CSCI 570 - Summer 2023 - HW 6 Solution

Out: July 26, 2023
Due: July 31, 2023

Graded Problems

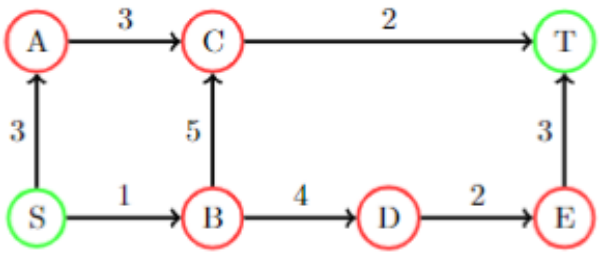
Problem 1

The following graph G has labeled nodes and edges between them. Each edge is labeled with its capacity.

(a) Draw the final residual graph G_f using the Ford-Fulkerson algorithm corresponding to the max flow. Please do NOT show all intermediate steps.

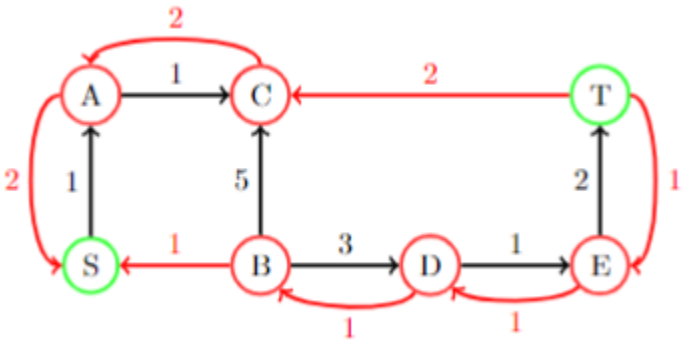
(b) What is the max-flow value?

(c) What is the min-cut?

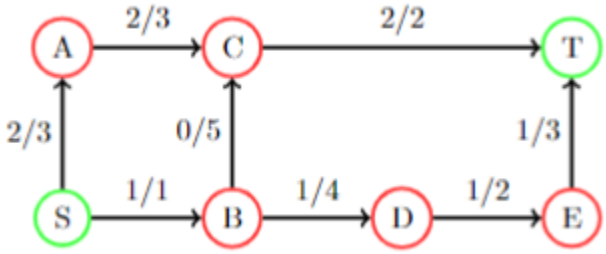


Solution:

(a) Final Residual Graph



Final Flow graph (not required to be drawn)



(b) The max flow is $2 + 1 = 3$.

(c) The min cut is {S, A, C} and {B, D, E, T}. (Quick check - This is correct because all the edges to and from the min cut either are saturated (CT, SB) or have zero flow (BC).)

Rubric (10 pts)

- 5 pts: Correct Final Residual graph.

  (−2 pts: Incorrect edge capacity.)
- 2 pts: Correct max flow value.
- 3 pts: Correct min cut sets.

Problem 2

Determine if the following statements are true or false. For each statement, briefly explain your reasoning.

(a) In a flow network, the value of flow from S to T can be higher than the maximum number of edge-disjoint paths from S to T. (Recall that edge-disjoint paths are paths that do not share any edge)

(b) For a flow network, there always exists a maximum flow that doesn't include a cycle containing positive flow.

(c) If you have non-integer edge capacities, then you cannot have an integer max-flow value.

(d) Suppose the maximum s-t flow of a graph has value f. Now we increase the capacity of every edge by 1. Then the maximum s-t flow in this modified graph will have a value of at most $f + 1$.

(e) If all edge capacities are multiplied by a positive number k, then the min-cut remains unchanged.

Solution:

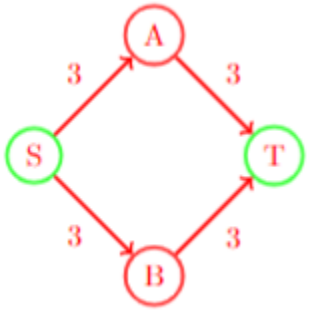
(a) True, consider the case when edges have capacity greater than 1.

(b) True, we can always remove the flow from such a cycle and still get the same flow value.

(c) False, consider a graph with source s, sink t and two nodes a and b. Let's say there is an edge from s to both a and b with capacity 0.5 and from both a and b to t with capacity 0.5, then the max-flow value for this graph is 1, which is an integer.

(d) False. Counter-Example:

In the following graph, the maximum flow has value $f = 3 + 3 = 6$. Increasing the capacity of every edge by 1 causes the maximum flow in the modified graph to have value $4 + 4 = 8$.



(e) True. The value of every cut gets multiplied by k, thus the relative-order of min-cut remains the same.

