ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 336E

ANALYSIS OF ALGORITHMS
ASSIGNMENT 3

070170450 MUHAMMED ENES DENİZ

SPRING 2022

Contents

1 Presented Approach and Pseudocode

The presented problem is in the fundamental level to find the maximum flow in a weighted flow graph. Uniquely to this assignment, there exist possibilities to drop off the edges and as a result of this, there exists a set of possible maximum flow regarding the same source and sink.

After examining the fundamentals of the given assignment I have figured out the cruical steps as following: In first, I have to determine the possible subset of new graphs after the edge drop. To be more exact, let's imagine that I have totally 4 Vertices and 5 weighted edges between them as presented in the assignment file. And if 2 of the edges have a possibility to drop then it means that I have 2^n possible graphs to calculate maximum flow, in other words there exists 4 possible subset to examine. From the subset formula I know that the following fact, if the number of possible dropable index is n then it means that I have 2^n subsets. In next step, I have to find a way to represent this subsets and from the subsets I have to find which edges to be dropped and the possibility of that subset. I have searched online to find how to represent the subsets in this manner and find out that binary representation could be beneficial for my situation. The length of the binary will be determined by how many possible droppable edge e.g. if it 3, than it means my binary will vary between 000 to 111. And each bit of the binary should correspond the one of the droppable edge, e.g. 001 needs to mean that drop the X edge, 010 mean drop the Y edge, 011 mean drop the X and Y. I have find all of the subsets in binary and from looking bits I have figured out which edges will be dropped and calculated their probability in this manner.

Now I have obtained all of the possible graph configuration, for each of it I have to run the Ford-Fulkerson and find out the maximum flow of it. To roughly describe the Ford-Fulkerson algorithm: In first place assume that the flow is 0, look for a path that exists from source to sink calculate the paths minimum residual capacity and add it to the follow then update the graph according to this, repeat the last two process repedately until all of the valid paths had been discovered and maximum flow obtained by this approach. To traverse in the graph I have utilized the BFS algorithm.

So for each drop condition subset is generated and each of the subsets flow is calculated and the calculated flow and it's probability is mapped accordingly. In case of the maximum flow is equal, I have increased that flows probability. So at the end I could acquire a expected maximum flow. The expected maximum flow of the represented problem could be expressed as mathmetically following(assume, maximum flow and it's probability in flow map): $\sum_{n=0}^{n} Probflow[i] * flow[i]$

Let's examine the pseudo-codes of the explained procedures.

```
** In below code as could be seen subset creation operation has been handled.
** Subsets created according to the possible number of droppable elements.
** If we say that n as 2^possibleDropElement time complexity will be 2^n

subsetOperations()
{
        //create 2^possibleDropElement subset
        for i in range 2^2^possibleDropElement:
                subsetsBinary = decimalToBinary(i)

        for each element in subsetsBinary:
                toDrop = subsetsBinary[element]
                for each droppabledEdge in possibleDropList:
                        if(toDrop %10 == 1)
                                mark to drop possibleDropList[droppableEdge]
                                probability = probability * (probability to drop of droppableEdge)
                        else:
                                probability = probability * (1 - probability to drop of droppableEdge)
                        toDrop = toDrop / 10

                subsetsMap = droppableEdges //store marked droppableEdges of subset;
                subsetsMap = probability // store probability of subset;
}
maxFlowCalculator()
{
        for each element in subsetsMap:
                create residualGraph from mainGraph
                drop edges of residualGraph stored in subsetsMap
                maxFlow = (call of fordFulkerson for residualGraph)
                if maxFlow is not exists:
                        maxFlowProbabilityMap[maxFlow] = subsetsMap[probability]
                else:
                        maxFlowProbabilityMap[maxFlow].probability += subsetsMap[probability]

        print results
}
```

