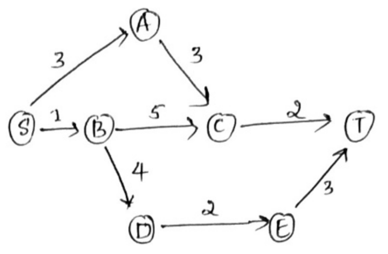
CSCI570 - HW4

1.

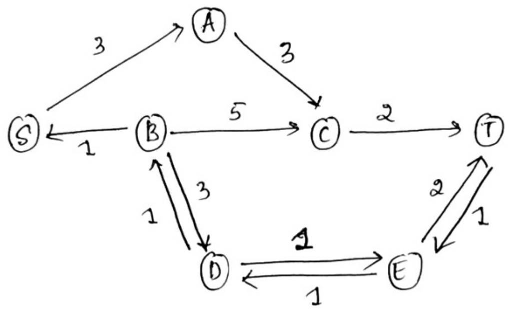
Question 1:



(a) (i) Path : SBDET

we have $C_f(P) = 1$

$f(u,v) = 1$



- Now, we have G_f - residual graph after our first iteration.
- Edge SB is now saturated.

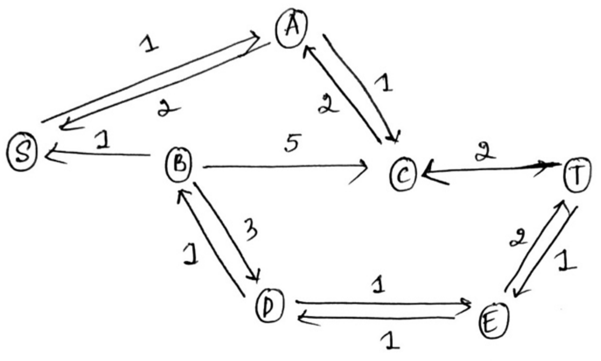
(ii)

Path: <u>S A CT</u>

We are taking the augmenting path S A C T.

with $C_f(P) = 2$

$f(u,v) = 3$

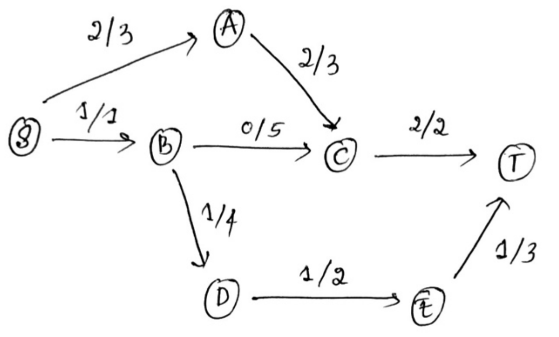


- G_f – Residual graph after our second iteration. Edge C–T is now has been saturated. We are left with no augmenting path P in the residual graph anymore. Thus, the graph is now disconnected.
- We have a max flow $f(u,v) = 3$ units

(b)

We have got the maxflow for the given network flow $f(u,v) = 3$ units.

Our network flow graph with flow/capacity on each edge after running the Ford-Fulkerson algorithm is:



Starting from S, nodes reached are S, A, C.
Mincut obtained is:

Partition A: { S, A, C }
Partion B: { B, D, E, T }

2.

We start by solving the problem as a bipartite graph problem. In this graph, one partition is a set of vertices t_1, t_2, t_3,....,t_n representing all n traders. Another partition is a set of currencies, c_1, c_2, c_3,....,c_m representing m currencies. Trader t_k is willing to trade as much as