Rubric (15 pts)

For each question:

- 1 pt: Correct choice
- 2 pt: Valid reasoning for the answer

Problem 3

You are given a flow network with unit-capacity edges. It consists of a directed graph $G = (V, E)$ with source s and sink t, and $u_e = 1$ for every edge e. You are also given a positive integer parameter k. The goal is delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a subset of edges $F \subseteq E$ such that $|F| = k$ and the maximum s-t flow in the graph $G' = (V, E \setminus F)$ is as small as possible. Give a polynomial-time algorithm to solve this problem.

Follow up: If the edges have more than unit capacity, will your algorithm produce the smallest possible max-flow value?

Solution:

Algorithm:

- Assume the value of max-flow of given flow network $G(V, E)$ is g. By removing k edges, the resulting max-flow can never be less than $g - k$ when $|E| \geq k$, since each edge has a capacity 1 and removing each edge reduces the max-flow value by at most 1.
- According to max-flow min-cut theorem, there is an s-t cut with g edges crossing it. If $g \leq k$, then remove all edges in this s-t cut, decreasing max-flow to 0 and disconnecting s and t. Else if $g > k$, then remove k edges from this cut, creating a new cut with $g - k$ edges crossing it.
- In both cases, whether the max-flow is 0 or $g - k$, the max-flow cannot be decreased any further, thus giving us the maximum reduction in flow value.
- The algorithm has polynomial run-time. It takes polynomial time to compute the min-cut and linear time in k to remove k edges.

If some edges don't have unit capacity, removing k edges from min-cut in the above mentioned way does not guarantee to have the smallest possible max-flow value.

Rubric (15 pts)

- 12 pts: Correct algorithm proposal with polynomial time.
- 3 pts: Correct answer for non-unit edges.

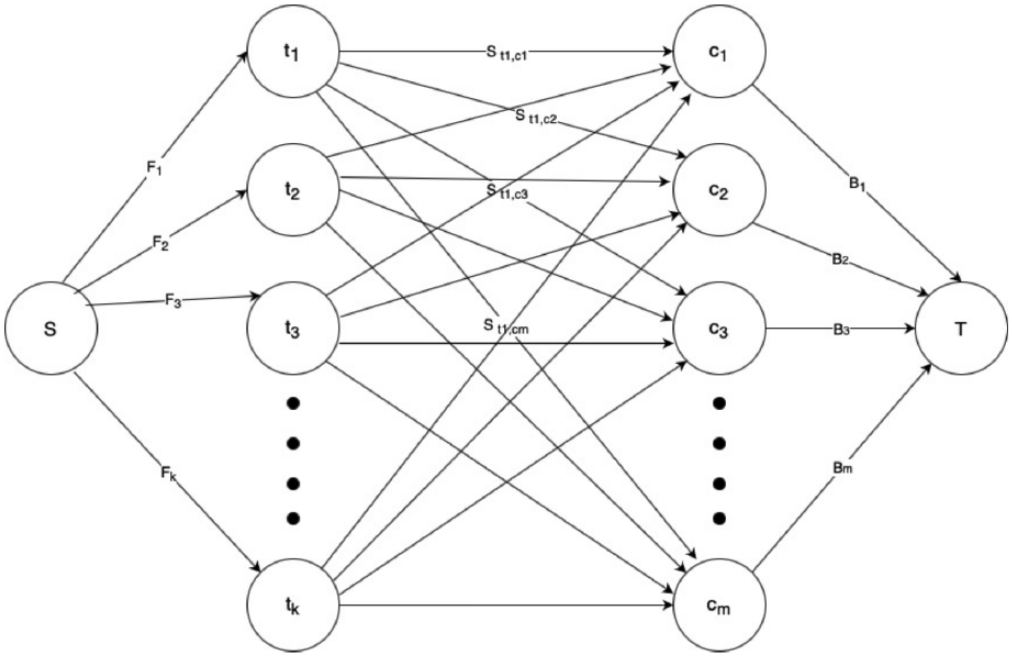
Problem 4

A tourist group needs to convert all of their USD into various international currencies. There are n tourists t_1, t_2, ..., t_n and m currencies c_1, c_2, ..., c_m. Each tourist t_k has F_k Dollars to convert. For each currency c_j, the bank can convert at most B_j Dollars to c_j. Tourist t_k is willing to trade at most S_{kj} of their Dollars for currency c_j. (For example, a tourist with 1000 dollars might be willing to convert up to 300 of their USD for Rupees, up to 500 of their USD for Japanese Yen, and up to 400 of their USD for Euros). Assume that all tourists give their requests to the bank at the same time.

(a) Design an algorithm that the bank can use to determine whether all requests can be satisfied. To do this, construct and draw a network flow graph, with appropriate source and sink nodes, and edge capacities.

(b) Prove your algorithm is correct by making an if-and-only-if claim and proving it in both directions.

Solution:



(a) Network Flow Construction and Algorithm:

We will solve the problem, by first constructing a network flow graph and then running Ford-Fulkerson or any other Network Flow Algorithm to get the max flow value.

