Algorithm Coursework Report

P.C.C. Peiris

2018116/ w1714892

Algorithmic strategy

- In this program, we should find the maximum possible flow in a flow network which the user created. I used ford Fulkerson algorithm to find the maximum flow. To this process, we need to find the path in the actual graph and residual graphs which is one of the outputs in the Ford Fulkerson algorithm. Therefore, we should pass the graphs to the breadth-first search algorithm to find paths in the graph.

The chosen data structure and its traversal towards the solution

- I used three two-dimensional arrays in this program. First one is to get the vertices in a graph and set capacities to each edge. Then in the Ford Fulkerson algorithm, I used two 2d arrays to define the residual graphs of the flow network and to present the paths with the distributed flows. In the breadth-first search algorithm, I used a linked array list as a queue. Each time value gets from the queue and passes it to the parent array if that node isn't visited node or it has any capacity. Then get the path and calculate bottleneck capacity. After the last residual graph finished, calculate all the flows and give the maximum possible flow.

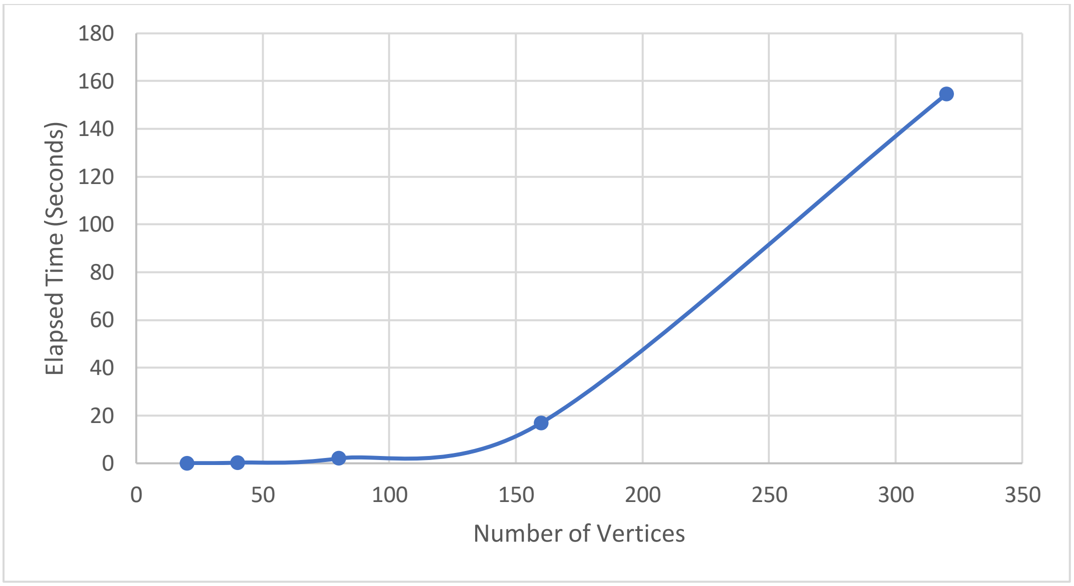
Pseudo-code in plain English

```
1.  Start
2.  Print “Please enter the number of vertices at least 6”
3.  Ask the user to enter a valid integer if invalid integer entered
4.  Print "-----------Adjacency Matrix--------------"
5.  Print "* Diagonal entries are zero in an Adjacency Matrix "
6.  Print "* Program will automatically set diagonal entries to zero "
7.  Print "* Let's assume this is a complete directed graph"
8.  Print "* Source node is 0 (s = 0)"
9.  Define a 2d array
10. Ask the user to enter capacity for each edge
11. Set diagonal capacities to zero
12. Check for valid inputs if invalid input entered
13. Ask the user to define the sink node
14. Check if the sink node equals to zero or greater than the input
15. If valid input, get the sink node
16. Ask the user to edit the graph
17. Get the final graph
18. Pass the graph into ford Fulkerson method
19. Define residualGraph 2d array
20. Define maximumFlowGraph 2d array
21. Set residual graph to zero
22. define the parent array which use to store parent nodes in the path
```

```
23. set max flow to zero
24. find that there is a path to the source from to the sink
25. Pass the residualGraph into bfs method
26. Create a Boolean visited array
27. set all vertices to false
28. create a linked list
29. add source node to the linked list
30. set source value to -1
31. for (int v = 0; v < vertices; v++)
32.     if (visited[v] == false && residualGraph[u][v] > 0)
33.             add the node to the queue linked list
34.             store path with parent nodes to each node
35.             Set visited nodes to true
36. ENDIF
37. ENDFOR
38. return true if we reached to the sink node
39. set path flow to the max value
40. for (v = sink; v != source; v = parent[v])
41.     get the minimum value of the given path flow to get the bottle neck capacity
42. ENDFOR
43. for (v = sink; v != source; v = parent[v])
44.     subtract the capacities in the path
45.     set the bottleneck capacities to reverse
46.     set the bottleneck capacities to reverse to get the path with the distributed flows
47. ENDFOR
48. Print residualGraph
49. update the path flow
50. Print the maxim flow path graph
51. Print the overall possible flow
52. End
```

Performance Analyze

Number of vertices	Elapsed time
20	0.047
40	0.27
80	2.042
160	16.937
320	154.51



Conclusions algorithmic performance

In this chart we can see when we are doubling the number of vertices, the elapsed time is increasing exponentially. This justifies the O(N^2) of the algorithm. Bfs algorithm used to find the augmented path and ford Fulkerson algorithm gives the maximum flow of the network. This program run for 5 times. We can see the elapsed time is increase each time. From 20 nodes 160 nodes, the graphs show a normal increase but from 160 to 320, it is a massive increase. So, the behaviour of the graph is polynomial. So, we say the Order of Growth of the algorithm is N^2