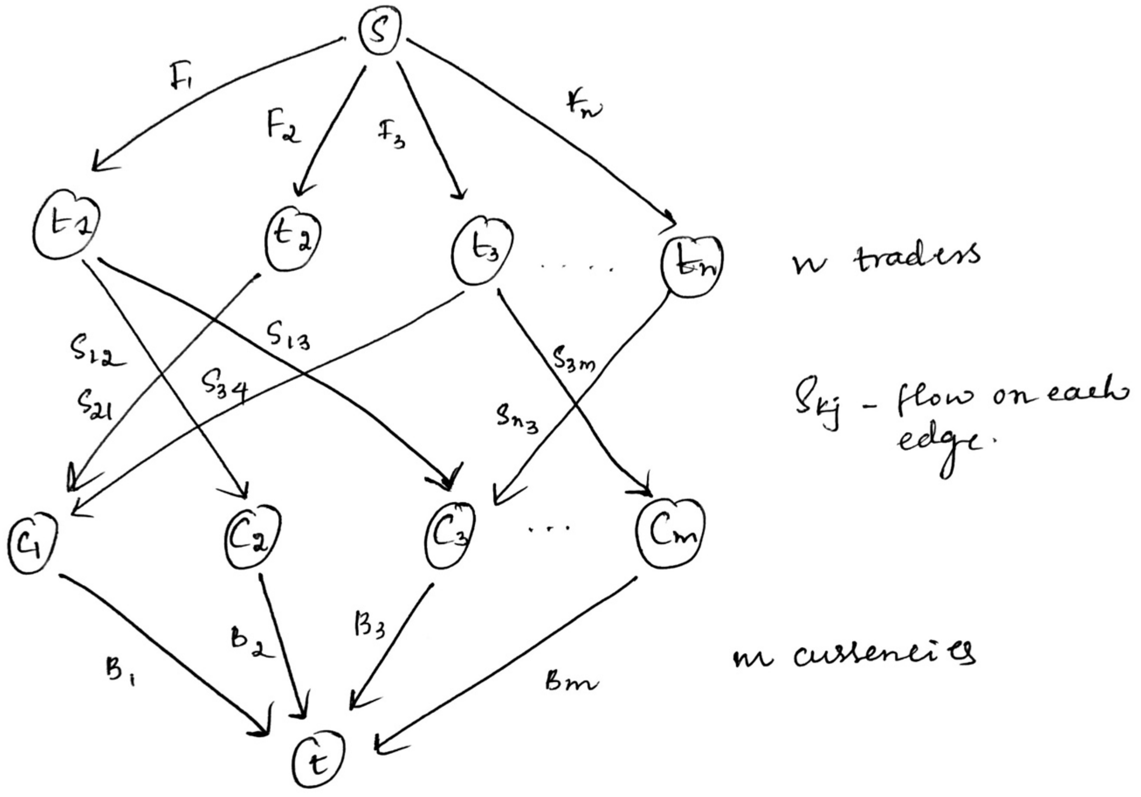
S_{kj} of his Francs for currency *$c_{j.}$* Then, each trader is connected to currencies he wants to exchange by directed edge with the capacity *S_{kj}*.
Next, we extend this bipartite graph to a network flow. We add the source S and connect it to every trader vertex *t_i* by an edge *(s, t_i)* of capacity *f_i*. We add the target *t* and for every currency vertex *$c_{j.}$* we add an edge *(c_j,t)* of capacity *b_j*.

Claim : The original problem has a solution that is bank can satisfy all the requests, if and only if the constructed network has a max-flow value $\sum_{i=1}^{n} f_i$

Proof =>) Assume that there is a solution. It means that the bank can process all the requests. So, we can push a flow of *f_i* i.e *f_1 , f_2 , f_3 …, f_n* from the source 's' to each traders *t_1 , t_2 , t_3, …, t_n* respectively. On the edges between traders and currencies, we assign a flow of *$S_{kj.}$* One trader can convert his Francs to multiple currencies and each currency exchange request can come from multiple traders. The edges between currencies and bank will have capacities *Bj* and we assign a flow value equal to S_{kj} . due to the capacity constraints. The Bank can convert to a particular currency (where *Bj >= S_{kj}.).* This is achievable since we are making valid requests of currency conversion to the Bank and bank can process all the requests.

Conversely =>) Assume there is a max-flow of value $\sum_{i=1}^{n} f_i$ This means that each trader will get a flow of f_i. Since the trader only needs the S_{kj} francs to be converted to the currency cj and all the currencies are unique by nature, the edge *t_i -> c_j* will have only S_{kj}.Because of capacity constraints on the edges, flow on the edge *t_i -> c_j* is also S_{kj}. Here we can see that, the bank is not overloaded with currency requests because of the capacity constraints.