Construction:

- Insert two new vertices, source node S and sink node T.
- Connect all the tourists t_k with source node S assigning edge weight equal to F_k. Here, F_k stands for the maximum USD tourist t_k can exchange.
- Then, connect all tourists t_k to all the currencies available i.e., c_j with each edge having weight S_{kj} which is the t_k tourist's limit to exchange their USD for a particular currency c_j.
- Connect currencies c_j with sink node T, with edge weight B_j, which is the maximum limit of that particular currency c_j that the bank can convert from USD.

In the graph constructed this way we can run Ford-Fulkerson or any Network Flow Algorithm from source S to sink T, and if the max-flow value is equal to $\sum_k F_k$ then we know all requests can be satisfied. In this case the flow on each edge (t_k, c_j) represents the amount that tourist t_k exchanges into currency c_j.

(b)

Claim:

The problem has a solution (i.e., all the tourists are able to exchange their specified USD while following all the constraints), if and only if the max-flow value on the constructed graph is $\sum_k F_k$.

Proof:

We have to prove the claim in both directions:

- **Proof for Forward Direction:**

  Assume there is a valid assignment such that the bank can satisfy all the tourist's requests for conversion while following all the constraints, and we want to prove a max-flow of value $\sum_k F_k$ exists.

  Suppose in this assignment, each tourist t_k is converting x_{kj} amount into each currency c_j. Then in the constructed network, we can send x_{kj} units of flow along each path $S \to t_k \to c_j \to T$. This clearly satisfies flow conservation and non-negativity. The flow along each edge (S, t_k) is exactly F_k, since that's equal to the amount of dollars that t_k exchanges in total. The flow along each edge (t_k, c_j) is at most S_{kj}, since that's the maximum amount that t_k is willing to convert into c_j. Also the flow along each edge (c_j, T) is at most B_j, since we know the bank is able to satisfy the total amount for each currency. Thus this flow also satisfies the capacity constraint and has total flow value $\sum_k F_k$.

- **Proof for Backward Direction:**

Assume a max-flow of value $\sum_k F_k$ exists in the constructed network, and we want to prove that a valid assignment exists.

We know the flow on each edge (t_k, c_j) represents the amount that tourist t_k exchanges into currency c_j. If max-flow of value $\sum_k F_k$ exists, then it means all the edges out of S are saturated. By flow conservation that means the flow values out of each t_k will sum up to F_k, meaning t_k can exchange all their requested amount. Also by the capacity constraint, t_k will convert at most S_{kj} dollars into c_j, and the total amount needed for each currency c_j doesn't exceed B_j, so they can be satisfied by the bank. Thus we know the bank can satisfy all the tourist's requests for conversion while following all the constraints.

Rubric (20 pts)

For part (a) - 10 pts

- 10 pts: Correct Construction of Network Flow graph.

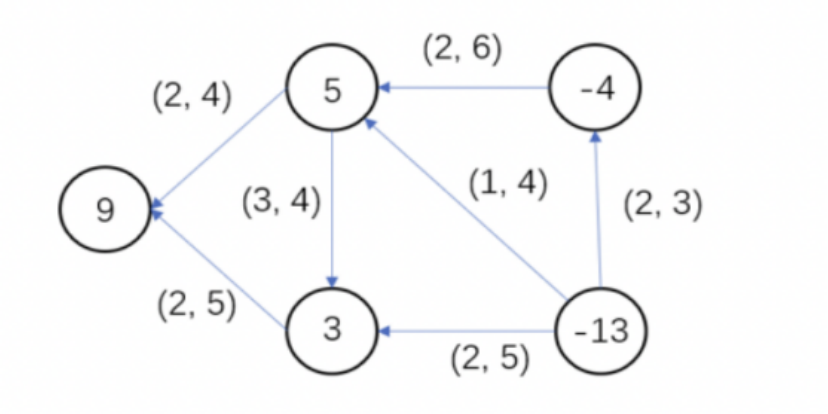
  (-2 pts: For each incorrect edge weight/node. (source and sink can be reversed))

For part (b) - 10 pts

- 4 pts: Correct Claim.
- 3 pts: Correct Forward Proof.
- 3 pts: Correct Backward Proof.

Problem 5

In the network G below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.



(a) Reduce the Feasible Circulation with Lower Bounds problem to a Feasible Circulation problem without lower bounds.

