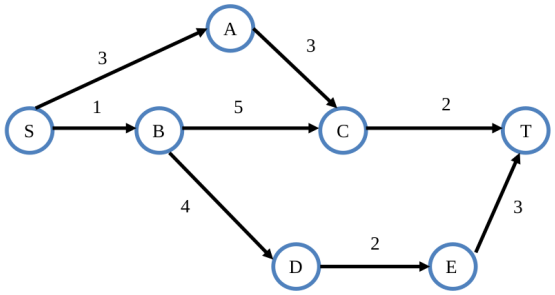
CSCI 570 - Spring 2021 - HW 4

Due April 1, 2021

1 Running the Algorithm (15 points)

You are given the following graph G. Each edge is labeled with the capacity of that edge.

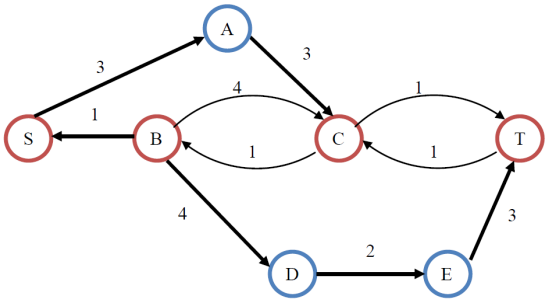


a. Find a max-flow in G using the Ford-Fulkerson algorithm. Draw the residual graph G_f corresponding to the max flow. You do not need to show all intermediate steps. (10 pts)

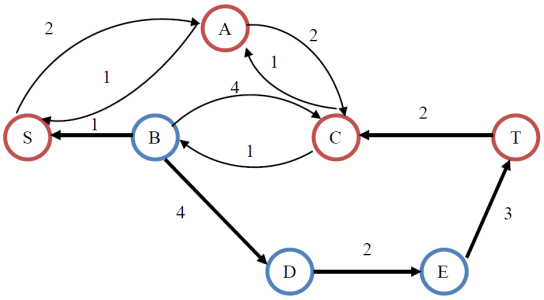
b. Find the max-flow value and a min-cut. (5 pts)

Solution:

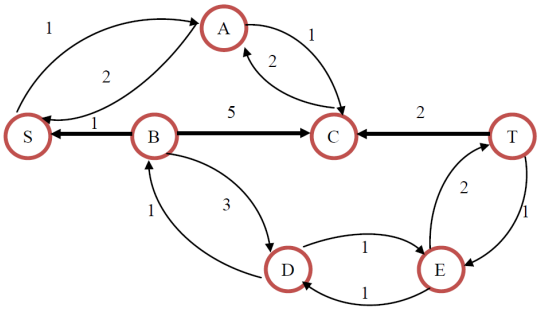
(a) Step 1: Augment path [S, B, C, T], bottleneck is 1, residual graph:



Step 2: Augment path [S, A, C, T], bottleneck is 1, residual graph:

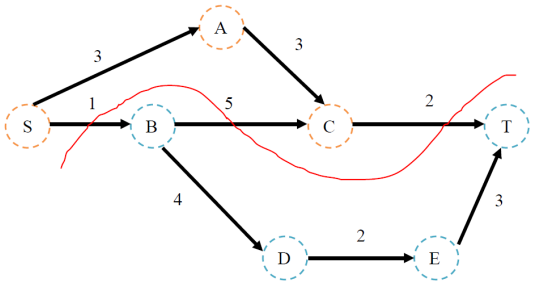


Step 3: Augment path: [S, A, C, B, D, E, T], bottleneck is 1, residual graph:



Step 4: No forward path from S to T, terminate.

(b) The maximum flow is 3. The Min-Cut is: [S, A, C] and [B, D, E, T].



Grading:
(1) - Given the correct residual graph,show the augment path of each step, and draw the residual graph of the last step.: 10 pts.
(2) - The max flow is correct: 2 pts.
(3) - The min-cut is correct: 3 pts.

