AOA Assignment 5

Sunny Shah - 112044068

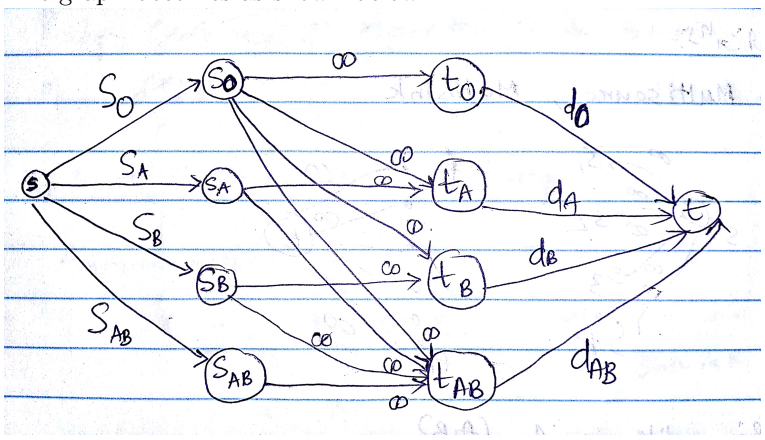
December 5, 2018

1) Page 415 problem #8

a: Give a polynomial-time algorithm to evaluate if the blood on hand would suffice for the projected need.

Solution:

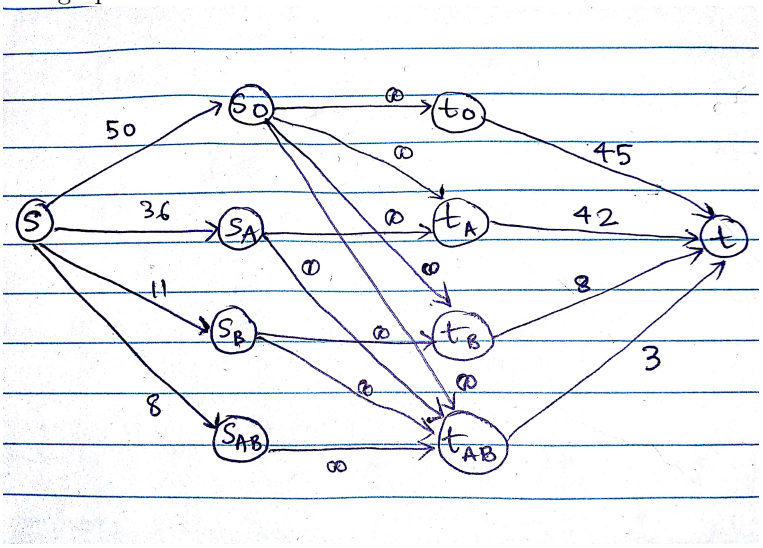
- If we denote the demand for O, A, B , AB as input to the sink (t), then the demand is satisfied if the capacity of each of the input edge to sink (t) is saturated.
- To convert this problem into a max-flow graph problem, we create the graph as follows:
  - Let $s_O, s_A, s_B, s_{AB} =>$ denote the supply of each blood type
  - Let $t_O, t_A, t_B, t_{AB} =>$ denote the node that takes in blood from valid sources to fulfill the demand
  - Create a source node 's' and connect it to each of the node s_O, s_A, s_B, s_{AB} using a directed edge
  - Create a target node 't' and connect each of t_O, t_A, t_B, t_{AB} to the target node with a directed edge
  - Since, blood type O can be used to in place of blood type O, A, B, AB. We create a directed edge from s_O to t_O, t_A, t_B, t_{AB}
  - Similarly add a directly edge from s_A to t_A, t_{AB}, s_B to t_B, t_{AB}, s_{AB} to t_AB
- Assign the weights as follows:
  - Assign S_O, S_A, S_B, S_{AB} i.e. the supply of blood type of O, A, B, AB to edge from S to s_O, s_A, s_B, s_{AB} respectively.
  - Assign d_O, d_A, d_B, d_{AB} i.e. the demand of given blood type to edge from t_O, t_A, t_B, t_{AB} to t repectively
  - Assign a weight of ∞ to each edge from s_i to t_j
- The graph becomes as shown below:



- We can now apply **Edmond-Karp** algorithm to the above graph to find the max-flow. If the max-flow is equal to $(d_O + d_A + d_B + d_{AB})$, then we can say that the demand is met, else the demand cannot be satisfied.

b)