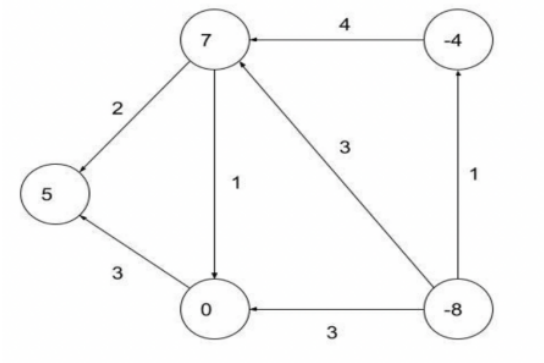
Solution:

i. Assign flow to each edge e to satisfy lower bound ℓ_e.

ii. At each node v, $L_v = f^{in}(v) - f^{out}(v)$, followed by $d'_v = d_v - L_v$ to update the demand.

iii. At each edge, $c'_e = c_e - \ell_e$ to update the capacity.

Updated network G':

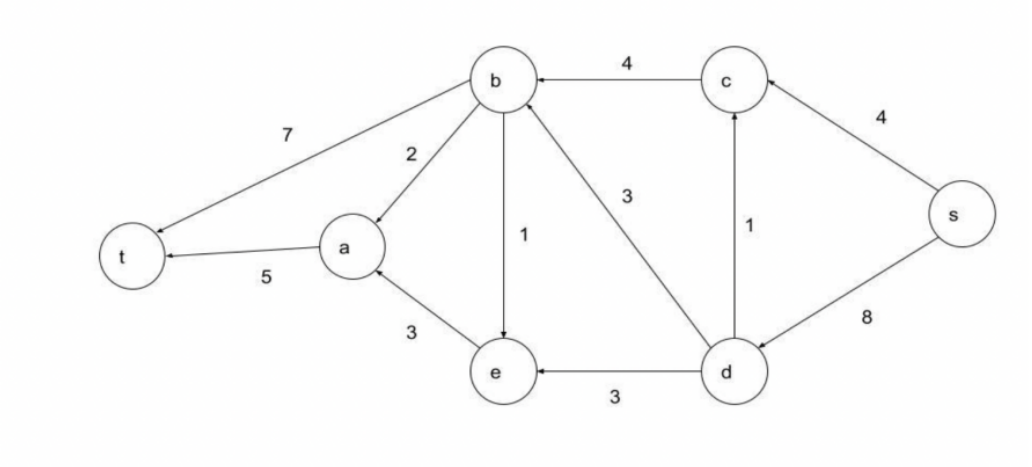


Rubric (8 pts)

- 4 pts: If all edge capacities are correctly calculated
- 4 pts: If all demand values are correctly calculated

(b) Reduce the Feasible Circulation problem obtained in part (a) to a Max Flow problem in a Flow Network.

Solution: The max flow network is as follows:



Rubric (8 pts)

- 2 pts: Add S and T
- 2 pts: Connect S with correct nodes

- 2 pts: Connect T with correct nodes
- 2 pts: Give correct capacities to edges

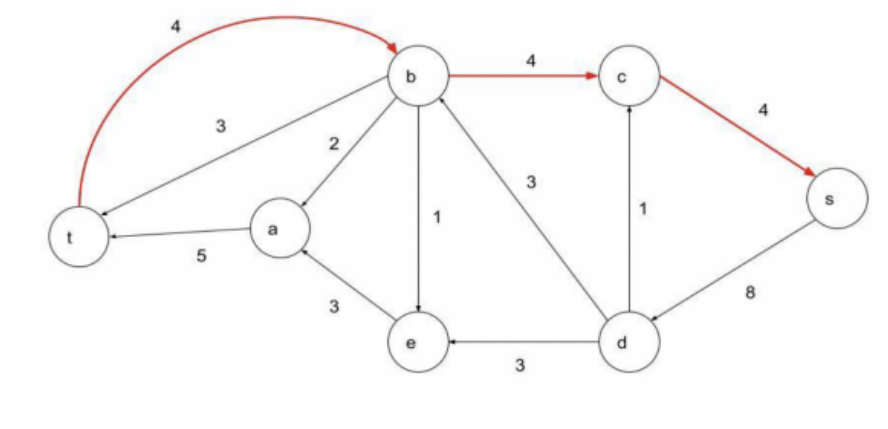
(c) Solve the resulting Max Flow problem and explain whether there is a Feasible Circulation in the original G.

Solution:

- Candidate Solution 1:

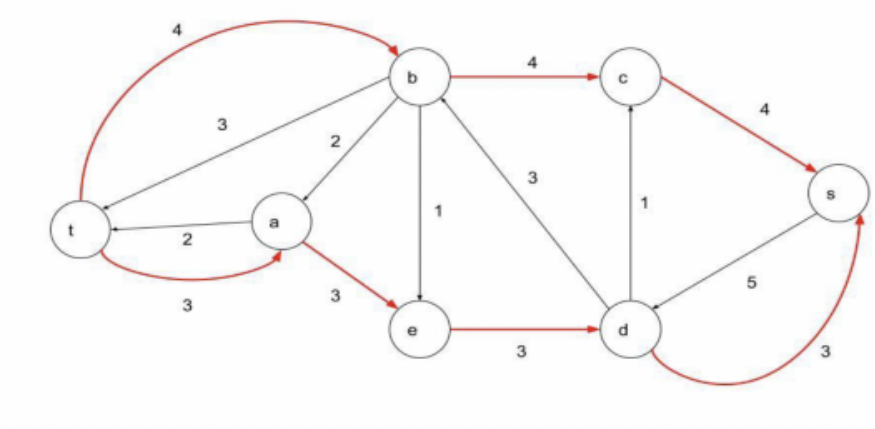
  First Augmenting Path: $s \to c \to b \to t$ with flow $= 4$

