Assignment 5 – Bhanu Verma

903151012

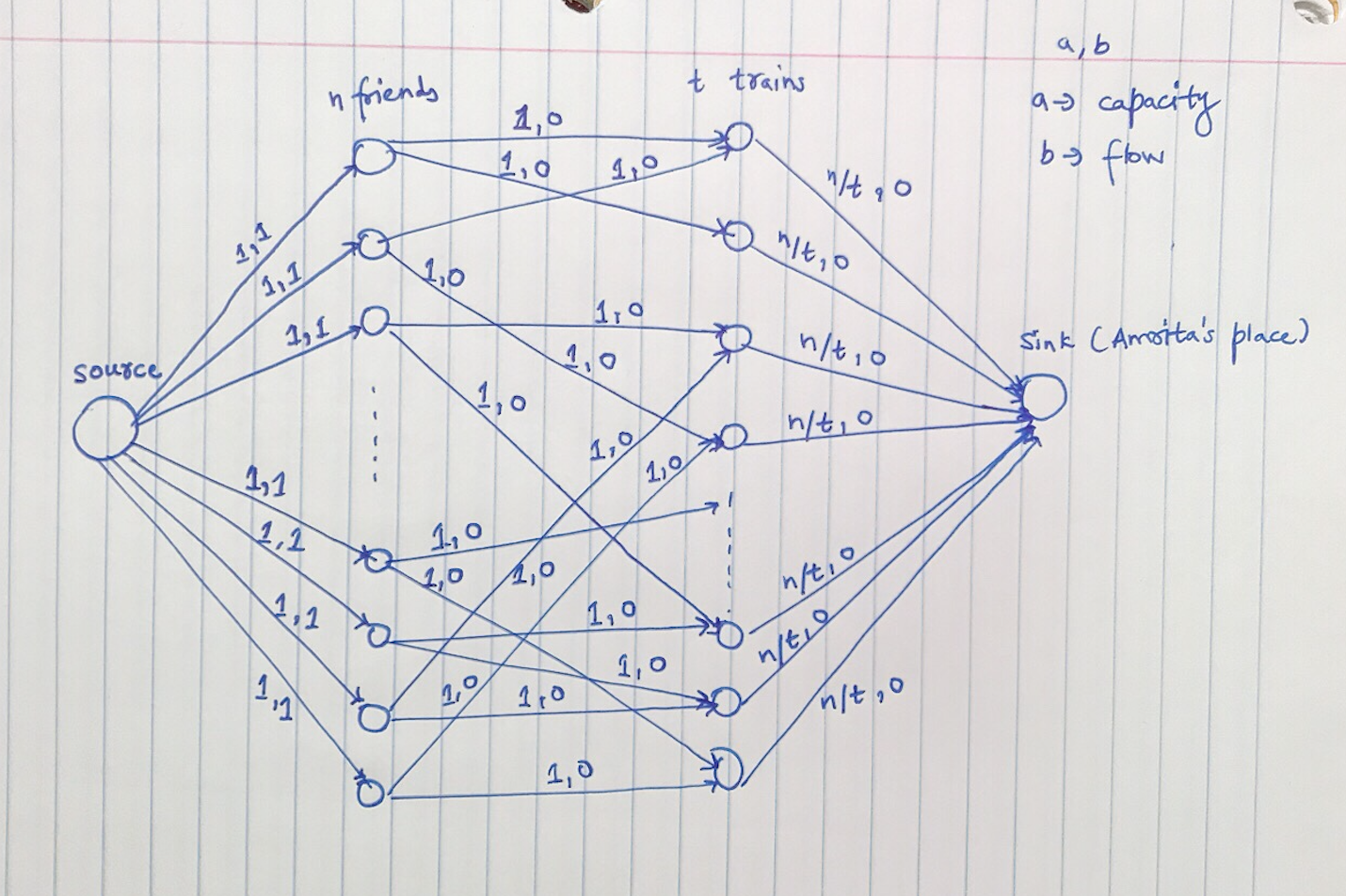
**Question 1 🡺 Oh ye, of little freight**

Given trains, friends and a list of trains for each person so that each person can take any of those trains, we need to find a polynomial-time algorithm that determines whether or not all the friends can reach Amrita’s place or not. Algorithm should divide all the friends **as evenly as possible** over trains so that the loss is minimized in case any of the trains are delayed.