Furthermore, we get a valid request assignment for the bank, when we run Ford Fulkerson algorithm (or any network flow algorithm) and pick edges with the above capacity constraints.



3.
We start by solving the problem as a bipartite graph problem. In this graph, one partition is a set of vertices $s_1, s_2, s_3, \ldots, s_n$ representing all n students. Another partition is a set of in-person classes, $c_1, c_2, c_3, \ldots, c_k$ representing k in-person classes. A Student s_i can be enrolled in more than one in-person classes and each class c_j consists of several students. Each student is connected to the classes he/she enrolled into through a directed edge with each edge **capacity being 1** as one student can enroll to one particular class only once.

Next, we extend this bipartite graph to a network flow. We add the source '*s*' and connect it to every student vertex S_i by an edge (s, s_i) of capacity m-1.This is because, we need to select one student from each class and maximum times a student can be selected should be less than m. Due to capacity constraint, maximum flow a student vertex can have is m-1.We add the target '*t* 'and for every class vertex c_j. we add an edge (c_j,t) of capacity 1 as only one student is selected from each class.

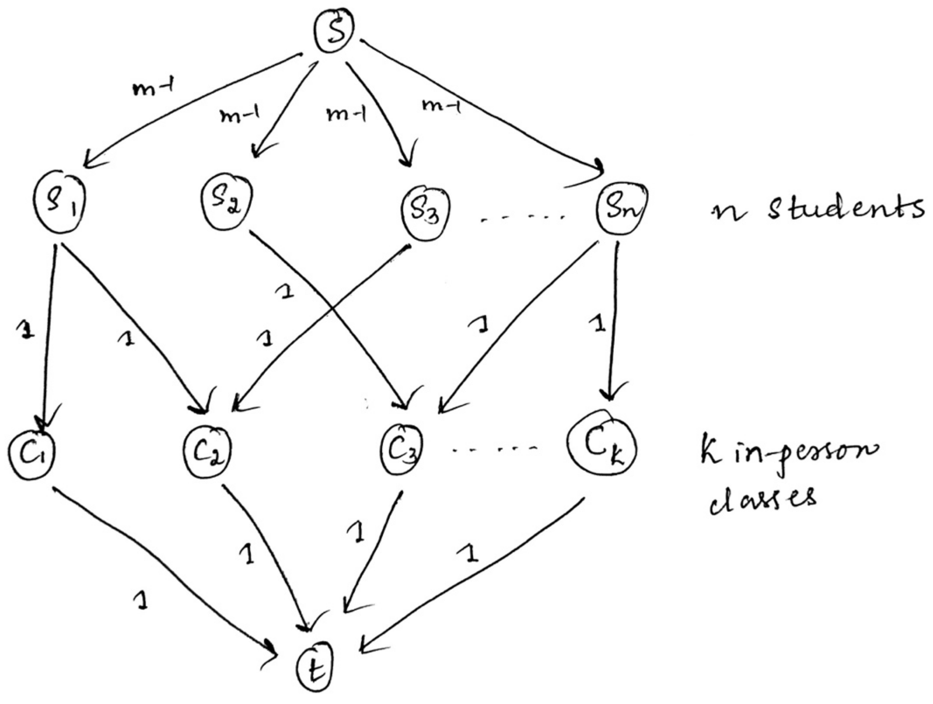
Claim : The original problem has a solution that is selection of one student from each in-person classes such that the maximum times a student can be selected is less than m exists if and only if the constructed network has a **maximum flow of 'k'**.

Proof =>) Assume that there is a solution. It means that the selection exists, i.e one student is selected from each in-person class such that the maximum times a student can be selected is less than m. So, we can push a flow value r_i $\{0 <= r_i <= (m-1)\}$ from the source s to each student (for valid selection). Since each student can only enroll in an in-person class only once and all the classes are unique by nature, the edge (s_i , c_j) capacity between student s_i and class c_j will only be 1 and we assign the flow value as 1 or 0 (each student can enroll in a particular class only once or he/she doesn't enroll). Since all the classes are unique, each outgoing flow from the student is 1. On the edges between the classes and target, we assign a flow value equal to 1, as we are required to select only one student from each in-person class.