  Residual Graph 1:



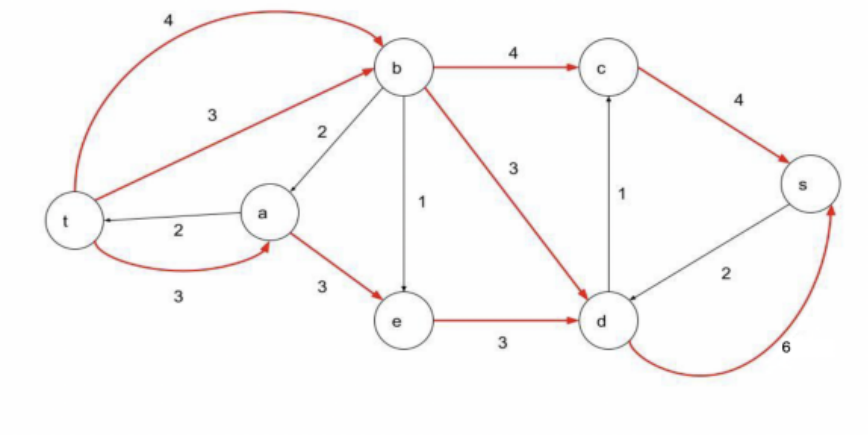
  Second Augmenting Path: $s \to d \to e \to a \to t$ with flow $= 3$

  Residual Graph 2:



Third Augmenting Path: $s \to d \to b \to t$ with flow $= 3$

Residual Graph 3:



- Candidate Solution 2:

  First Augmenting Path: $s \to c \to b \to t$ with flow $= 4$

  Second Augmenting Path: $s \to d \to b \to a \to t$ with flow $= 2$

  Third Augmenting Path: $s \to d \to b \to t$ with flow $= 1$

  Fourth Augmenting Path: $s \to d \to e \to a \to t$ with flow $= 3$

- Candidate Solution 3:

  First Augmenting Path: $s \to d \to e \to a \to t$ with flow $= 3$

Second Augmenting Path: $s \to d \to b \to a \to t$ with flow $= 2$

Third Augmenting Path: $s \to d \to c \to b \to t$ with flow $= 1$

Fourth Augmenting Path: $s \to c \to b \to t$ with flow $= 3$

Fifth Augmenting Path: $s \to d \to b \to t$ with flow $= 1$

All solutions give Max-flow value $= 10$.

Since the value of Max Flow is less than the total demand value $D = 12$, there is No Feasible solution in the circulation network G', and therefore there is no feasible circulation in the circulation with lower bounds network G.

Rubric (4 pts)

- 2 pts: Finding correct Max-Flow and presenting their appropriate residual graphs according to the sequence of augmenting paths that the student has chosen
- 1 pt: Correctly concluding "No Feasible Circulation"
- 1 pt: Reasoning behind "No Feasible Circulation"

Ungraded Problems

Problem 6

USC Admissions Center needs your help in planning paths for Campus tours given to prospective students or interested groups. Let USC campus be modeled as a weighted, directed graph G containing locations V connected by one-way roads E. On a busy day, let k be the number of campus tours that have to be done at the same time. It is required that the paths of campus tours do not use the same roads. Let the tour have k starting locations $A = \{a_1, a_2, ..., a_k\} \subseteq V$. From the starting locations, the groups are taken by a guide on a path through G to some ending location in $B = \{b_1, b_2, ..., b_k\} \subseteq V$. Your goal is to find a path for each group i from the starting location, a_i, to any ending location b_j such that no two paths share any edges, and no two groups end in the same location b_j.

(a) Design an algorithm to find k paths $a_i \to b_j$ that start and end at different vertices, such that they do not share any edges.

(b) Modify your algorithm to find k paths $a_i \to b_j$ that start and end in different locations, such that they do not share any edges **or vertices**.

Solution:

(a) The complete algorithm is as follows:

1. Create a flow network G' containing all vertices in V, all directed edges in E with capacity 1, and additionally a source vertex s and a sink vertex t. Connect the source to each starting location with a directed edge (s, a_i) and each ending location to the sink with a directed edge (b_i, t), all with capacity 1.

2. Run Ford-Fulkerson on this network to get a maximum flow f on this network. If $|f| = k$, then there is a solution; if $|f| < k$, then there is no solution, so we return FALSE.