Figure 1: Subset Creation Pseudo-Code

```
**Consider E as edge and N as node**
** The below code has the have the time complexity of O(N^2 * E)**

Graph:: FordFulkerson()
{
        initiate listOfparents;
        create variables flowOfPath, pathTrack;
        while(BFS(listOfparents,flowOfPath, pathTrack) returns valid path exists)
        {
                for each u v in the found path:
                        graph[u][v] -= flowOfPath;  //simply reduce the capacity since it gives that much flow
                        graph[v][u] -= flowOfPath; // reverse the edges in the existing path

                increment maximumFlow by flowOfpath
        }

        return maximumFlow
}


Graph::BFS(vector<int> &listOfparent, int &flowOfPath, int &pathTrack)
{
  visited[numberOfNodes] = false; initiate Array to track visited node

  BFSqueue.push(source); push the source to the queue to start the BFS loop


 while BFSqueue is not empty
        {
           top of queue stored in examinedNode;
           pop queue

           for each element in the numberOfNodes:
               edgeWeight = graph[examineNode][element]
                if element not visited and edgeWeight > 0:
                        if reached to sink
                                for each node,edge in the found path:
                                        flowOfPath = min(flowOfPath, graph[node][edge]
                                return valid path exists
                        else
                                BFSqueue.push(element)
                                element parent set as examinedNode
                                create path and store based on element and element parent
                                set element as visited.
        }

        returns no valid path
}
```

Figure 2: Ford-Fulkerson Pseudo-Code

Consider Edge as E and Node as N from now on. If you examine the Subset creation Pseudo-Code figure you could acknowledge that the possible maximum droppable edge is equal to number of E so that's complexity of that part is equal to the 2^E because of creation of subset operations. In case of examination of the Ford-Fulkerson pseudo-code figure you will figure out that the found time complexity as $O(N^2 * E)$. I have adopted BFS while implementing Ford-Fulkerson method and it is a special method of Ford-Fulkerson which is also known as Edmonds-Karph algorithm. To figure out how I come with this solution first focus on the BFS side of the presented algorithm, as it could be seen the BFS will bound by $O(N^2)$ in case of using of the adjancecy matrix. The possible path length is bounded by the E it means that finding minimum residual capacity and updating the residual capacities will take as much as time as that so that time complexity will bound by $O(N^2 * E)$. If we combine this and come up with actual result the time complexity bounds of the given pseudocodes will be $O(2^E * N^2 * E)$ we have come up with this since we call/run the Ford-Fullkerson ($O(N^2 * E)$) algorithm as 2^E (maximum number of subsets) at maximum since for each possible graph we have calculate the maximum flow again.

2 How is network flow related to the graph centrality measures?

In first place let's define graph centrality. It is roughy mean that it is identify the more important/crucial nodes among the other nodes exists in the graph. To indicated centrality several measures had been introduced some of the following measures are: closeness, harmonic, betweness, eigenvector centrality.

Closeness centrality gives importance whether a node is closer to the other nodes or not. If the flow is following the shortest path closeness became more prominent measure since it identifies nodes that closer to other ones. In case of betweeness it roughly identifies the importance as how many time it occurs in the shortest path. In same manner flows who require fastest way to access could be related with this centrality measure. Eigenvector centrality simply gives utmost importance to the nodes who are close or surronded to the other important nodes. So, If a node has a higher importance it is neighbors also will have highest score. If a node has higher eigenvector centrality score flow through it becames more prominent.

3 Describe a potential real-life scenario where this sort of stochastic graph modelling and expected maximum flow can be useful.

The attempt to maximizing the total amount of flow out of source that is going to sink has numerous benefits and solutions to the varying real life problems. Airline scheduling, Circulation demand problem (avaiable at wikipedia) and even the algorithm itself is developed for maximum flow attainable from the railway from one city to another.