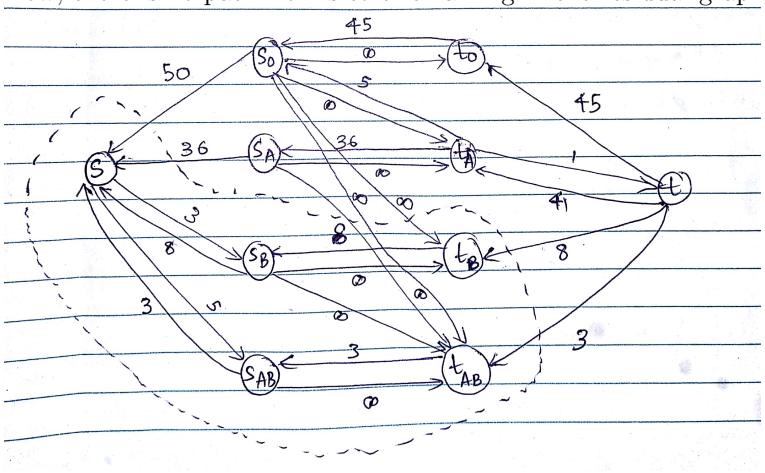
The graph will be as follows:



- The Edmonds-Karp algorithm will select the following paths from s to t in each of its iteration as follows:

  1) $s => s_O => t_O => t$ and Flow = 0 + 45 and $t_O => t$ is now saturated
  2) $s => s_A => t_A => t$ and Flow = 0 + 36 and $s => s_A$ is now saturated
  3) $s => s_B => t_B => t$ and Flow = 0 + 8 and $t_B => t$ is now saturated
  4) $s => s_{AB} => t_{AB} => t$ and Flow = 0 + 3 and $t_{AB} => t$ is now saturated
  5) $s => s_O => t_A => t$ and Flow = 0 + 5 and $s => s_O$ is now saturated

- Now, there is no path from s to t remaining in the residual graph which looks as follows:



- The max-flow will be 45+36+8+3+5 = 97 and the demand was 45+42+8+3 = 98
- Thus, the demand won't be satisfied
- As shown in the graph, the min-cut(S,T) can be given as:
  $S => (s, s_B, s_{AB}, t_B, t_{AB})$
  $T => (s_O, s_A, t_O, t_A, t)$

- We see that, only t_A cannot be fulfilled i.e. we had to match 42 patients of blood type A, but could only match 41 by taking 36 sample from blood type A and 5 from blood type O. If we take 1 more additional sample from blood type A, then we won't be able to match the 45 patients of type O.
- Thus, in any case, 1 patient of either blood-type O or A won't be able to match.

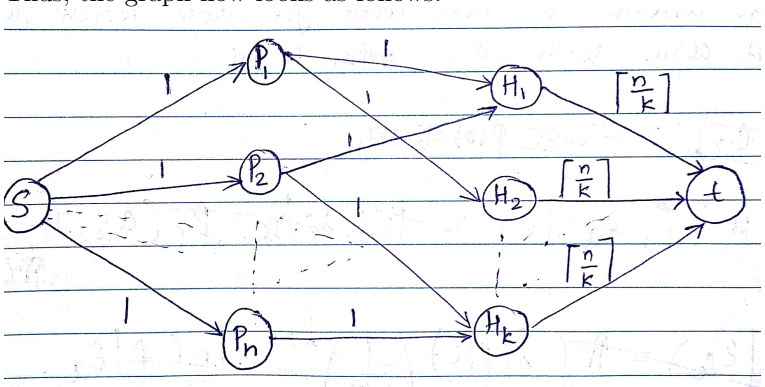
- In other words, the demand for O, A is 87 (i.e. 45+42) and combined supply of blood-type O and A is (50+36) = 86. So, there is no way for us to meet the blood necessity of 1 patient of either blood-type O or A.

2) Page 415 problem #9

Solution

We can convert this problem into a max-flow problem as follows:

- Denote each patient by a node P_i, where i => 1 to n
- Denote each hospital by a node H_i, where i => 1 to k
- Let us add a directed edge from P_i to H_j, if the hospital H_j is within half-hour's drive from the location of patient P_i
- To convert the graph into a max-flow problem, we create a source node 's', that has a directed edge going to each of patient node P_i
- We join each of the hospital node H_i to the sink node 't'
- Since, each patient can be taken to any one of the hospitals it is connected to, we assign capacity of 1 to each of the edge going from P_i to H_i
- Since, patient can only go to any one of the hospital, the incoming edge to each patient from source node s, should have a capacity of 1. Thus, each edge from s to P_i will be assigned a capacity of 1
- Since, each hospital should receive at most $\lceil \frac{n}{k} \rceil$ patients, we denote each edge from hospital H_i to t to have a capacity of denote each edge from Hospital H_i to t to have a capacity of $\lceil \frac{n}{k} \rceil$
- Thus, the graph now looks as follows:



- The task now is to find that whether the flow of n is possible for the graph. If yes, then, we can deduce that, all the patients can be assigned a hospital and each hospital received at most $\lceil \frac{n}{k} \rceil$ patients
- Thus, we make use of Edmonds-Karp algorithm to solve the problem
- if tge max-flow is nm then a balanced allocation of patients is possible, else it is not possible.