3. To extract the paths from a_i to b_j (as well as which starting location ultimately connects to which ending location), run a depth-first search on the returned max flow f starting from s, tracing a path to t. Remove these edges and repeat k times until we have the k edge disjoint paths. To justify correctness, any argument conveying that there is a flow of size k if and only if there are k edge disjoint paths from A to B is enough.

(b) Duplicate each vertex v into two vertices v_{in} and v_{out}, with a directed edge between them. All edges (u, v) now become (u, v_{in}); all edges (v, w) now become (v_{out}, w). Assign the edge (v_{in}, v_{out}) capacity 1. With this transformation, we now have a graph in which there is a single edge corresponding to each vertex, and thus any paths that formerly shared vertices would be forced to share this edge. Now, we can use the same algorithm as in part

(a) on the modified graph to find k edge disjoint paths sharing neither edges nor vertices, if they exist.

Rubric (20 pts)

(a)

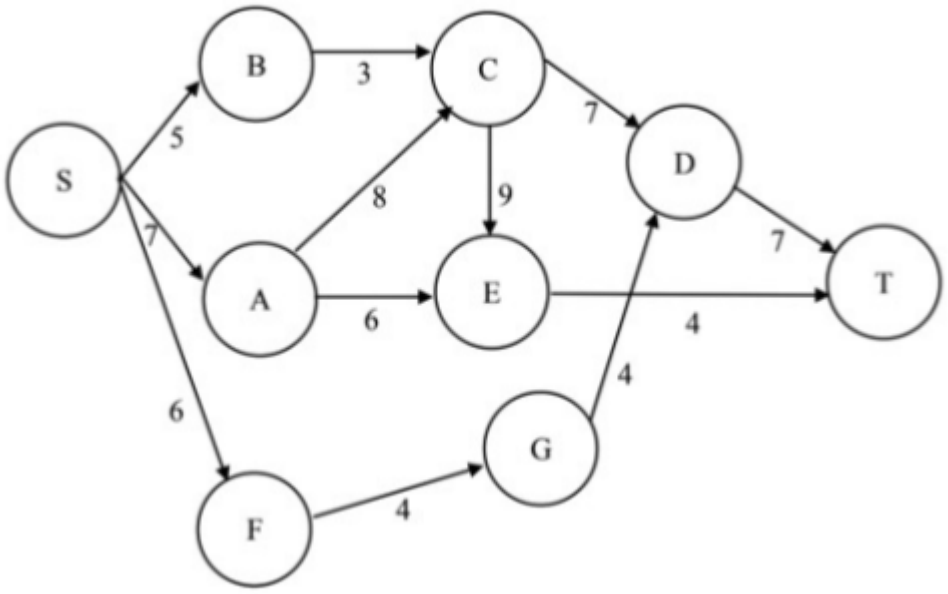
- 8 pts: Correct Construction of graph.
- 4 pts: Output k paths.

(b)

- 8 pts: Correct Construction of graph and apply (a) to solve the graph.

Problem 7

Perform two iterations (i.e., two augmentation steps) of the scaled version of the Ford-Fulkerson algorithm on the flow network given below. You need to show the value of Δ and the augmentation path for each iteration, and the flow f and $G_f(\Delta)$ after each iteration. (Note: iterations may or may not belong to the same Δ-scaling phase)



(a)

i. Give the value of Δ and the augmentation path

ii. Show the flow after the first iteration

iii. Show the $G_f(\Delta)$ after the first iteration

(b)

i. Give the value of Δ and the augmentation path

ii. Show the flow after the second iteration

iii. Show the $G_f(\Delta)$ after the second iteration

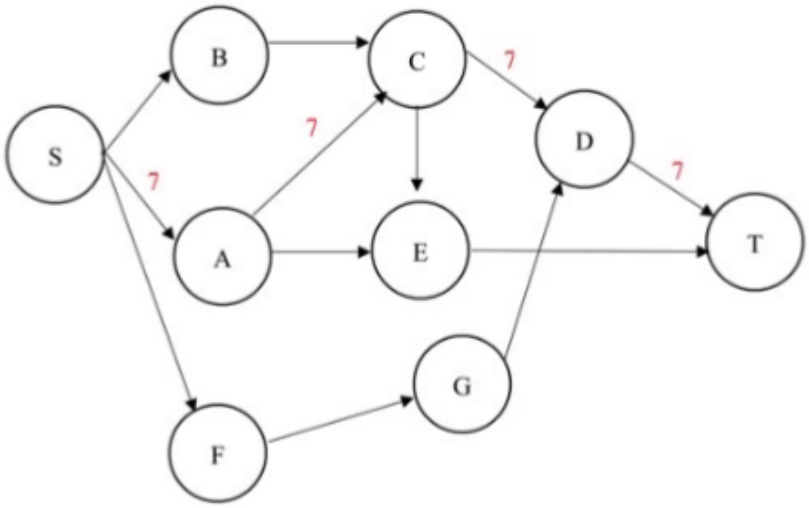
(c) Can the choice of augmentation paths in the scaled version of Ford-Fulkerson affect the number of iterations? Explain why.