This is achievable since we are making valid selection of students where each student is not selected more than (m – 1) times, and one student is selected from each in-person class. So, k students are selected. Thus, ***max flow = k***.

Conversely =>) Assume that, there is a max-flow of value ***k*.** This means that each student will get a flow of r_i $\{0 <= r_i <= (m-1)\}$. Because of capacity constraints, a student can register in maximum m-1 classes. Student can register in one class only once, so edge weight between student and class is either 0 or 1. We only take one student from each class without repeating the students m or more than m times. Here we can see that, whichever student we may choose, he will never be selected m times or more than m times. Thus, the final flow from each class to the target will also be 1. In total the flow would be k. There is never a mis selection because of our capacity constraints.

Furthermore, we get a valid selection of k students, when we run Ford Fulkerson algorithm (or any network flow algorithm) and pick edges with the above capacity constraints.



4.
a) We start by solving the problem as Directed acyclic graph. We have set of starting locations 'a_i' where i = 1 to k and set of ending vertices 'b_j' where j = 1 to k both depicting k starting and ending vertices respectively.

Apart from starting and ending vertices, we have intermediate vertices which are other locations, say 'p_m'. There are directed edges between the starting locations, intermediate locations and ending locations. We have been given that the edges are weighted. But we will be ignoring the edge weights and give edge capacities as 1 between starting locations, intermediate locations and ending locations. Since there are edges between intermediate locations as well, we give capacity 1 for the edges between intermediate locations.

Next step is to convert this graph into a network flow. We add new source vertex 's' and connect it with all the starting locations a_i via directed edges (s,a_i) of capacity 1.
We add another vertex target 't' and connect it with all the ending locations b_j via directed edges (b_j,t) of capacity 1.
The problem states that each group starts from one location, we know that no two groups can start from the same location.

Claim : The original problem has a feasible solution, that there exists k paths a_i -> b_j that start and end at different vertices such that they do not share edges if and only if **max flow =k** paths.

Proof =>) : Assume that we have a feasible solution, i.e, there are **k** paths a_i -> b_j such that no edges are shared.

Now we do push a flow of value 1, from source s to each starting location a_i as only one group can start from location a_i. On the edges between starting locations a_i and intermediate locations p_m, we assign a flow value as 1. With this, we can ensure that no two groups take same path. Since the edge used by a group becomes saturated (flow = capacity on that edge), no other group can take this edge to travel further. On the edges among intermediate locations and other intermediate locations, we again assign a flow of value 1. Since, no two groups take the same edge, the outgoing flow on each edge from intermediate locations p_m to ending locations b_j is also 1.

The final flow from the ending locations b_j to the sink t on each edge is 1. Therefore, we have a max flow of value k, which is same as the number of starting locations/ending locations. This must be possible since we have valid capacity constraints, and thus valid k tours/paths.

Conversely =>) Let us assume that there is a max flow of value k. This represents each starting location gets a flow of value 1. In total, the flow would be k, since there are k starting locations. Due to the capacity constraints on each edge in the graph G $\{(a_i, p_t), (p_t, p_{t'}), (p_t, b_j)\}$ have unit capacities and no two groups take the same edge. We also observe that all the tours reach an ending location when there is a maxflow of value k. Since, there are k ending locations all tours reach the destination.

Lastly, we get k valid paths a_i to b_j by running a network flow algorithm and pick edges with the above capacity constraints and unit flow on them.

Part b)

Let's setup the problem as a Directed Acyclic Graph. In this graph, the set vertices of starting locations 'a_i' where i = 1 to k and set of ending vertices 'b_j' where j = 1 to k both depicting k starting and ending vertices respectively.

Apart from starting and ending vertices, we have intermediate vertices which are other locations, say 'p_m'. There are directed edges between the starting locations, intermediate locations and ending locations. We have been given that the edges are weighted. But we will be ignoring the edge weights and give edge capacities as 1 between starting locations, intermediate locations and ending locations. Since there are edges between intermediate locations as well, we give capacity 1 for the edges between intermediate locations.

Now, we are splitting all the intermediate vertices/locations into two vertices each. We denote them as V_{in} and V_{out} ($V = \{V_{in}, V_{out}\}$).
We also create an edge between the vertices V_{in} and V_{out}.
The edge capacity between these two vertices would be 1.