This basically is an assignment problem where have to assign friends evenly to trains and it is given that we have to solve this problem using the knowledge of network flows. We’ll have to reduce the assignment problem to a network flow problem. This can be done using following steps:

* Take a source node.
* Connect it to each of the friends with an edge of capacity and flow equal to 1. So, we will have such arcs or edge coming out of source.
* For each friend, connect it to the given list of walkable train stations with an edge of capacity 1 and flow 0.
* Connect each of train stations to sink or Amrita’s place with an edge of capacity and flow 0. This ensures that we will have never have a flow value of more than .

Here is an image of the network that we’ll get after performing these steps:



After getting this network, we can run **Ford-Fulkerson** algorithm over this network. Eventually we’ll get a cut as the output of the algorithm that will give us the maximum flow of this graph.

* If maximum flow is less than then it means that all of the Amrita’s friends won’t be even to reach.
* If maximum flow is equal to then it means that all of Amrita’s friends will be able to reach the event.
* Note that maximum flow can not be more than . We have ensured this by making the capacity of each edge coming out of trains as .

Here is the Ford-Fulkerson algorithm:

f

Please note that, we have already calculated the flow values from source to friends. So we can assume that Ford-Fulkerson was already running and the above shown graph is an intermediate state of the graph. Or we can change those flow values to 0 and then run the Ford-Fulkerson from the beginning.

**Time Complexity**

Number of edges = \*n (at-most for a case when each person can take any of the trains)

Number of nodes =

Capacity =

Time complexity of Ford-Fulkerson algorithm is . This is a **polynomial time** complexity algorithm in terms of edges and nodes.

**Space Complexity**

Space complexity of Ford-Fulkerson algorithm is . This is a **linear time** space complexity algorithm in terms of edges and nodes.

**Question 2** 🡺 **Sweat in the Sweet Shop**

This question is basically asking us to find a deterministic algorithm for checking if the graph has a unique min-cut or not. This algorithm should run in polynomial time in terms of number of nodes (elves).

Here is the algorithm that would determine if the min-cut is unique or not:

Please note min cut in the above algorithm can be computed using Ford Fulkerson algorithm plus BFS (for finding where the min cut is).

Here is the formal proof of the above algorithm:

**Claim:** There exists an edge in C cut which when increased does not result in an increased max-flow Original cut C is not unique

🡺 Let e1, e2, e3, ……., ek be the edges in cut C. For each edge, increase the capacity by 1 and compute a new min cut C’. If for any edge, |C’| = |C| then it means that edge is not in the new min cut. If it were there in the new min cut, it would have resulted in increase in the capacity. This means that new min cut C’ has different set of edges but with same capacity. Hence, the original min cut is not unique.

🡸 Original min cut C is not unique which means that there is another min cut C’. We will at-least have an edge in C that will not be in C’, otherwise both the cuts would be same. Hence, if we increase the capacity of that edge by 1, capacity of cut C will increase by 1 whereas capacity of C’ will remain the same and will give us the max-flow. Hence, the original min cut is not unique.

Conclusively, we can say that graph will have a unique minimum cut if and only if |C| < |C’| for all edges in cut C.

**Time Complexity:**

Our algorithm computes min-cut at-most m+1 times, where m is the number of edges. Our total time complexity becomes where m is number of edges, n is number of nodes or elves and c is max capacity. It is same as . Hence, time complexity is linear in terms of number of nodes or elves.