capacities are stored in the 2-D array such that the capacities[i][j] is the capacity of the edge between vertices i and j
8) **int findBottleneck(int[][] residualGraph, int[] path)**: This function finds the bottleneck of the path given in the residual graph. The function takes the residual graph and the augmenting path as an input.

9) **boolean findPath(int[][] residualGraph, int[] path)**: This function find whether there is any augmenting path from source to sink in the residual graph. The path is found by using breadth for search.

ScalingMaxFlow.Java:

External classes used (Provided in the project folder along with graph generation codes):

1) Edge.java : Stores information of an edge. Its endpoints, the edge capacity and the flow on the edge.
2) Vertex.java : Stores information of a vertex. Its name, its incident edge list and a flag as to whether the vertex has been visited in a particular run of the BFS.
3) SimpleGraph.java : This class represents a graph. It stores the list of vertices and edges and provides functions that can be performed on them.
4) GraphInput.java: Reads and initializes a SimpleGraph.
5) KeyboardReader.java : This class reads strings and numbers from the keyboard.
6) InputLib.java : Manages input output.

Functions in the main class ScalingMaxFlow.java :

1) **int run(String filename)** : This function initializes the graph from the given filename. It creates an instance of SimpleGraph.java and calls the function LoadSimpleGraph(this.G, filename) in class GraphInput.java. This function also Initializes the residual graph and calls the function which returns the maxflow.
2) **void initializeGf() :** This function initializes the residual graph with forward and backward edges.
3) **int findMaxFlow()** : This function sets Delta for choosing good augmenting paths. It then finds the augmenting path in accordance with the delta. Then the graph is augmented, i.e flow values are updated in the original graph G on all edges. Then the residual graph Gf is updated.
4) **void setDelta()** : This function sets delta to highest power of 2, such that max capacity of any edge out of s is greater than delta.
5) **void findAugmentingPath()** : This function finds the augmenting path using BFS.
6) **void resetResidualGraph()** : This function gets the residual graph ready for finding the next augmenting path by marking all vertices as not visited and resetting the lists which store the vertices and edges involved in the augmenting path.

7) **void augment()** : This function finds the bottleneck in the augmenting path, updates the maxflow and updates the flow on all edges in original graph G.
8) **double findBottleneck()** : This function finds bottleneck in augmenting path.
9) **void updateResidualGraph() :** This function updates the forward and backward edges involved in the augmenting path, in the residual graph (In accordance with the bottleneck found)

PreFlowPushAlgo.java:

Classes:

1) The PreFlowPushAlgo() class : This class has the implementation of whole algorithm.
2) The Node() class: This class provide the implementation of vertices/node. It has variables Node; height, excess, issourceorsink identifier and neighbours stored in arraylist
3) The Edge() class: This class provide the implementation of an forward/backward edges between nodes, carrying the capacity. It contains a member function remaining(), to calculate the possible remaining flow on an edge: subtracting flow from capacity.

Member Functions:

1) **The runner()**: This has the implementation to load the input file and call the appropriate classes and functions to execute the code. The tcss543 file makes a call to runner() class to execute the pre-flow push algorithm.
2) **The getOrAddVertex()**: This function take the vertex as an argument. Thereafter it performs a check on, if we already have an Node implementation of that vertex, just to avoid the overwork and it maintains the hashmap of vertices for this purpose.
3) **The PreFlowPush()**: This takes the source/sink node, which I identified as s/t respectively. This function is responsible for executing the heart of the algorithm. It has implementation for relabel() and Push() functions.
4) **The clear()**: This functions flush out the hashmaps, so that it can be used for next run.
5) **The main()**: The main function is just for testing purposes. It accepts the input file as a parameter.

