CSCI 570 - Fall 2022 - HW 8 Solution

Due: October 26, 2022

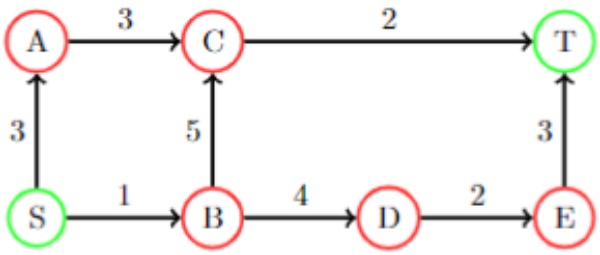
Problem 1

The following graph G has labeled nodes and edges between it. 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.

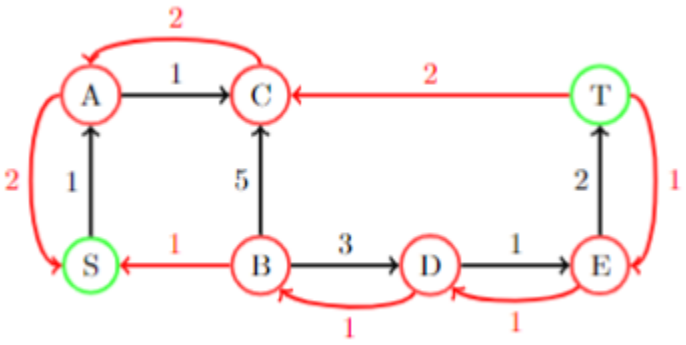
(b) What is the max-flow value?

(c) What is the min-cut?

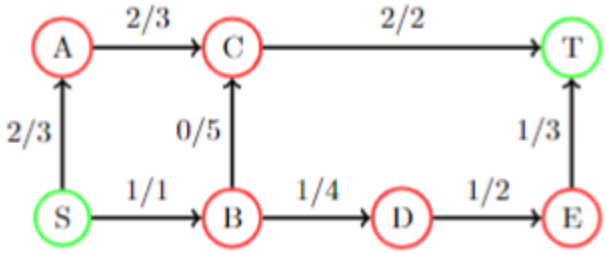


Solution:

(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 cuts sets are either saturated or have no no-flow.)

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. (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.

(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 edges are multiplied by a positive number k, then the min-cut remains unchanged.

Solution:

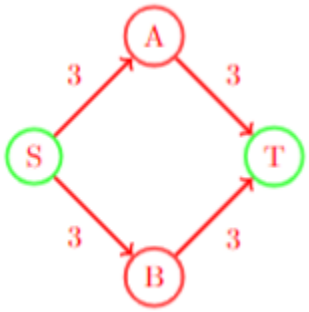
(a) True, consider the case when edges have capacity greater than 1.

(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 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 and briefly explain its correctness.

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. 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, and create a new cut with $g - k$ edges.
- In both the 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.
- The algorithm has polynomial run-time. It takes polynomial time to compute the minimal-cut and linear time in k to remove k edges.

If all the 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)

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

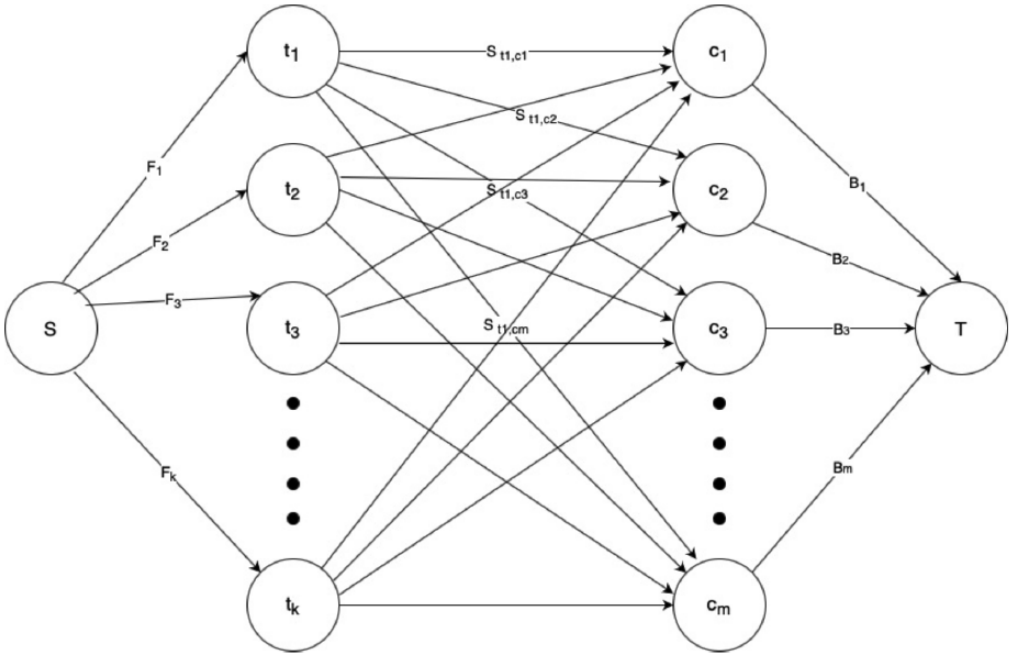
Problem 4

A tourist group needs to convert 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 as much as S_{kj} of his Dollars for currency c_j. (For example, a tourist with 1000 dollars might be willing to convert up to 200 of his USD for Rupees, up to 500 of his USD for Japanese Yen, and up to 300 of his 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 satisfy all the requests (if it is possible). 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 a 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 assignments.