By doing this we are converting each vertex into an edge and placing the capacity constraint on the vertex similar to an edge. By this, we are not allowing two or more groups to take the same vertex.

We will run our network flow algorithm on this new graph, where we have split the vertex into two vertices, V_{in} vertex incoming, and V_{out} vertex outgoing and there is a directed edge between V_{in} and V_{out} of capacity 1.

We then convert the above graph to a network flow.

We add a new vertex - source 's' and connect it to all the starting locations a_i via a directed edge (s, a_i) of capacity 1. We add another new vertex target 't' and connect all the ending locations b_j to t via a directed edge (b_j, t) of capacity 1.

Since, the problem states that each group i, starts from a starting location a_i, we know that no two groups start from the same location.

Claim - The original problem has a feasible solution if and only if max flow = k paths

Proof =>) Assume that we have a feasible solution, i.e., there are k paths from a_i to b_j such that no edges are shared as well as no vertices are shared.

We push a flow of value 1, from source s to each starting location a_i (a single tour starting from location a_i). On the edges between starting locations a_i and intermediate locations **(only V_{in})** p_t, we assign a flow value as 1. So, this ensures no two groups take same path ever. Since the edge used by a group becomes saturated (flow = capacity on that edge), no other group can take this edge to travel further. On the edges between V_{in} and V_{out} we also push a flow of value 1. This makes sure that the vertex V is visited only by one group. On the edges among intermediate locations **(only V_{out})** and other intermediate locations **(only V_{in})**, we again assign a flow of value 1. Since, no two groups take the same edge and there isn't any common vertex between the groups, the outgoing flow on each edge from intermediate locations **(only V_{out})** $_t$ to ending locations b_j is also 1.

The final flow from the ending locations b_j to the sink t on each edge is 1. Therefore, we have a max flow of value k, which is same as the number of starting locations/ending locations.This must be possible since we have valid capacity constraints, and thus valid k tours/paths.

Conversely =>) Assume that there is max flow of value k. This represents each starting location gets a flow of value 1. In total, the flow would be k, since there are k starting locations. Due to the capacity constraints on each edge in the graph G {(a_i, c_p), (p_t, $p_{t'}$), **(V_{in}, V_{out})**, (p_t, b_j)} have unit capacities, no two groups take the same edge, and no two groups take the same vertex. We also observe that all the tours reach an ending location when there is a maxflow of value k. Since, there are k ending locations all tours reach the destination.

Lastly, we get k valid paths a_i to b_j by running a network flow algorithm and pick edges with the above capacity constraints and unit flow on them.

5.

Question 5 :

(a) (i) We have to first check if the $\Sigma d(v) = 0$

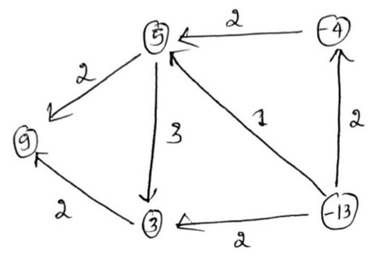
We have, $9+5+3-4-13=0$

Therefore, we can say that circulation is feasible in the given graph.

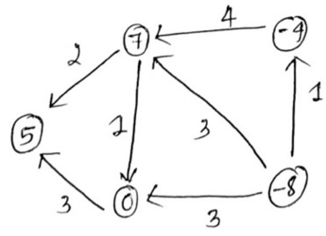
We will now proceed further.

(ii) We have to remove lower bounds from the graph.

$f(e) = l(e)$



iii) We have to remove demands and supply. After that, we get graph



The above graph has no more lower bounds.

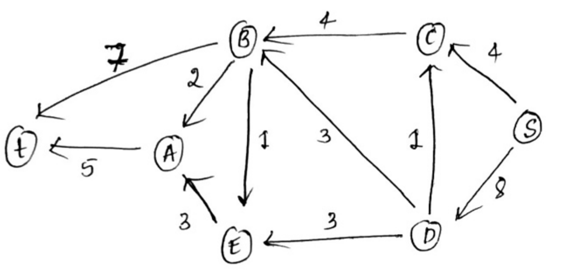
We have converted circulation with lower bounds and demands to circulation with demands.