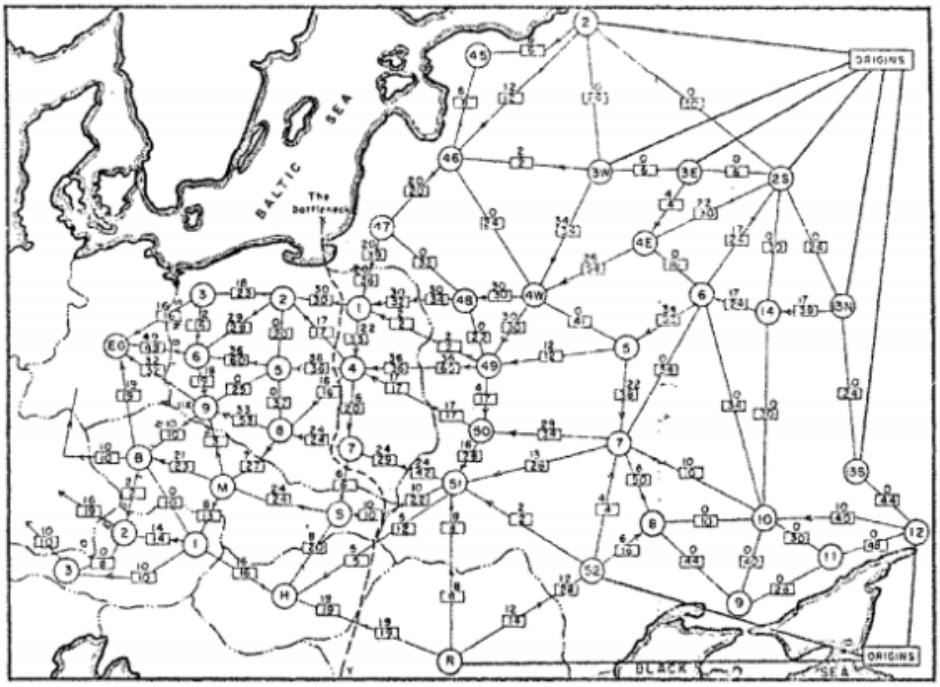

Figure 3: Illustrative Graph

As it could be seen from figure, finding the maximum flow attainable from a source to sink could be really messy and it needs to be formulated. Let's imagine that you are that the dawn of war and need to supply the front lines with the in need ammunation and soldier. Time is your enemy, and you have to send the supplies to there at maximum level. So you have to identify the residual capacities over the network and need to identify the bottleneck, by this you will understand by following the right path you will ensure the maximum amount of supplies will handled to your troops. Or in reverse thinking your enemy may also tries to disrupt your network and to choose which path to bomb and sneak out to damage you in maximum it needs to choose the path you would follow to attain the maximum flow over the network, by disrupting it, it will ensure no backup lines will coming after.

