5COSC006W Algorithms Theory Design and practice

Coursework

Individual Report Submission

Name: Vihanga Malinda
Student ID: 2019495

UOW ID: W1790098

Course: B.Eng. in Software Engineering

Module Leader: Mr. Sudharshana Welihindha

Submission: Coursework Individual Report

Submission Date: 08/04/2021

IIt id: 2019495 Uow: w1790098
Name: S.V.M. Fernando

1. Choice of data structure and algorithm

I have chosen Ford Fulkerson algorithm with depth for search .For that algorithm we have to chose a structure which supports the node with the adjacency node. Here, we can use adjacency list or adjacency matrix for the data structure. The following table will elaborate the comparison between adjacency list and adjacency matrix.

V is vertex, E is Edge and O is big O notation.

	Adjacency List	**Adjacency Matrix**
Storage	Represent **O[V+E]**	Represent **O[V^2]**
Adding a vertex	Front node and the rear node are the two pointers in adjacency list. Therefore adding a vertex can be easily done compare to adjacency matrix.	To add a matrix we have to copy to another matrix. Because initial matrix size is V^2 and the new matrix has to be [V+1]^2. Complexity is high.
Removing a matrix	In the worst case time taken to search the vertex will be **O ([V]).** To transfer the edges takes a time around **O[E].** Therefore total time will be taken as **O [V+E]**.	In order to remove a matrix we have to decrease the size of the matrix. That means current size of the matrix is [V+1]^2 and we have to make that into the size of V^2. Complexity is **O ([V^2]).**
Querying	In order to find an edge it will take a time around **O[V]**	In order to find an edge it will take a time around **O [1].**

Overall, adjacency list perform comparatively well than adjacency matrix. And also lectures have given benchmarks to test our algorithm. Most of them have a large number of vertices in them [eg: ladder_9.txt]. Therefore to implement an adjacency matrix we have to have a large size in the adjacency matrix [V^2]. **Therefore, I have chosen adjacency list as the data structure for my algorithm**. And the complexity of adjacency list is **O [V+E]**{ similar to **O[V]**}.

2. Code explanation

As I mentioned earlier, I have use ford Fulkerson algorithm with depth for search. Following are the classes that I have choosen

- Main
- EdgeClass
- NetworkFlowInitializer
- DepthForSearch

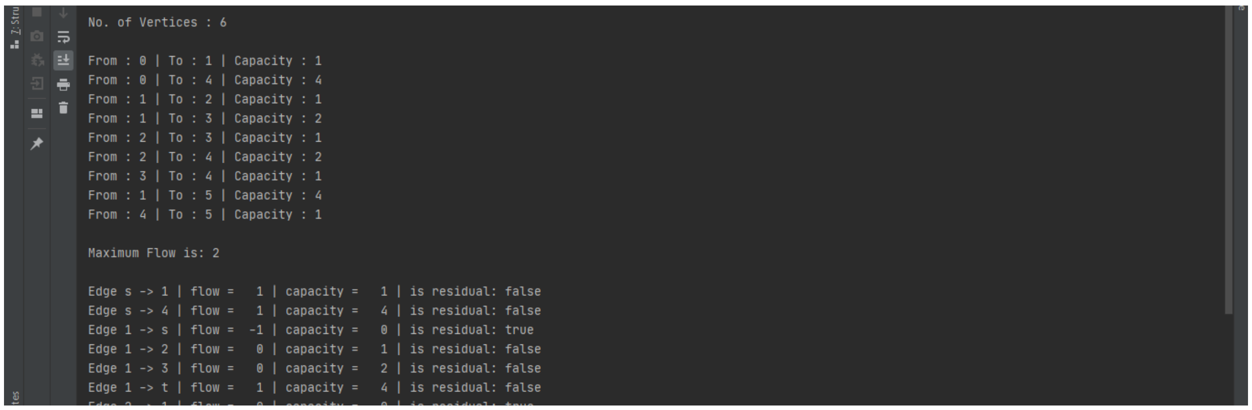
I have created EdgeClass so that I create an object on edge. Instance variables of this class are startingVertex, endingVertex, CAPACITY, every edge has a flow of network work and residual capacity which are represented by flowing and residualValue respectively. Argument constructor takes startingVertex, endingVertex, CAPACITY as the parameters. And I have initialized few public methods to check on the edge whether they are residual or not [residualCheck( )], to check on the remaining capacity of the edge [remainingCAPACITY()] and to check on the bottle neck value of the graph[clash(long bottleNeckValue)]. Bottle neck value makes a huge impact on the maximum flow of the net work flow. toString(int s, int t) method gives a better table view output on edges.

NetworkFlowInitializer is an abstract class which acts as a parent class on the algorithm. To avoid the overflow in the network, I have initialize the overFlowCapture given a value a little bit less than the infinity. Implemented some variables to support and display adjacency list ,they are node ,source and terminator. vistiedTag and the visited list have been initialized to prevent re-checking on the node that have already been visisted. MaxiumFlow and ranAlready was initialized to get the maximum flow through the network and to check whether the algorithm has been ran all ready respectively. Public methods have been initialized to adding the edges[addingEdge( )], to prevent no pointer exception[emptyFlowGraph( )] and to return the max flow respectively. Solver() method is an abstract method.

addingEdges() method in NetworkFlowInitializer throws exception when the capacity isles than or equall to zero. In here we declare two objects from the edgeClass where the residual edge has a capacity of zero.

DepthForSearch class is the child class of the NetworkFlowInitializer class. in here, solver method has been override to find the max flow by adding the all augmenting paths. dfs() method in this class provide the bottle neck value of the net work flow.

In the Main class I have used a parser to read the bench mark files. This will read nodes, source , terminating and the capacity. For this file path name has to be given. The fraction of the output [**bridge_1.txt**] is given bellow.



```
No. of Vertices : 6

From : 0 | To : 1 | Capacity : 1
From : 0 | To : 4 | Capacity : 4
From : 1 | To : 2 | Capacity : 1
From : 1 | To : 3 | Capacity : 2
From : 2 | To : 3 | Capacity : 1
From : 2 | To : 4 | Capacity : 2
From : 3 | To : 4 | Capacity : 1
From : 1 | To : 5 | Capacity : 4
From : 4 | To : 5 | Capacity : 1

Maximum Flow is: 2

Edge s -> 1 | flow =   1 | capacity =   1 | is residual: false
Edge s -> 4 | flow =   1 | capacity =   4 | is residual: false
Edge 1 -> s | flow =  -1 | capacity =   0 | is residual: true
Edge 1 -> 2 | flow =   0 | capacity =   1 | is residual: false
Edge 1 -> 3 | flow =   0 | capacity =   2 | is residual: false
Edge 1 -> t | flow =   1 | capacity =   4 | is residual: false
```

3. A performance analysis of your algorithmic design and implementation

Text file	**Time taken**
bridge_1	0.004 seconds
bridge_2	0.008 seconds
bridge_3	0.01 seconds
ladder_1	0.005 seconds
ladder_4	0.021 seconds
ladder_9	0.299 seconds

There is a fluctuation in time (mille seconds) taken to run the algorithm when the node are higher.