Construction:

- Insert two new vertices, source node S and sink node T.
- Connect all the tourists t_i with source node S assigning edge weight equal to F_i. Here, F_i stands for the maximum USD tourist t_i can exchange.
- Then, connect all tourists t_i to all the currencies available i.e., c_j with each edge having weight S_{ij} which is the t_i tourist's limit to exchange his 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 run Ford-Fulkerson or any Network Flow Algorithm from source S to sink T, the algorithm will return assignments, if it exists, that can solve the requests while following all the proposed constraints.

(b)

Claim:

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

Proof:

We have to prove the claim in both directions:

- **Proof for Forward Direction:**

  Let us assume there is a valid assignment and prove that a max-flow of value $\sum_k F_k$ exists.

  Assume there is a valid assignment such that the bank can satisfy all the tourist's requests for conversion while following all the constraints. This means that all the outgoing edges from source S to each tourist t_i is saturated. Thus, we get the max-flow in the graph from S to T is $\sum_k F_k$.

- **Proof for Backward Direction:**

  Let us assume a max-flow of value $\sum_k F_k$ exists and prove that a valid assignment exists.

  Let us assume we have max-flow in the above constructed graph. The max-flow = $\sum_k F_k$. From the above constructed Network flow graph we can see that, if max-flow = $\sum_k F_k$ exists, then it means all the edges are saturated, which in turn means all

the tourists were able to convert their currencies. Hence it is proved that a valid assignment exists if a max-flow $= \sum_k F_k$ exists.

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

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)

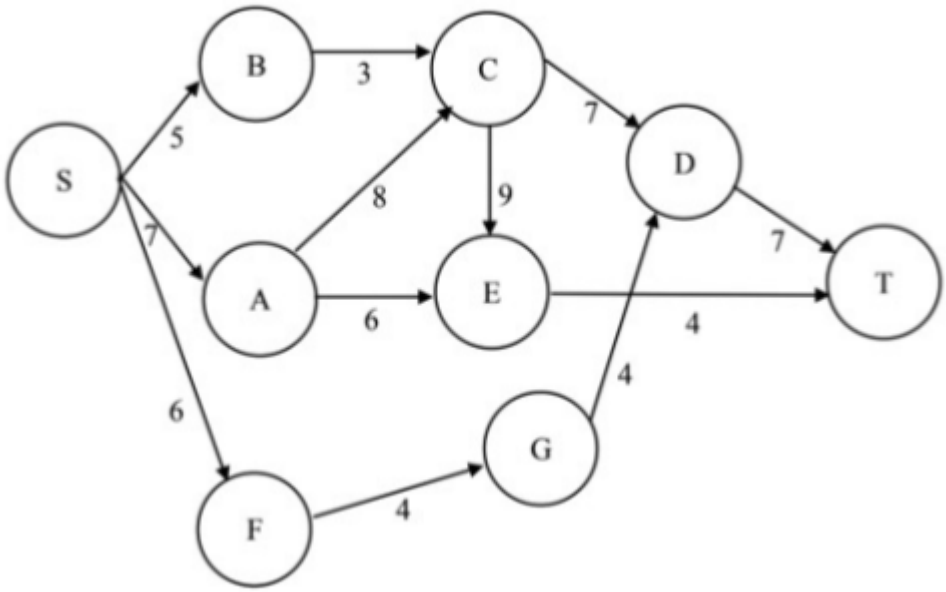
- 8 pts: Correct Construction of graph.
- 4 pts: Output k paths.

(b)

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

Problem 6

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 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

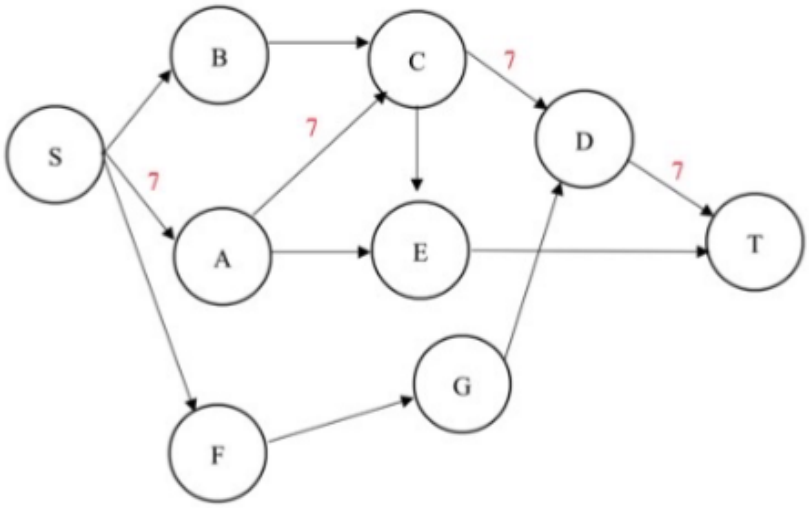
(c) Can the choice of augmentation paths in the scaled version of Ford-Fulkerson affect the number of iterations? Explain why.