We will have to again for $\Sigma d(v) = 0$

$5 + 7 + 0 - 4 - 8 = 0$

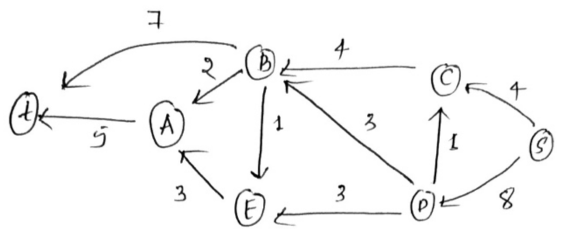
Therefore, there is still circulation feasible in the graph. We can proceed further.

(b) Converting the circulation with demands problem into the maximum flow problem.



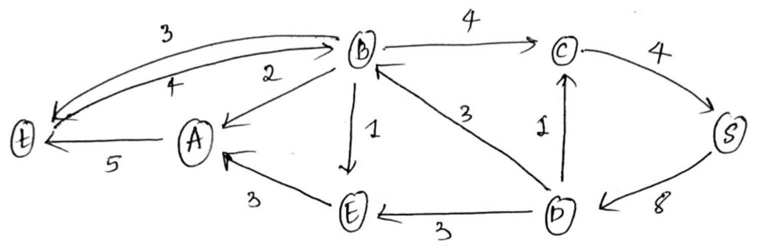
- Source node 's' is added and is connected to ~~+~~ $-(dv)$
- Sink node 't' is added and is connected to $+(dv)$
- We have removed demands / supplies from the vertices.
- With this, we have successfully converted the graph into max flow problem.

(c) We have to run the Ford-Fulkerson algorithm to get Max-Flow.



(i) lets consider the augmenting path : S-C-B-T

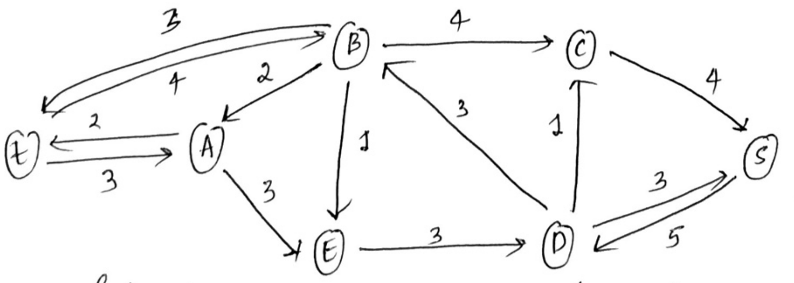
. $C_f(P) = 4$, $f(u,v) = 4$



After the first iteration edge $S \to C$, $C \to B$ have been saturated.

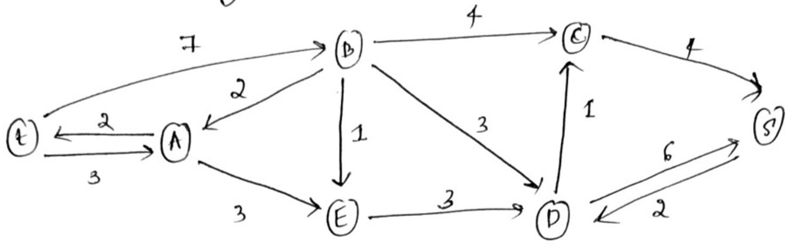
(ii) We are considering the augmenting path S-D-E-A-T

$C_f(P) = 3$, $f(u,v) = 7$



Edges $D \to E$ and $E \to A$ have been saturated.

iii) Augmenting path : S - D - B - T



$C_f(P) = 3$, $f(u,v) = 10$

Now, after third iteration, we can see that the edges D→B has been saturated. And, there is no more augmenting paths P found in the residual graph G_f.

Therefore, we can say that the graph is disconnected.

We have obtained max flow $f(u,v) = 10$ units.

We have $\Sigma d(v) = D = 12$

$D \neq$ max-flow.

There exist no circulation in the given graph.

Hence, circulation is not feasible.