Complexity:

- Edmonds-Karp can be solved in $O(nm^2)$, where n is the number of vertices in the graph and m is number of edges in the graph.
- For our graph, n = (1+n+k+1) = (2+n+k) and m = (n*k + n + k) in worst case when all hospitals are adjacent to each of the patient
- Thus, the complexity of our algorithm is $O((2+n+k)(n*k+n+k)^2)$

3) Page 415 problem #17

- We know that the attacker's removal of k edges, completely removed the connection between the source and target nodes
- Also, in the initial working before the hack, the max flow of the network was k
- Thus, there were k disjoint paths from s to t
- Thus, each of the edge e that was deleted must be a part of any one of the disjoint path from s to t. Also, this k edges are such that no new disjoint path could be formulated using any prefix paths of the unique disjoint paths we had in the actual network
  i.e. If e(u,v) was deleted, then there was no path from s to u and u to t, such that it does not pass through v
- Thus, if we find the k disjoint paths in our original network, it is guaranteed that the k edges deleted will always be part of the disjoint path. If not, then the max-flow after the edge deletion cannot be 0
- Thus, our task is to find the edge in each of the k disjoint paths, that might be deleted
- To find such an edge in one of the disjoint path, we can use binary search technique as follows:

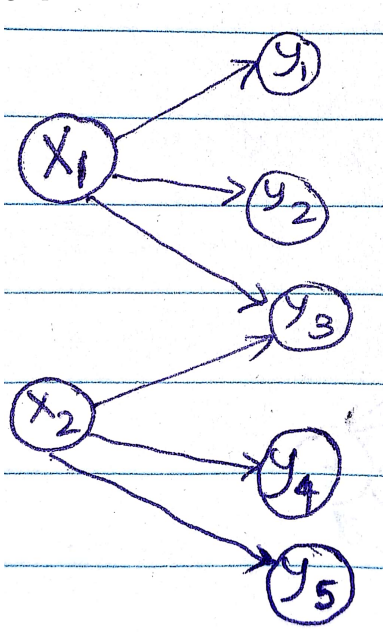
```
Let s,v_1,v_2 .....v_n,t be the order of the nodes in one of the disjoint path.
Let s => v_0 and t => v_n+1
Let low = 0, high = n+1

while( low<high ):
    let mid = (low + high)/2
    if ping(v_mid) == true: # Node v_mid is connected to s
        # Thus, the nodes to right of this are not connected
        low = mid + 1
    else:
        high = mid

print( "Edge ",v_(low-1) ," -> ", v_low, " was disconnected by the hacker" )
return E(v_(low-1), v_low)
```

- Thus, finding the missing edge in a disjoint path now takes O(log n)
- We apply the same binary search for each of the disjoint path.
- Thus, the overall complexity of our algorithm is O(klog n)

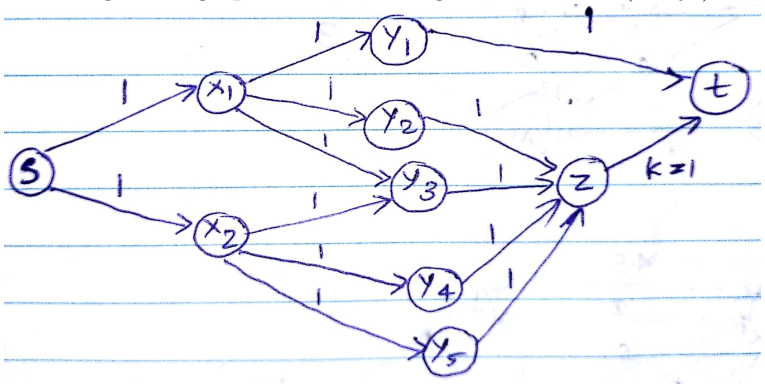
4) Page 415 problem #18

a:

- M' should have k more edges and should also cover every node $y \in Y$, that were covered in M
- Thus, to solve this, we need to formulate the above constraints using a max-flow graph. Consider the below graph:



- The matching M that was provided to us is M => (x_1, y_1) and k=1 is given
- We can create a directed graph using the given matching M and the value of k as follows:
  - For each y_i that were matched in M, we add a directed edge from Y_i to t and assign this edge a capacity of 1
  - We now connect the rest of the nodes Y_i not covered in M to another intermediate node Z, which is then connected to t with a directed edge with capacity of k. This will ensure that, out of the remaining nodes that were not matched in M, can we select k more Y_i
- Following is the graph for the above given G, M => (x_1, y_1) and k=1:



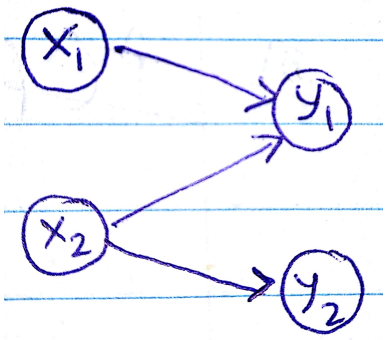
- Thus, in the above graph, if we ensure that max-flow is (number-of-matching-in-M + k), then, M' can be found

- Thus, we create a graph as described above and accordingly check if the new max-flow is (number-of-matching-in-M + k)

b:

Consider the following instance of G, M, K

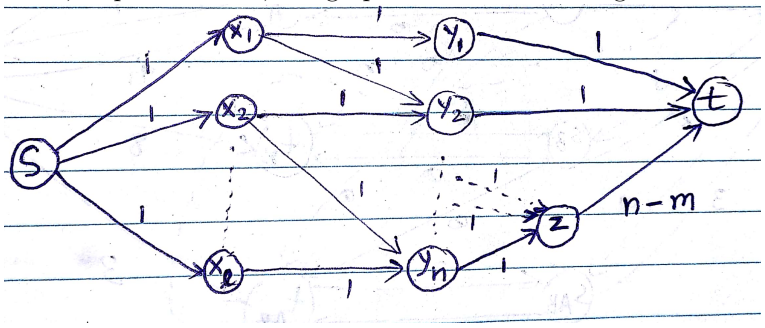
- Following is the original graph:



- Let the initial given matching be M = (x_2, y_1) and k = 1
- Since, M' should cover y_1, and an additional y as k = 1, it will have to select the edge (x_1, y_2) and (x_2, y_2). Thus, M' = (x_1, y_2),(x_2, y_2)

c)

- Using the idea mentioned in the section a, we can create a directed graph to find the largest matching (K1) by setting k = n - (count-of-nodes-covered-in-M) and solve as described in section a
- Let m => size of matching in given matching M and n be the number of $y_i s$
- Then, as per section a, our graph will look something as below:



- In the above graph, y_1, y_2 were covered in M
- The flow remains the same for both the graphs as we can see that the total capacity input to sink is n and each inner and outer edge to y_i is of capacity 1
- Thus, only the edge to y_1 and y_2 considered in maximum matching, might change, but the total flow can always be brought to the actual max-flow of the graph selecting the (max-flow - matching-of-M) number of Y from the remaining Y_is
- Therefore the flow for both the above graph is same and thus we can assert that K1 = K2

5) Page 415 problem #22

a)

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

The above matrix is not rearrangeable

b)

- This can be solved using a bipartite maximum matching algorithm
- The idea is to denote the given matrix in the form of a bipartite graph
- Main ideo is that any of the 1 in each row will only contribute to fill one element of the diagonal
- Thus, we need to find an alignment for each row that tells which column to choose for that row. i.e. for each row i, we find an column index j in 1 to n, where $M_{i,j} = 1$, and this is the 1 that will fill diagonal element on that row i.e. cell with coordinates (i,i)
- Once we choose j as the index for row i, we cannot use j as the index of alignment for any other row
- Thus, we denote the matrix as a Graph G with:
  - nodes R_i where i => 1 to n, and each R_i denotes a row in the matrix
  - nodes C_i where i => 1 to n, and each C_i denotes a column in the matrix
  - We connect R_i to C_j if $M_{i,j} = 1$
- Now our task is to find the maximum matching of the Graph G described above.
- All the column nodes C_i that are selected in the maximum matching denotes that the cell $M_{i,i}$ will be filled with 1
- If the maximum matching selects all C_i, then we can assure that all the diagonal elements will be filled with 1
- Thus, if the maximum matching of the graph is equal to n, then the matrix is rearrangeable, else it is not rearrangeable
- Now, the maximum matching in the bipartite graph can be found using Edmonds-Karps algorithm by adding the source s and sink t nodes to the Graph G