Output:

We executed the tcss543.java on various test files for 4 different graphs: Bipartite, Random, Mesh and Fixed Degree. The test results of the execution are as follows:
Times(**milliseconds**)

1. Bipartite Graph:

a. Varying number of nodes:

File Name	Nodes	Prob	Min cap	Max cap	Max flow	Preflow Push	Scaling	FFalt	FF
Bipartite-Nodes -10-10.txt	20	0.5	1	50	192	0.132	0.197	0.208	0.148
Bipartite-Nodes -20-10.txt	30	0.5	1	50	210	0.184	0.225	0.269	0.211
Bipartite-Nodes -70-70.txt	140	0.5	1	50	1638	2.774	7.127	25.1	15.276
Bipartite-Nodes -50-100.txt	150	0.5	1	50	1505	1.708	6.678	24.231	18.912
Bipartite-Nodes -100-100.txt	200	0.5	1	50	2218	3.996	21.257	62.379	41.759
Bipartite-Nodes -150-50.txt	200	0.5	1	50	1181	147.369	16.336	25.773	24.555
Bipartite-Nodes -120-100.txt	220	0.5	1	50	2615	71.53	32.856	81.672	57.893
Bipartite-Nodes -150-80.txt	230	0.5	1	50	1790	207.048	31.253	54.507	44.037
Bipartite-Nodes -150-100.txt	250	0.5	1	50	2237	254.148	45.44	77.503	72.504
Bipartite-Nodes -150-150.txt	300	0.5	1	50	3677	9.834	94.581	278.79	166.966

b. Varying number of edges by changing the probability

FileName	Nodes	Prob	Min cap	Max cap	Max flow	Preflow	Scaling	FFalt	FF
Bipartite-Edges Prob-0.1	200	0.1	1	50	1903	16.664	101.677	144.822	93.038
Bipartite-Edges Prob-0.2	200	0.2	1	50	2672	3.87	26.73	71.303	72.373
Bipartite-Edges Prob-0.3	200	0.3	1	50	2367	21.804	23.561	49.731	49.978
Bipartite-Edges Prob-0.4	200	0.4	1	50	2651	4.686	22.793	63.114	49.264
Bipartite-Edges Prob-0.5	200	0.5	1	50	2382	41.759	34.915	81.923	50.361
Bipartite-Edges Prob-0.6	200	0.6	1	50	2473	5.217	36.531	75.521	46.5
Bipartite-Edges Prob-0.7	200	0.7	1	50	2515	51.781	46.801	92.832	41.09
Bipartite-Edges Prob-0.8	200	0.8	1	50	2516	8.571	58.488	91.051	36.13
Bipartite-Edges Prob-0.9	200	0.9	1	50	2420	62.229	66.519	107.477	35.84
Bipartite-Edges Prob-1	200	1	1	50	2353	9.976	78.337	104.623	32.416

c. Varying the range of capacities

File Name	Nodes	Prob	Min cap	Max cap	Maxflow	Preflow	Scaling	FFalt	FF
Bipartite-minmax-1-50.txt	200	0.5	1	50	2429	3.767	20.614	77.087	41.968
Bipartite-minmax-1-100.txt	200	0.5	1	100	5025	3.984	21.791	71.098	44.617
Bipartite-minmax-50-100.txt	200	0.5	50	100	7464	3.763	24.577	56.379	31.741

Bipartite-minmax -50-150.txt	200	0.5	50	150	9620	3.639	26.414	54.862	33.347
Bipartite-minmax -100-1000.txt	200	0.5	100	1000	53541	27.2	33.73	62.333	42.822
Bipartite-minmax -100-2000.txt	200	0.5	100	2000	103964	3.815	23.088	66.604	43.084
Bipartite-minmax -1000-5000.txt	200	0.5	1000	5000	295232	3.854	24.955	59.741	37.241
Bipartite-minmax -1000-8000.txt	200	0.5	1000	8000	418222	3.834	24.443	59.705	37.509
Bipartite-minmax -5000-10000.txt	200	0.5	5000	10000	731998	43.144	36.82	49.24	32.953
Bipartite-minmax -5000-20000.txt	200	0.5	5000	20000	1270634	3.872	25.553	56.624	39.323

2. Mesh Graph

a. Varying number of nodes/Edges:

File Name	Nodes	Max cap	Maxflow	Preflow	Scaling	FFalt	FF
Mesh-rowcol -5-4.txt	20	50	250	0.055	0.106	0.096	0.097
Mesh-rowcol -10-10.txt	100	50	500	0.188	0.365	0.486	0.662
Mesh-rowcol -20-10.txt	200	50	1000	0.446	1.015	1.236	2.14
Mesh-rowcol -40-20.txt	800	50	2000	1.727	6.798	9.893	51.081
Mesh-rowcol -50-30.txt	1500	50	2500	3.138	15.036	21.07	204.808
Mesh-rowcol -50-50.txt	2500	50	2500	5.671	41.829	44.092	636.728

