Final Documentation

FLOW

Created by:
Hassan Jaber

1. List of modifications

- First of all, we changed the transformToNetwork(network) into a function that assign a source node to the network, and append all the rest of the node in a list of sink nodes.

  Pseudocode:

```
transformToNetwork(network):
foreach  node  u  in  network:
if degree(u) is minimum:
        source_node    =    u
foreach  node  v  in  network:
if v is not source_node:
        list_of_sink_nodes.append(v)
```

- We replaced the function isAugmentingPath(path) with a function that is called ford_fulkerson_alike(graph, s, t), that is slightly similar to Ford Fulkerson algorithm but with some tweaks,it takes a graph, source node s and sink node t as an input, then it create a residual graph that is a copy of the input graph but contains the updated capacities, and initialize every edges in the network with a capacity of 1, then while there is a augmenting path from source to sink, it update the capacities of the edges in the augmenting paths, then it updates the flow with 1, finally when there are no more augmenting paths, it returns the flow of the network from source s to sink t, which represent the minimum cut in one case of sink node:

Pseudocode:

```
ford_fulkerson_alike(graph, s ,t):
     flow = 0
     let residual
     foreach edge (u,v) in graph:
          residual[u][v] = 1 /*this represent the capacity of the edge*/
          residual[v][u] = 1
     while augmenting path exists:
          foreach edge (u,v) in augmenting path:
               residual[u][v] -= 1
               residual[v][u] += 1
          flow += 1

     return flow
```

- We replaced function FindMaxFlow() with FindEdgeConnectivity(graph), what it does it that it runs the above function to find the max flow in every case of sink nodes, then it append the max flow in a list_of_max_flows for each case finally, it returns the minimum of the list_of_max_flows that represent the edge connectivity of the graph:

  Pseudocode:

```
FindEdgeConnectivity(graph):
     foreach node t in list_of_sink_nodes:
          flow_of_network = ord_fulkerson_alike(graph, source_node, t)
          list_of_max_flows.append(flow_of_network)
     return min(list_of_max_flows)
```

- We changed the input description

  The input format now only consists of:
  1.A list of vertices: [1,2,3,4,…]
  2.A two- dimensional list of edges: [[1,2], [2,3],[3,4],…]

2. User Manual

2.1. Prepare input file

First of all it is mandatory to verify that the device (laptop or PC), that the user are using to run the app, has PYTHON installed on it, including numpy.

In case python is not install, go to https://www.python.org/downloads/ and download it, after that install it and be sure to check *(add python to PATH)*.
After finishing installing python, open your command prompt and write "*pip install numpy*" to download numpy.

After that, in order to run the application, an input file must be named and created.
In order to create and name an input file, it is required to run *Input_generator.py* (via command prompt, or some IDE), then the user has to enter firstly the name of the input file, secondly the number of vertices that he want, and lastly the percentage of the edges in this n vertices graph (graph density).

2.2. Algorithm execution

After creating the input file, in order to execute the algorithm and calculate the edge connectivity of the graph created, it is required to run *Flow.py*. That should be done using command line:

-Firstly, be sure that you need to write the command in the same directory that contains the Flow.py file, if not use cd (for example: cd C:\Users\HASSAN\Desktop\Semester1\Algorithm\Flow).

-Secondly, the command used to run the algorithm should contain the name of input file and the name of the output file that you need to use (for example: python Flow.py Input.txt Output.txt)

-Finally the result of the edge connectivity will be seen in the output file that you chose.

