CSE 548 Assignment 5

Divyansh Upreti (SBU ID: 112026646)

1 Prob 8

This problem can be solved as both flow problem and greedy problem and I have provided brief solutions for each below.

Flow solution:

a.) The following figure explains the solution by making a graph with a set of vertices as supply and another set of vertices as demand, a source and sink and then calculating the max flow of the graph. The algorithm will take time proportional to logarithmic of the total supply. There are edges between source node and supply nodes with values equal to given supplies and edges between demand nodes to sink with values equal to demands and edges between supply nodes and demand nodes if demand blood group can receive blood from supply blood group.

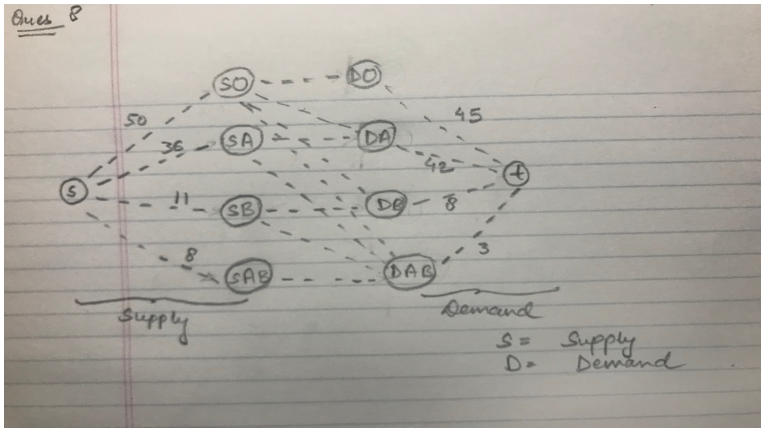

Figure 1: Flow of supply and demand

There is sufficient supply for demand edges if all are fulfilled by the supply nodes.

b.) In the present scenario, the supply is not able to fulfill demand for nodes O or A. We can see that there is a cut in the graph that corresponds to only 99 units and the demand is for 100 units. This can be told to hospital staff by saying that for blood group O or A, the cumulative demand is more than

cumulative supply hence supply for either of them can fall short.

Greedy solution:

a.) and b.) Given the supply and demand of each blood group known by hospital, we can use the following 'Greedy' algorithm to check if supply will suffice the present demand, presented to be understood by hospital staff.

Algorithm 1 supplyCheck(supply,demand)

1: begin
2: Suppose supply denotes the supply for sO, sA,sB, and sAB.
3: Suppose demand denotes the demand for DO, DA,DB, and DAB.
4: For each blood group, we will calculate the 'extraSupply' and 'missingSupply' by subtracting demand by supply, when supply is greater than demand and by subtracting supply by demand, when demand is greater than or equal to supply.
5: **for** each blood type **do**
6: if demand < supply then subtract demand from supply and update 'extraSupply' and 'missingSupply' for the blood type.
7: if demand > supply then we will find a valid blood type having extra supply that can work with this blood type and subtract the extraSupply for the helper blood group to fulfill missingSupply. If missingSupply is fulfilled after trying all or less valid blood groups, we stop fulfilling the current blood type and start fulfilling the other blood supplies. However, if blood supply is not fulfilled, we can report to the hospital staff
8: **end for**

2 Prob 9

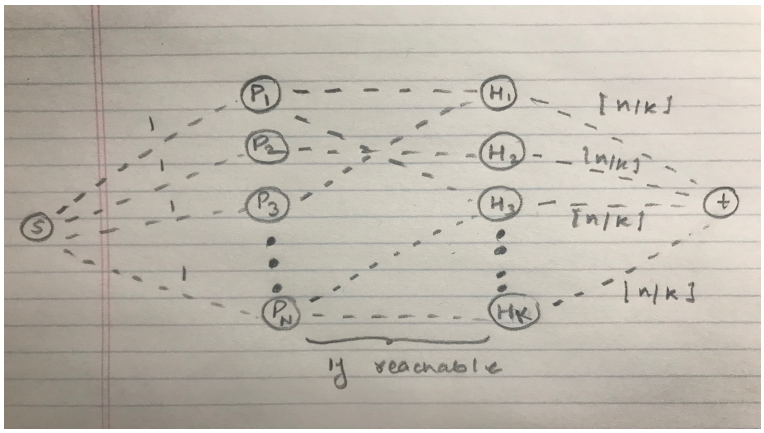

Figure 2: Example graph

The figure explains the solution by making a graph with a set of vertices as patients and another set of vertices as hospitals, a source and sink. The edges between source s and patients P_n, n=(1,2,...n) are given value of 1 and edges between patients and hospitals H_n, n=(1,2,...k) are given value of $\lceil n/k \rceil$. There are edges between patient nodes and hostpital nodes if patient can be driven to the hospital in half an hour. Total there will be O(n+k) nodes. We can send all patients to hospitals if there is a flow valued 'n' so that all patients can be sent to hospitals. We send 1 unit along a possible path and if all patients can be linked to some hospital, there exists a solution. Running time of algorithm will be the time to find maximum flow of the given graph.