Solution:

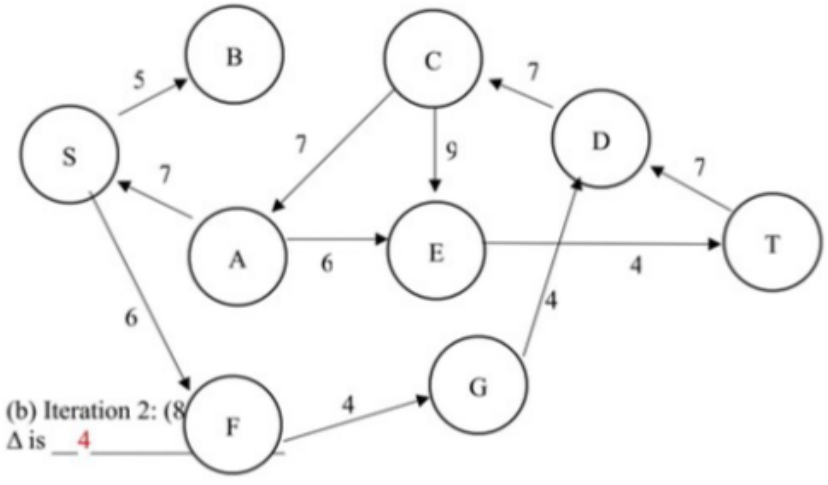
(a)

i. $\Delta = 4$; Augmentation path: SACDT (there are a few different choices of augmentation path)

ii. Flow after first iteration :



iii. $G_f(\Delta)$ after first iteration :

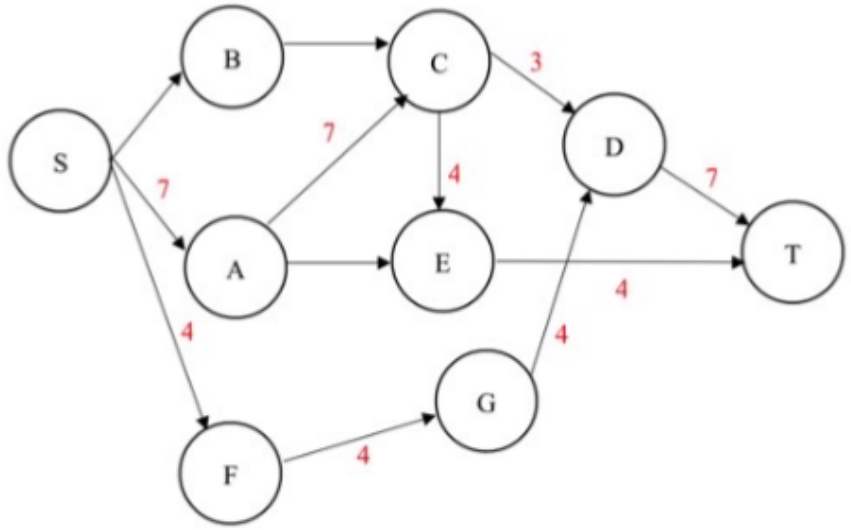


(b) Iteration 2: (8
Δ is 4

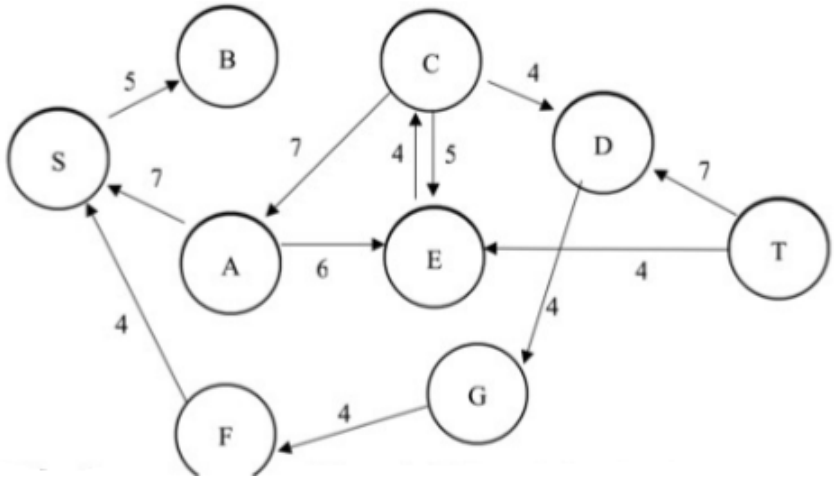
(b)

i. $\Delta = 4$; Augmentation path: SFGDCET (there are a few different choices of augmentation path)

ii. Flow after second iteration :



iii. $G_f(\Delta)$ after second iteration :



(c) Yes, for example, if we had chosen paths with bottleneck values of 4 (SAET and SFGDT) in the first two iterations, then we should have needed more than two iterations to get to max flow. But with the choice of augmentation paths given above, we were able to get to max flow in 2 iterations.