Mesh-rowcol-60-50.txt	3000	50	3000	7.007	65.137	86.712	1033.106
Mesh-rowcol-70-50.txt	3500	50	3500	8.362	88.517	110.465	1529.215
Mesh-rowcol-80-50.txt	4000	50	4000	10.171	107.955	139.148	2201.141
Mesh-rowcol-80-80.txt	6400	50	4000	16.032	213.282	248.974	6357.463

b. Varying maximum capacity:

File Name	Nodes	Maxcap	Max flow	Preflow	Scaling	FFalt	FF
Mesh-cap-50.txt	200	50	322	2.054	2.519	5.765	10.656
Mesh-capconst-50.txt	200	50	1000	0.788	1.406	1.384	2.213
Mesh-cap-100.txt	200	100	602	1.007	1.974	5.664	10.323
Mesh-capconst-100.txt	200	100	2000	0.713	1.016	1.253	3.265
Mesh-cap-500.txt	200	500	3572	1.472	2.511	6.437	12.016
Mesh-capconst-500.txt	200	500	10000	0.63	0.921	1.194	2.147
Mesh-cap-1000.txt	200	1000	6823	1.172	2.825	6.668	12.431
Mesh-capconst-1000.txt	200	1000	20000	0.699	1.027	1.352	2.448
Mesh-cap-10000.txt	200	10000	51541	2.883	2.534	4.9	10.29
Mesh-capconst-10000.txt	200	10000	200000	0.465	1.038	1.217	3.155

3. Fixed Degree Graph:

a. Varying number of nodes:

File Name	Nodes	Edges	Min-Max Capacity	Max flow	Preflow	Scaling	FFalt	FF
FD-Nodes-40.txt	40	30	1-50	829	0.713	1.723	2.457	1.486
FD-Nodes-50.txt	50	30	1-50	763	0.869	2.108	2.95	2.119
FD-Nodes-60.txt	60	30	1-50	696	1.034	3.088	4.059	3.191
FD-Nodes-70.txt	70	30	1-50	688	1.17	2.64	5.218	4.458
FD-Nodes-80.txt	80	30	1-50	793	4.691	6.487	6.784	4.456
FD-Nodes-90.txt	90	30	1-50	650	1.785	4.465	7.717	6.287
FD-Nodes-100.txt	100	30	1-50	761	5.677	6.307	7.44	6.626
FD-Nodes-120.txt	120	30	1-50	690	2.065	5.757	9.65	7.574
FD-Nodes-150.txt	150	30	1-50	694	12.882	10.79	13.237	10.85
FD-Nodes-160.txt	160	30	1-50	704	19.647	11.902	14.965	13.269

b. Varying number of edges leaving each vertex:

File Name	Nodes	Edges	Min-Max Capacity	Max flow	Preflow	Scaling	FFalt	FF
FD-Edges-10.txt	100	10	1-50	213	0.589	0.961	2.165	1.7
FD-Edges-20.txt	100	20	1-50	405	1.138	2.307	5.522	3.907

FD-Edges-30.txt	100	30	1-50	765	6.686	6.839	8.594	6.76
FD-Edges-40.txt	100	40	1-50	889	11.866	12.653	10.695	10.192
FD-Edges-50.txt	100	50	1-50	1134	3.717	8.1	13.943	11.12
FD-Edges-60.txt	100	60	1-50	1487	4.102	26.047	18.21	13.923
FD-Edges-70.txt	100	70	1-50	1398	3.942	14.54	22.269	13.753
FD-Edges-80.txt	100	80	1-50	1895	4.605	26.756	24.305	14.167
FD-Edges-90.txt	100	90	1-50	2133	5.49	29.547	26.075	13.181
FD-Edges-95.txt	100	95	1-50	2464	5.821	65.453	28.834	14.562

c. Varying range of capacities:

File Name	Nodes	Edges	Max Capacity	Maxflow	Preflow	Scaling	FFalt	FF
FD-cap-1-50.txt	100	30	1-50	728	1.847	5.022	8.115	6.608
FD-cap-1-100.txt	100	30	1-100	1533	8.95	6.704	9.27	6.096
FD-cap-50-100.txt	100	30	50-100	2178	6.813	6.623	7.59	5.991
FD-cap-50-150.txt	100	30	50-150	2697	1.765	6.111	7.366	5.958

