Homework 8

Due: Friday, Oct 31, 11:59 PM PT

1. Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.
   For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a range parameter r—a client can only be connected to a base station that is within distance r. There is also a load parameter C_i for each base station—no more than C_i clients can be connected to the base station i. Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the range and load conditions in the previous paragraph. (20 points)

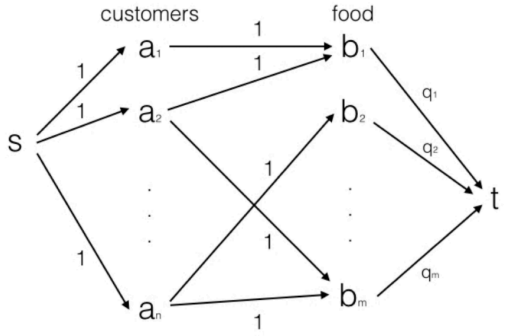
   Solution:
   Proof. We construct the following flow network. There is a node v_i for each client i, a node w_j for each base station j, and an edge (v_i, w_j) of capacity 1 if client i is within range of base station j. We then connect a super source s to each of the client nodes by an edge of capacity 1, and we connect each of the base station nodes to a super-sink t by an edge of capacity C_i. We claim that there is a feasible way to connect all clients to base stations if and only if there is an $s \rightarrow t$ flow of value n. If there is a feasible connection, then we send one unit of flow from s to t along each of the paths s, v_i, w_j, t, where client i is connected to base station j. This does not violate the capacity conditions, in particular on the edges (w_j, t), due to the load constraints. Conversely, if there is a flow of value n, then there is one with integer values. We connect client i to base station j if the edge (v_i, w_j) carries one unit of flow, and we observe that the capacity condition ensures that no base station is overloaded. The time complexity is the time required to solve a max-flow problem on a graph with $O(n + k)$ nodes and $O(nk)$ edges. This can be made to run in polynomial time with any algorithm that solves the max flow problem in polynomial time.

   Rubrics:
   - 10 points for the correct construction of the Network Flow graph. (-1 point for each incorrect edge weight/node. (source and sink can be reversed)
   - 2 points for the final algorithm (i.e., what problem to solve in the constructed network using which algorithm, and how to decide the final output)
   - 2 points for the correct forward and backward claims.
   - 3 points for the correct Forward Proof.
   - 3 points for the correct Backward Proof.

2. The Sushi Express food truck produces a large variety of different lunch menu items. Unfortunately, they can only produce their foods in limited quantities, so they often run

out of popular items, making customers sad. To minimize sadness, Sushi Express is implementing a sophisticated lunch-ordering system. Customers text in their acceptable choices before lunch time. Then they can use an algorithm to pre-assign lunches to customers. Customers who do not get one of their choices should receive a $10 voucher. Sushi Express would like to minimize the number of vouchers they give out.
Give an efficient algorithm for Sushi Express to assign lunches to customers. In general, suppose that, on a given day, Sushi Express has produced m types of food items $b_1, ..., b_m$, and the quantity of each type of food item b_j is exactly q_j. Suppose that n customers $a_1, ..., a_n$ text in their preferences, where each customer ai submits a set A_i of one or more acceptable lunch choices. The algorithm should assign each customer either one of his/her choices or a $10 voucher. It should minimize the number of vouchers. Furthermore, justify your algorithm. (20 points)

Solution:
Model this as a max flow problem. Define a flow network as follows. Include special vertices s and t as usual. Have a "first layer" of n vertices corresponding to the customers, and a "second layer" of m vertices corresponding to the foods. Include an edge from s to each customer, with capacity 1. Include an edge from customer a_i to food item b_j exactly if $j \in A_i$; this will also have capacity 1. Include an edge from food item bj to t with capacity q_j.