3. Tests description and conclusions

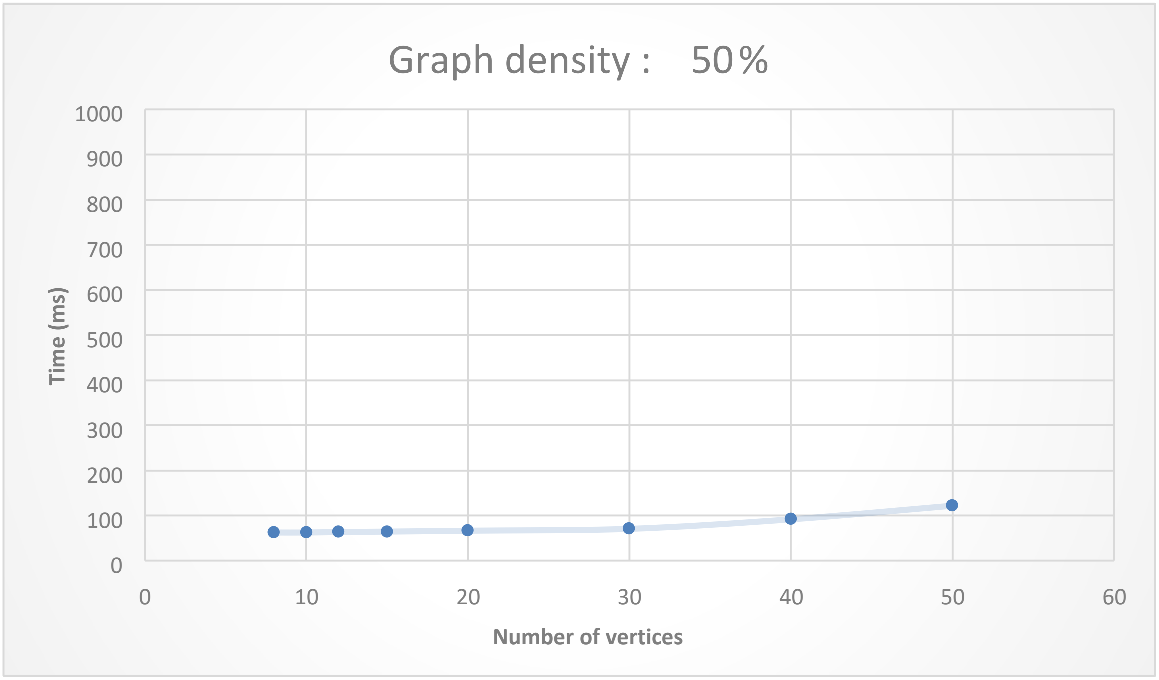
3.1. Tests description

In order to test the algorithm, we created multiple test files that contains different number of vertices and different percentage of edges, to compare the runtime of the application for each graph. For each number of vertices, we took 50% and 100% edges between those vertices (graph density).

First we tested the graphs for a density of 50% :

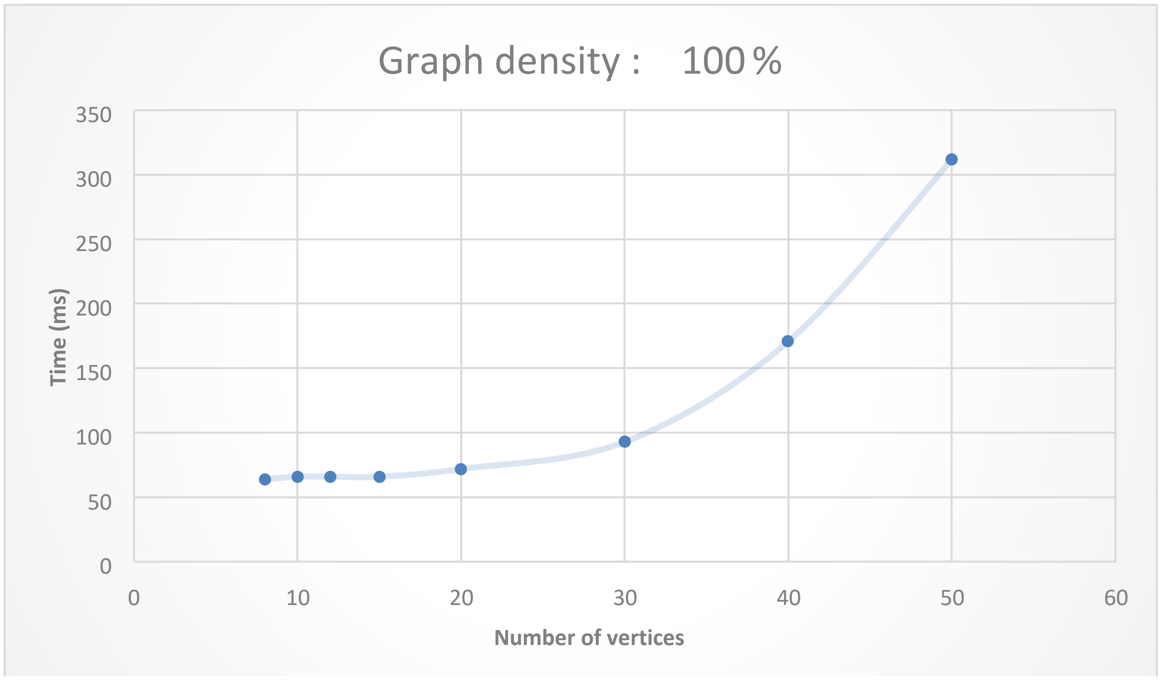
Number of vertices	***Number of edges***	***Maximum degree***	***Time (milliseconds)***
8	***14***	***6***	***63***
10	***22***	***7***	***63***
12	***33***	***9***	***64***
15	***52***	***9***	***65***

20	***95***	***15***	***67***
30	***217***	***20***	***71***
40	***390***	***25***	***92***
50	***612***	***34***	***122***



Second we tested the graphs for a density of 100% :

Number of vertices	***Number of edges***	***Maximum degree***	***Time (milliseconds)***
8	***28***	***7***	***64***
10	***45***	***9***	***66***
12	***66***	***11***	***66***
15	***105***	***14***	***66***
20	***190***	***19***	***72***
30	***435***	***29***	***93***
40	***780***	***39***	***171***
50	***1225***	***49***	***312***



3.2. Conclusion

Correctness

The algorithm runs correctly, and it output the correct result

Performance

After implementing , testing and analyzing our application, we can conclude that the time complexity of our algorithm is indeed O(|E|*Δ*|V|), where |E| represent the number of edges, Δ is maximum degree of the graph (this would happen in the worst case scenario, for example when graph density is 100%) and |V| represent the number of vertices.
We do admit that our algorithm is still slower than the original Ford-Fulkerson, but we shouldn't forget that in our case, we are not dealing with weighted directed graph, or an already built network, so we can't calculate the flow just one time from sink to source, instead we have to calculate it in every graph case for every vertex as a sink node beside the source that we chose, in order to calculate the correct edge connectivity of a undirected, unweighted graph.

4. Job partition

#	Tasks	Hassan Jaber	Rahul Raj
1	Solution Description	X	X
2	Pseudocode Description		X
3	Correctness and Time Complexity Analysis	X	X
4	Testing and Performance Analysis		X

5	Test and Input File Generator	X	
6	Algorithm Development and Implementation	X	
7	Final Report	X	