Or the argument could be: Yes, there are still different choices for augmentation paths within a given scaling phase. The different augmentation paths could have different bottleneck values. This can affect how quickly the value of flow goes up and therefore affect how many iteration we will have to perform.

Rubric (20 pts)

Parts a) + b)

- Computing the Delta values: 3+3 = 6
- Aug path + flow: 2+2 = 4
- G_f (Δ): 3+3 = 6

  (-1 for 1 edge missing/wrong. -2 for more.)

Part c)

0 If the answer is No. If Yes, 1 for the answer, 3 for the explanation. Partial credit 1/2 if the explanation is incomplete/unclear.

Problem 8

Consider a case where there are n tasks, with time requirements $r_1, r_2, \ldots, r_n$ in hours. On the project team, there are k people with time availabilities $a_1, a_2, \ldots, a_k$ in hours. For each task h and person i, you are told if person i has the skills to do task h. **You are to decide if the tasks can be split up among the people so that all tasks get done.** People only execute tasks they are qualified for, and no one exceeds their time availability. Remember that you can split up one task between multiple qualified people.

In addition, there are group constraints. For instance, even if each of you and your two teammates can in principle spend 4 hours on the project, you may have decided that between the three of you, you only want to spend 10 hours. Formally, we assume that there are m constraint sets of people $S_1, S_2, \ldots, S_m$ (where each $S_j \subseteq \{1, \ldots, k\}$), with set constraints $t_1, t_2, \ldots, t_m$. Then, any valid solution must ensure, in addition to the previous constraints, that the combined work of all people in S_j does not exceed t_j, for each set j. Note that one person may belong to at most one constraint set.

Give an algorithm with running time polynomial in n, m, k for this problem, under the assumption that all the S_j's are disjoint, and prove that your algorithm is correct.

Solution:

We will have one node u_h for each task h, one node v_i for each person i, and one node w_j for each constraint set S_j. In addition, there is a source s and a sink t. The source connects to each node u_h with capacity r_h. Each u_h connects to each node v_i where person i is able to do task h, with infinite capacity. If a person i is in no constraint set, node v_i connects to the sink t with capacity a_i; otherwise, v_i connects to the node w_j where $i \in S_j$, with capacity a_i. (Notice that because the constraint sets are disjoint, each person's node connects to at most one other node.) Finally, each node w_j connects to the sink t with capacity t_j .

We claim that this network has an s-t flow of value at least $\sum_h r_h$ if and only if the tasks can be divided between the people based on their availability.

- For the forward proof, assume that there is such a flow. For each person i, assign them to do as many units of work on task h as the flow from u_h to v_i. As the flow value is at least $\sum_h r_h$, the flow saturates all the edges out of the source (the total capacity out of the source is only $\sum_h r_h$), and by flow conservation, each job is fully assigned to people. Also, because the capacity on the (unique) edges out of v_i is a_i, no person does more than a_i units of work. Moreover, because the only way to send on the flow into v_i is to the node w_j for nodes $i \in S_j$, by the capacity constraint on the edge (w_j, t), the total work done by people in S_j is at most t_j.
- Conversely, if we have an assignment that has person i doing x_i^h units of work on task

h, meeting all constraints, then we send x_i^h units of flow along the path $s \to u_h \to v_i \to t$ (if person i is in no constraint sets), or along $s \to u_h \to v_i \to w_j \to t$ (if person i is in constraint set S_j). This clearly satisfies conservation and non-negativity. The flow along each edge (s, u_h) is exactly r_h, because that is the total amount of work assigned on job h. The flow along each edge (v_i, t) or (v_i, w_j) is at most a_i, because that is the maximum amount of work assigned to person i. And the flow along each (w_j, t) is at most t_j, because each set constraint was satisfied by the assignment. So we have exhibited an s-t flow of total value at least $\sum_h r_h$.

The runtime is the time required to solve a max-flow problem on a graph with $O(n+k+m)$ nodes and $O(nk+m)$ edges. This can be made to run in polynomial time with any algorithm that solves the max flow problem in polynomial time.

Rubric (20 pts)

- 10 pts: Correct construction of Network Flow graph.

  (-2 pts: For each incorrect edge weight/node.)

- 4 pts: Proving that a valid solution of the flow problem corresponds to a valid solution for the given problem.

- 4 pts: Proving that a valid solution for the given problem corresponds to a valid solution of the flow problem.

- 2 pts: for polynomial time complexity.