3 Prob 17

We can solve this problem using logarithmic binary search method on each disjoint path of the graph (disjoint paths found using Ford Fulkerson algorithm). As the max flow of graph is k and each arc has capacity 1, flow can be decomposed into k disjoint paths. These paths can be found using Ford Fulkerson algorithm.
Attacker must have destroyed one arc on each of the paths. We can say this as all paths are exclusive. This also serves as the proof of correctness of this algorithm as to why we will process each arc separately.

So, we run logarithmic binary search algorithm on each path of graph, pinging the middle vertex on a path and if the vertex is found then trying the starting half of the path otherwise trying the later half of the path in similiar manner. So, we can find the removed edges in O($\log n$) pings as n is the length of each path.

After finding the removed edges, we can reconstruct the graph and then run any graph traversing algorithm to find which vertices are reachable and which vertices are not reachable. The running time of this algorithm is decided by logarithmic binary search and thus for k disjoint paths the algorithm takes only O(k*log(n)) pings which is our problem constraint.

4 Prob 18

a.) We can try max flow approach to solve this problem. We will consider two sets X and Y of vertices of the bipartite graph each corresponding to the an exclusive set of bipartite graph(property of bipartite graph) and add edges between these two sets of vertices are created with value 0 if the vertices are part of matching M and 1 otherwise. Next we will compute the max flow and if it comes out to be greater than matching(M)+k, then, we will return the matching M' otherwise report no solution. This solution works as we are ignoring the original matching in our flow and running algorithm for a new matching and if

its flow comes out to be larger than matching(M)+k, we can say we have found the solution.

b.) A required example is below with matching M (upper) and M'(lower). Edges of M do not form subset of M'.

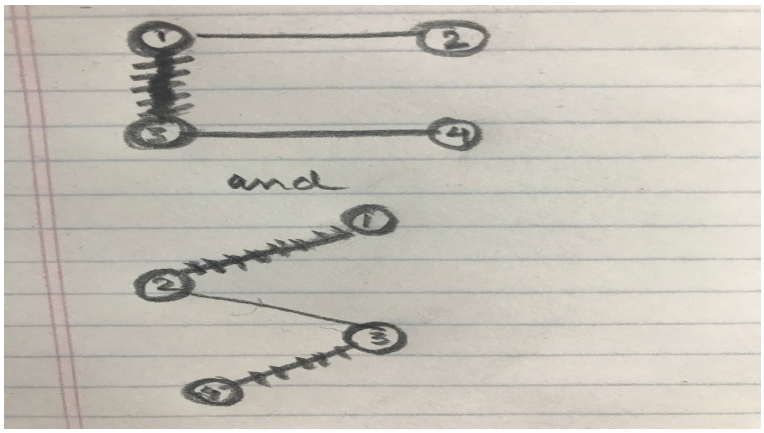

Figure 3: Graph

c.) K_1 will be equal to K_2. This follows directly from the definition of maximum matching. K_1 will be equivaluent to K_2 as every node y that is covered by any other matching is covered by this matching, making it largest matching.

5 Prob 22

a.) Example of a matrix which is not rearrangable :

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

b.) This problem can be solved using polynomial max flow algorithm by vizualising the below graph.

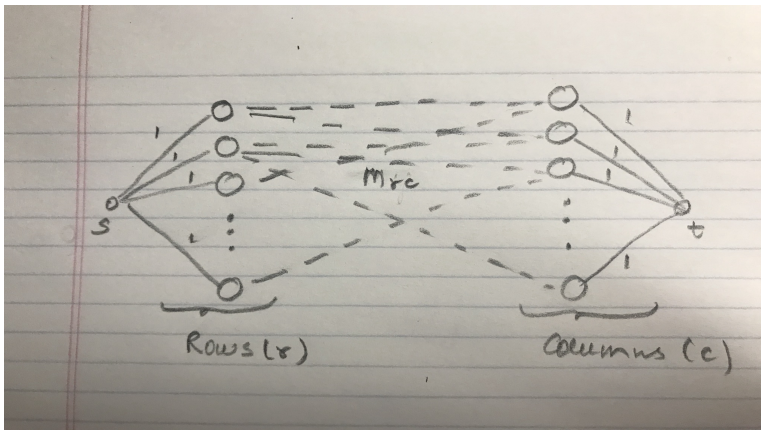

Figure 4: Graph

In the above graph, there is a set of nodes corresponding to rows of a square matrix and another set corresponding to column of a square matrix. There is an edge of value 1 between two nodes in two sets if M_{rc} is 1. We can say that a matrix is rearrangable if max flow of this network comes out to be equal to 'n' where n is the number of rows or columns in the square matrix. The time complexity will be equal to that of finding max flow of graph i.e. $O(n * n^2)$ (O(max-flow*edges)).