FD-cap-100-1000.txt	100	30	100-1000	16655	1.754	5.416	7.309	6.549
FD-cap-100-2000.txt	100	30	100-2000	29295	1.858	4.618	8.129	6.73
FD-cap-1000-5000.txt	100	30	1000-5000	80364	6.916	10.962	7.575	6.042
FD-cap-1000-8000.txt	100	30	1000-8000	125781	8.165	9.821	7.649	6.722
FD-cap-5000-10000.txt	100	30	5000-10000	224916	2.021	6.63	7.23	5.932
FD-cap-5000-20000.txt	100	30	5000-20000	410381	2.429	5.805	9.251	7.629

4. Random Graph:

a. Varying number of nodes

File Name	Nodes	Density	Min-Max cap	Max flow	Preflow	Scaling	FFalt	FF
Random-Nodes-20.txt	20	30%	1-50	101	0.082	0.123	0.128	0.139
Random-Nodes-40.txt	40	30%	1-50	311	0.549	0.556	0.628	0.572
Random-Nodes-60.txt	60	30%	1-50	404	0.595	1.128	1.478	1.433
Random-Nodes-80.txt	80	30%	1-50	500	5.143	3.156	3.279	3.018
Random-Nodes-100.txt	100	30%	1-50	630	2.38	3.381	4.56	5.308

Random-Nodes -120.txt	120	30%	1-50	925	2.653	8.615	10.021	10.205
Random-Nodes -140.txt	140	30%	1-50	1006	3.931	10.331	13.299	15.961
Random-Nodes -160.txt	160	30%	1-50	1484	4.74	18.159	20.656	21.983
Random-Nodes -180.txt	180	30%	1-50	1060	58.921	43.477	22.838	24.362
Random-Nodes -200.txt	200	30%	1-50	1329	7.484	39.763	35.639	36.146

b. Varying number of edges by varying the density

File Name	Nodes	Density	Min-Max cap	Max flow	Preflow	Scaling	FFalt	FF
Random-Edges -10.txt	100	10%	1-50	204	2.424	1.406	1.395	1.7
Random-Edges -20.txt	100	20%	1-50	427	1.146	2.67	3.126	3.283
Random-Edges -30.txt	100	30%	1-50	714	1.723	4.485	5.227	5.627
Random-Edges -40.txt	100	40%	1-50	1116	2.36	9.64	10.221	9.563
Random-Edges -50.txt	100	50%	1-50	1169	2.973	12.586	11.781	11.772
Random-Edges -60.txt	100	60%	1-50	1640	12.955	30.587	15.416	13.571
Random-Edges -70.txt	100	70%	1-50	1787	22.124	47.102	17.191	12.802
Random-Edges -80.txt	100	80%	1-50	1860	18.863	70.19	22.016	13.56
Random-Edges -90.txt	100	90%	1-50	2193	5.402	95.293	25.955	13.752

Random-Edges-100.txt	100	100%	1-50	2261	7.014	81.059	25.963	12.791

c. Varying the range of capacities

File Name	Nodes	Density	Min-Max cap	Max flow	Preflow	Scaling	FFalt	FF
Random-cap-1-50.txt	100	30%	1-50	531	2.583	3.973	5.493	6.398
Random-cap-1-100.txt	100	30%	1-100	1439	7.084	6.61	4.877	6.006
Random-cap-50-100.txt	100	30%	50-100	1994	1.892	5.193	5.725	5.3
Random-cap-50-150.txt	100	30%	50-150	2954	9.796	8.092	7.968	5.779
Random-cap-100-1000.txt	100	30%	100-1000	16542	3.535	5.652	5.788	7.242
Random-cap-100-2000.txt	100	30%	100-2000	26331	17.011	7.804	5.142	6.938
Random-cap-1000-5000.txt	100	30%	1000-5000	84235	1.899	4.351	5.801	5.783
Random-cap-1000-8000.txt	100	30%	1000-8000	107116	1.83	4.601	5.032	5.856
Random-cap-5000-10000.txt	100	30%	5000-10000	207151	8.503	8.18	5.21	5.974
Random-cap-5000-20000.txt	100	30%	5000-20000	406858	8.023	8.301	5.687	6.308