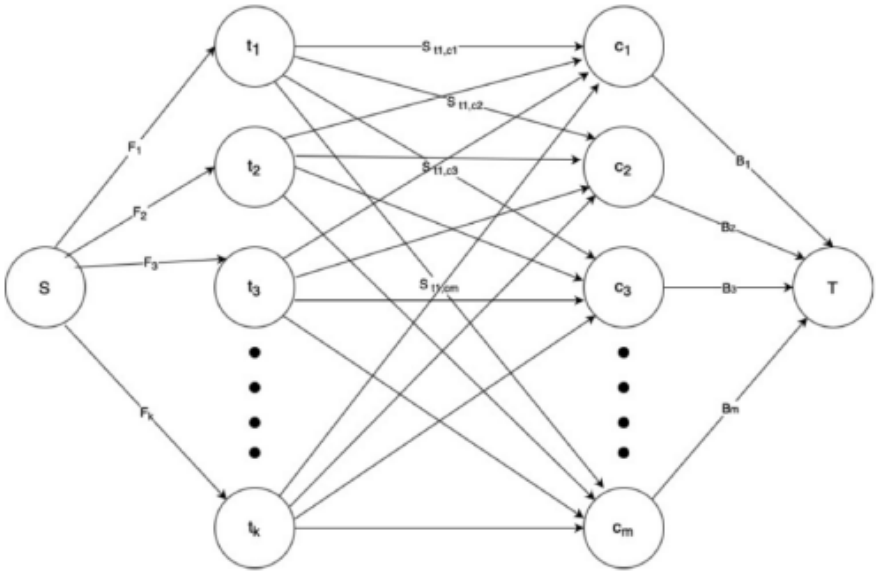
Proof. A flow f on this network yields an assignment of food items to customers that satisfies the customer and food quantity constraints. Specifically, for each customer ai, if some edge $f(a_i, b_j)$ has flow 1, then assign food item b_j to customer a_i. Note that each customer ai can get at most one food item, because a_i has an incoming flow of at most 1, so its outgoing flows must also total at most 1. Since all flows are integral, only one outgoing edge can have a positive flow. Also note that each food item b_j cannot be assigned to more than q_j customers, because b_j has outgoing flow at most q_j , so its incoming flows must also total at most q_j. Thus, the food assignment arising from flow f satisfies all the customer and food constraints. Conversely, any food assignment satisfying all these constraints corresponds directly to a flow through the network: Assign flow 1 to edge (a_i, b_j) exactly if customer a_1 gets assigned food item b_j. Assign flows to the edges from s and to t to achieve flow conservation. Moreover, a max flow through this network satisfies the maximum number of customers, because the definition of a max flow says that it maximizes the total flow out of s (which corresponds to the number of customers satisfied).

Rubric:

- 10 points for the correct construction of the Network Flow graph. (-1 point for each incorrect edge weight/node. (source and sink can be reversed)
- 2 points for the final algorithm (i.e., what problem to solve in the constructed network using which algorithm, and how to decide the final output)
- 2 points for the correct forward and backward claims.
- 3 points for the correct Forward Proof.
- 3 points for the correct Backward Proof.

3. A group of tourists 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 Indian 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. (10 points)
(b) Prove your algorithm is correct by making an if-and-only-if claim and proving it in both directions. (10 points)

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.
Recall, 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 the 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 any efficient Network Flow Algorithm, say Edmond Karp, 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 $\sum_k F_k$.

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 that max-flow value is $\sum_k F_k$.

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 $S \rightarrow t_k \rightarrow c_j \rightarrow T$. 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 tk 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 a 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 tk 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 tourists' requests for conversion while following all the constraints.

Rubric:

For part (a)

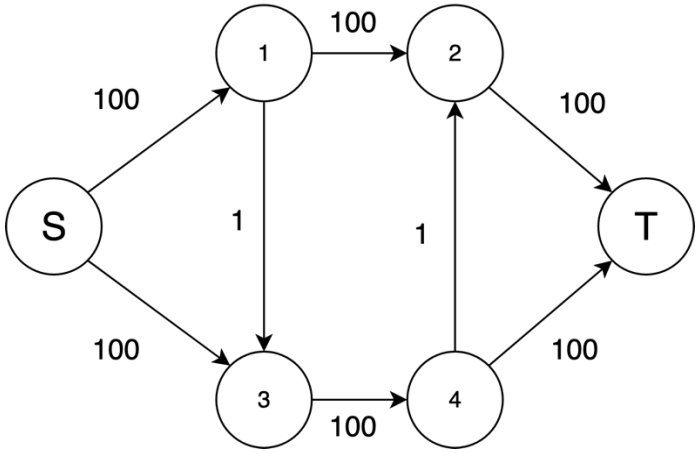
- 10 points for the correct construction of the Network Flow graph. (-1 point for each incorrect edge weight/node. (source and sink can be reversed)
- 2 points for the final algorithm (i.e., what problem to solve in the constructed network using which algorithm, and how to decide the final output)

For part (b)

- 2 points for the correct forward and backward claims.
- 3 points for the correct Forward Proof.
- 3 points for the correct Backward Proof.