2 Trading Currencies (20 points)

A group of traders are leaving Switzerland, and need to convert their Francs into various international currencies. There are n traders $t_1, t_2, ..., t_n$ and m currencies $c_1, c_2, ..., c_m$. Trader t_k has F_k Francs to convert. For each currency c_j, the bank can convert at most B_j Franks to c_j. Trader t_k is willing to trade as much as S_{kj} of his Francs for currency c_j.(For example, a trader with 1000 Francs might be willing to convert up to 200 of his Francs for USD, up to 500 of his Francs for Japanese's Yen, and up to 200 of his Francs for Euros). Assuming that all traders give their requests to the bank at the same time, design an algorithm that the bank can use to satisfy the requests (if it is possible).

Solution:

This is set up as a flow network, with Francs flowing from the source s to the sink t. A row of vertices $t_1, ..., t_n$ represents the traders, and a row of vertices $b_1, ..., b_m$ represents the currency held by the bank. The edge (s, t_i) has capacity F_i which gives the amount trader i wants to change. The edge $(s_i, b_j$) has capacity S_{ij} giving the maximum number of Francs i wants to trade into currency j. Finally, the edge (b_j, t) with capacity B_j gives the limit on the amount of currency j that is available. If there is a flow f in the network with $|f| = \sum_i F_i$ then all traders are able to convert their currencies.
To justify correctness, you need to prove that all traders are able to convert their currencies if and only if there is a flow f, $|f| = \sum_i F_i$.
If you apply the Ford-Fulkerson algorithm on the flow network, the time complexity would be $O(n \cdot m \cdot |f|)$.
(An alternate solution which reverses the source and sink, and has currency flow

to from the banks to traders is also valid.)

Grading:

- Given a reasonable flow network to solve this problem: 6 pts.
- Given the right description of the bipartite graph, including nodes, edges, edge directions and their capacities: 10 pts.
- Describe on what condition the bank can satisfy all requests ($|f| = \sum_i T_i$): 4 pts

3 Getting Vaccinated (20 points)

After getting vaccinated, students will be able to return to in-person classes next semester. There will be n students, $s_1, s_2, ..., s_n$, return to in-person classes, and there will be k in-person classes. Each in-person class consists of several students and a student can be enrolled in more than one in-person class. We need to select one student from each in-person class and the maximum times a student is selected should be less than m. Give a polynomial time algorithm to find such a selection if there is one, or to indicate that such a selection is not possible. Prove that your solution is correct.
[Hint: Reduce this problem to maximum flow problem.]

Solution:

We reduce this problem to maximum flow problem as follows:
Construct a graph with $n + k + 2$ vertices: n vertices $s_1, s_2, \ldots, s_n$ corresponding to the n students, k vertices $c_1, c_2, \ldots, c_k$ corresponding to the k classes, and two special vertices s and t.
For $i = 1, \ldots, n$, put an edge from s to s_i with capacity $m - 1$. Put an edge from s_i to c_j with capacity 1 if student s_i is in the c_j. For $j = 1, \ldots, k$, put an edge from c_j to t with capacity 1.
We now find a maximum flow in this graph. If the flow has value k, then there is a way to select one student from each class such that each student is not selected more than $m - 1$ times. The flow has one incoming edge for each class; if for class c_j this edge comes from student s_i then s_i is the student selected for class c_j.
To prove that the solution is correct is to prove that the reduction is correct: the graph has a maximum flow of value k only if there is a way to select one student from each class such that each student is not selected more than $m - 1$ times. Since the graph has a flow of value k, those edges from s_j to t are all saturated. The conservation constraint for each c_j implies that some students in c_j is selected. The capacities of those edges from s to s_i being $m - 1$ ensure

that each student is selected less than m times.

Grading:

- Given the correct vertices, $s_1, s_2, \ldots, s_n$, $c_1, c_2, \ldots, c_k$, s and t: 4 pts.
- Given the correct edges, s to s_i, s_i to c_j, c_j to t: 4 pts
- How to a find a selection using the maximum flow of the constructed graph: 6 pts.
- Proof of correctness: 6 pts.