Solution:

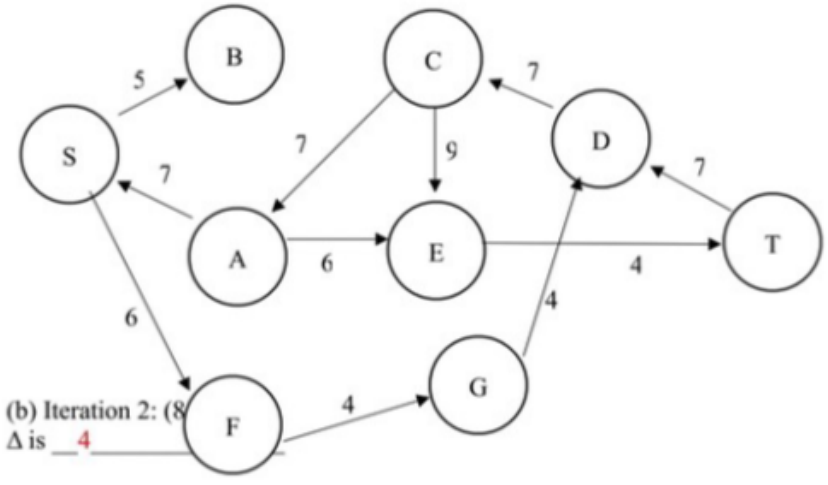
(a)

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 :

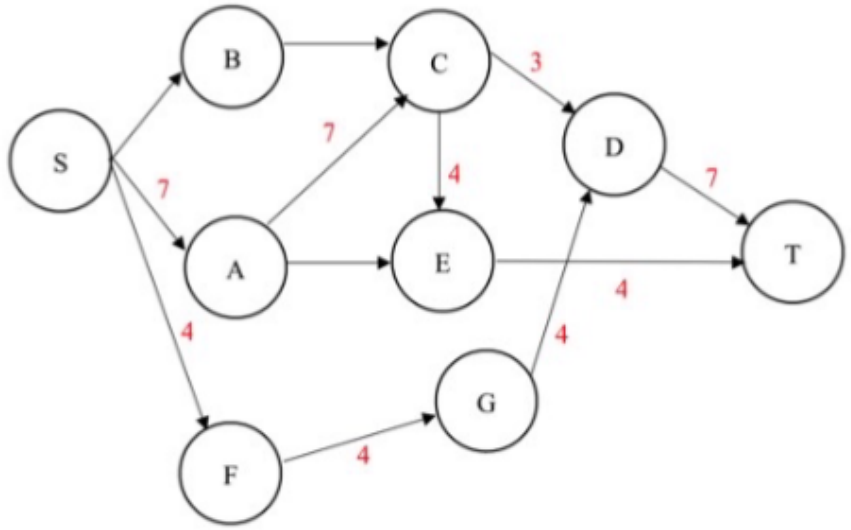


(b) Iteration 2: (8
Δ is __4__

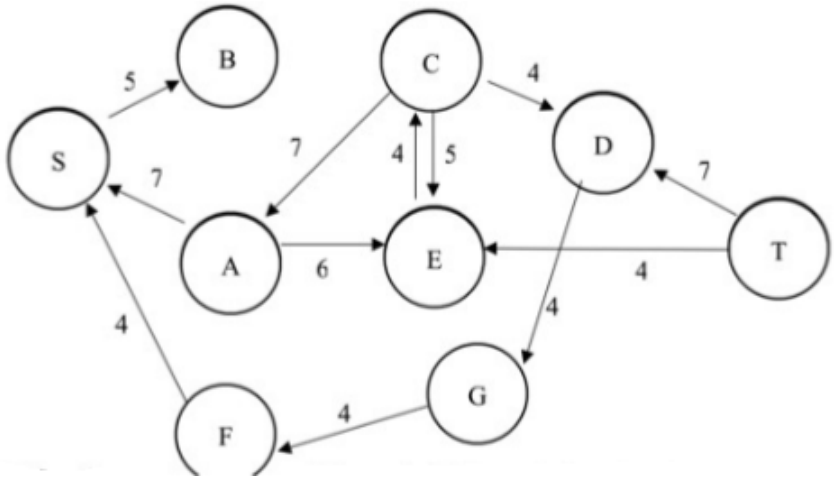
(b)

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.