Ungraded Problems:

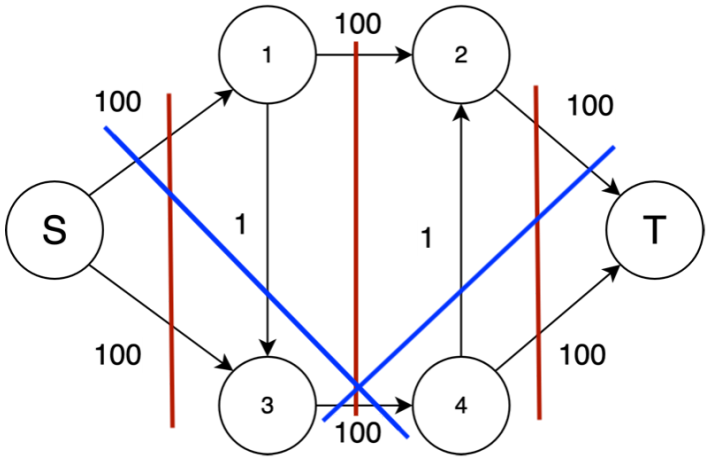
1. Based on the following graph, answer the following questions.



1. What is the max flow from S to T? (1 point)
2. Find all the min-cuts in the graph. What is the relation between min-cuts and the result from part 1? (2 points)
3. What is the minimum number of iterations required for the Ford–Fulkerson algorithm to find max flow in this graph? Explain how it happens. (2 points)
4. What is the maximum number of iterations required for the Ford–Fulkerson algorithm to find max flow in this graph? Explain how it happens. (3 points)
5. Based on the results from parts 3 and 4, justify the runtime complexity of the Ford–Fulkerson algorithm. (1 point)
6. How many iterations does Edmonds-Karp require? Does it have the same minimum and maximum number of iterations issue as the Ford–Fulkerson algorithm? (2 points)

Solution:

1. 200
2. There are 5 min cuts:



Relation: Min-cut is equal to max flow.

3. The minimum number of iterations happens when the Ford-Fulkerson algorithm chooses the two 100 paths first and doesn't pass through edges with a weight of 1. And after just two passes the algorithm terminates.
   Total number of iterations: 2 iterations

4. The maximum number of iterations happens when the Ford-Fulkerson algorithm constantly chooses the paths that involve the edges with the weight of 1, and as a result, only increments the flow by one at each pass, since after taking each of the edges with the weight of 1, they create a new augmenting path with weight 1 that can be taken again. This process can be repeated 200 times.
   Total number of iterations: 200 iterations

5. The runtime complexity of the Ford-Fulkerson algorithm is O(fE), where E is the number of edges and f is the value of the maximum flow, because, similar to what might happen is that at each pass the algorithm might take the suboptimal path and end up repeating the path finding process f times.

   The Edmonds-Karp algorithm solves this issue by finding paths using BFS instead of DFS, and as a result, it will always choose the shortest paths (in terms of number of edges) first. And therefore, it will always choose the first two hundred weight paths first

2. For a flow network with source S and sink T, determine if the following statements are true or false. For each statement, briefly explain your reasoning. (12 points)

   (a) There always exists a maximum flow that doesn't include a cycle containing positive flow.
   (b) If you have non-integer edge capacities, then you cannot have an integer max-flow value.

(c) Suppose the maximum s-t flow has value v(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 v(f) + 1.
(d) If all edge capacities are multiplied by a positive number k, then the min-cut remains unchanged.

Solution:
(a) True. We can always remove the flow from such a cycle and still get the same flow value.
(b) 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.
(c) False. Counter-Example: consider the 'diamond' graph as above, with all capacities, say 3 (instead of 0.5 in the previous example). Then, 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.
(d) True. The value of every cut gets multiplied by k, thus the relative order of all cuts remains the same, hence any min-cut remains a min-cut.

3. 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. (20 points)

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 uh 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, otherwise, v_i connects to the node w_j where $i \in S_j$, with edge capacity a_i in either case. (Notice that because the constraint sets are disjoint, each person's node connects to at most one constraint node.) Finally,

each node wj connects to the sink t with capacity tj.

Compute max-flow in this network using Edmond-Karp. Output 'Yes' if and only if max-flow value is $\sum_h r_h$. 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. Hence, using Edmond-karp to compute max flow in this network ensures a polynomial running time.

We now prove the claim that this network has max 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 constraints.

• 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^h_i units of work on task h, meeting all constraints, then we send x^h_i 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 (wj , 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 $\sum_h r_h$.