4 Visiting Campus (25 points)

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 $A = \{ a_1, a_2, ..., a_k \} \subset V$. From the starting locations the groups are taken by a guide on a path through G to some ending location in $B = \{ b_1, b_2, ..., b_k \} \subset V$. 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 $a_i \rightarrow b_j$ that start and end at different vertices and such that they do not share any edges. (15 pts)

b. Modify your algorithm to find k paths $a_i \rightarrow b_j$, that start and end in different locations and such that they share neither vertices nor edges. (10 pts)

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. We will later show that it is always the case that $|f| <= k$.
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 disjoint paths.
To justify correctness, any argument conveying that there is a flow of size k if and only if there are k disjoint paths is correct.

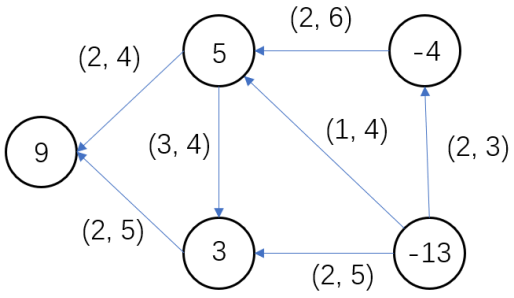
(b) Duplicate each vertex v into two vertices vin and $vout$, with a directed edge between them. All edges (u, v) now become (u, vin); all edges (v, w) now become $(vout, w)$. Assign the edge $(vin, vout)$ capacity 1. With this transformation, we now have a graph in which there is a single edge corresponding 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 disjoint paths sharing neither edges nor vertices, if they exist.

Grading:

(a) - Given a reasonable flow network to solve this problem, including nodes, edges, edge directions and their capacities: 15 pts.
(b) - Given the full credits if the algorithm is reasonable.

5 Circulation (20 points)

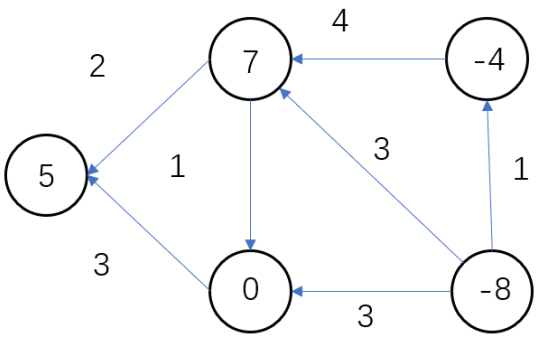
In the network below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.



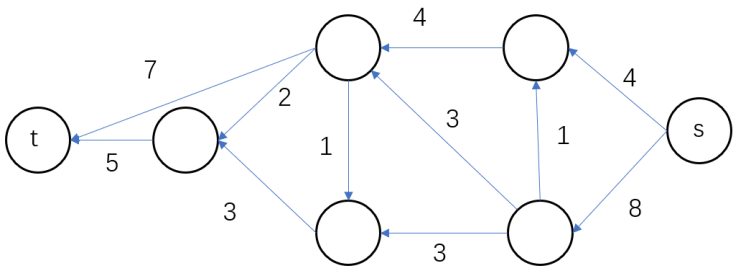
(a) Turn the circulation with lower bounds problem into a circulation problem without lower bounds. (8 pts)
(b) Turn the circulation with demands problem into the maximum flow problem. (8 pts)
(c) Does a feasible circulation exist? Explain your answer. (4 pts)

Solution:

(a)



(b)



(c) No, a feasible circulation doesn't exist. In the given original network, total demand values is 12, and the max flow is 10.

Grading:

(a) - There are total 12 values in the network-flow graph that needs to be written, 5 values (demands) on vertices and 7 values on edges (modified capacities). If a value is incorrect, deduct 0.5 pts.
- 2 pts will be deducted for adding a source and sink vertices and edges.