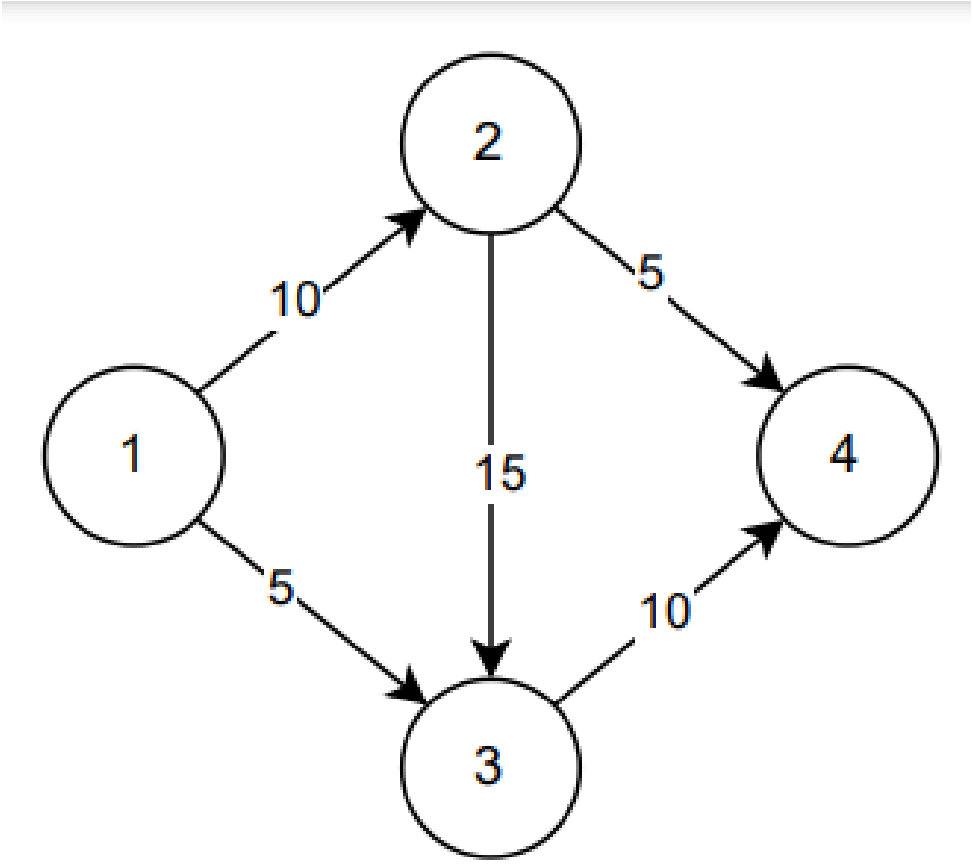

Figure 4: Simple Graph

Now narrow the problem and do not think yourself as general or leader. You are a plumber and need to brought maximum amount of water to the pipe and the pipe system has to following structure. Since your customer dont want to make huge repair you have to find the most suitable way to organize the way the water follow by having minor operations on the joints. So as in this case you have feed up the system with the maximum possible water. From the source joint 1 you could feed up it with totally 15 gallon and by minor fixes you could try to send this water as 10 gallon and 5 gallon to the other joints. After that with that same manner at joint 2 you have to make minor mixes and send 5 gallon of directly to source and remaining one back to the joint 3 to send it over there. Or in a reverse thinking procedure let's imagine that one of the possible connection between joints had been disrupted let's say the connection between 2 and 4 and connection between 3 to 4, and you have only chance to fix one of it, how would you choose which one? To maximize the follow of course you would go for 3 and 4 connection.

4 Does adding or removing edges always alter the maximum flow? If yes, please describe the reason. If not, state your objection with a counterexample.

No, I disaggre that adding or removing always alter the maximum flow. The edge weights and topology of the graph has utmost importance for the maximum flow. Adding or removing edges is not assured to change maximum flow.

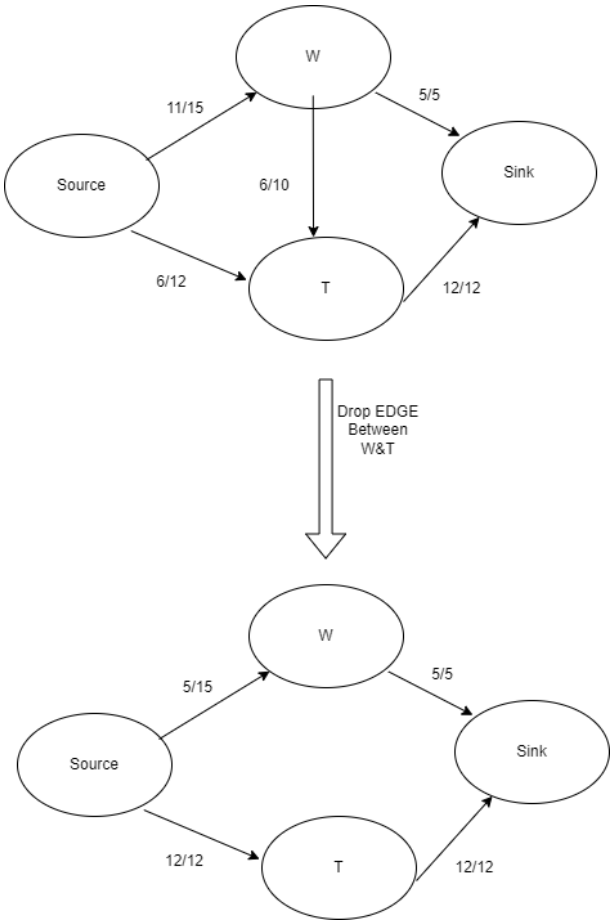

Figure 5: Counter Example

As it could be seen from the figure 5 I have disconnected W and T nodes, the maximum flow did not changed. Maximum flow is very directly related with the minimum cut concept and which links it to the bottleneck. If an added or removed edge did not change the minimum cut than it means that it will not also alter the maximum flow.