(b) - 2 pts for adding two super nodes, 1 for each node.
- 2 pts for connecting negative demand vertices to supply nodes and positive

demands to sink nodes, 0 if they are connected in opposite way.
- 4 pts for placing the capacities on the newly added edges between super nodes and the vertices, will lose 1 pts for each edge capacity mistake (maximum deduction here is 4 pts).
- Deduct 0.5 pts for any incorrect edge capacity value on the edges in the original network flow.

(c) - If the answer is no: 2 pts.
- If the explanation is reasonable: 2 pts.
- If you have answered yes and obtained a max-flow value 1.5 pts are awarded.

6 Installing Software (Not Graded)

Suppose that you have just bought a new computer and you want to install software on that. Specifically, two companies, which you can think of like Microsoft and Apple, are trying to sell their own copy of n different products, like Operation System. Spread Sheet, Web Browser. For each product i, $i \in \{1, 2, \ldots, n\}$, we have

- the price $p_i \geq 0$ that Microsoft charges and the price $p'_i \geq 0$ that Apple charges.
- the quality $q_i \geq 0$ of Microsoft version and the quality $q'_i \geq 0$ of Apple version.

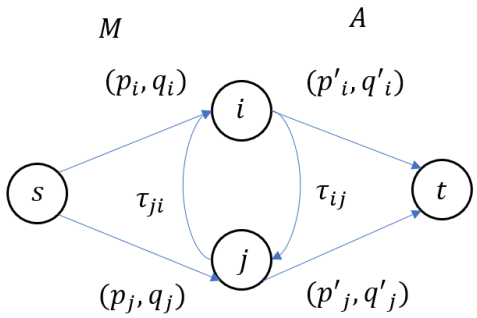
For example, Apple may provide a better Web Browser Safari, but Microsoft a better Word Processor. You want to assemble your favorite computer by installing exactly one copy of each of the n products, e.g. you want to buy one operating system, one Web Browser, one Word Processor, etc. However, you don't want to spend too much money on that. Therefore, your goal is to maximize the quality minus total price.
However, as you may know, the products of different companies may not be compatible. More concretely, for each product pair (i, j), we will suffer a penalty $\tau_{ij} \geq 0$ if we install product i of Microsoft and product j of Apple. Note that τ_{ij} may not be equal to τ_{ji} just because Apple's Safari does not work well on Microsoft Windows doesn't mean that Microsoft's Edge does not work well in Mac-OS. We assume that products are always compatible internally, which means that there is no penalty for installing two products from the same company. All pairwise penalties will be subtracted from the total quality of the system.

Your task is then to give a polynomial-time algorithm for computing which product i to purchase from which of the two companies (Apple and Microsoft) for all $i \in \{1, 2, \ldots, n\}$, to maximize the total system quality (including the penalties) minus the total price.

Solution:

Select a feasible set of products (M, A) maximizing the overall system quality minus the total price, $u(M, A)$. The idea of the solution is to reduce the problem to a minimum-cut in a graph.
Given the directed graph $G = (V, E)$.

- We add two special source and sink vertices, denoted by s and t respectively.
- For all the products $i \in V$, we add an edge $e_i = (s \to i)$ to the graph, setting its capacity, with lower bound p_i, to be $c(e_i) = (p_i, q_i)$.
- Similarly, we add an edge $e'_i = (j \to t)$ to the graph, setting its capacity, with lower bound p'_i, to be $c(e_i) = (p'_i, q'_i)$.
- Similarly, or each product pair (i, j) in that graph that are adjacent, we assign the cost τ_{ij} to the edges $(i \to j)$ and τ_{ji} to the edges $(j \to i)$.
- Let H denote the resulting graph.



Clearly, make a reduction of graph H, a cut in the graph can be interpreted as the value of $u(M, A)$ of the corresponding product sets M and A, since all the edges in the selection from nodes of M to nodes of A are contributing their separation value to the cut value. Thus, if we extend this idea to the directed graph G, the minimum-cut in the resulting graph would corresponds to the required products selection. Therefore, we can solve this problem, in polynomial time, by computing the max flow in the graph H.