COMP 360 - Fall 2015 - Assignment 1

Due: 6pm Sept 29th.

---

1. (5 points) Recall that for every flow f and every cut (A, B), we have $\text{val}(f) = f^{out}(A) - f^{in}(A)$ where $f^{out}(A) = \sum_{\substack{uv\in E \\ u\in A, v\in B}} f(uv)$ and $f^{in}(A) = \sum_{\substack{uv\in E \\ u\in B, v\in A}} f(uv)$. Use this fact to show that $\text{val}(f)$ is equal to the total flow on the edges going into the sink.

   Solution: Take $B = \{t\}$ and $A = V - \{t\}$. Then by the above fact

   $$\text{val}(f) = f^{out}(A) - f^{in}(A) = f^{out}(A) = f^{in}(t).$$

2. (10 Points) Five types of packages are to be delivered by four trucks. There are three packages of each type. The capacities of the three trucks are 5, 4, 4, and 3 packages respectively. Set up a maximum flow problem that can be used to determine whether the packages can be loaded so that no truck carries two packages of the same type.

   Solution: Create the following flow network. Let vertices be s (the source), t_i, for $i = 1, \ldots, 5$ (corresponding to types), m_i, $i = 1, \ldots, 4$ (corresponding to trucks). Join s with each t_i with an edge of capacity equal to 3, join every t_i with m_j with an edge of capacity 1 (this makes sure no truck loads more than one of the same type), then join every m_i with t with its given capacity $(5, 4, 4, 3$ respectively). Now if we run the max-flow algorithm and it finds a flow of value $15 = 5 \times 3$, then it is easy to see that the desired arrangement is possible.

3. (10 points) Prove that if an edge e goes from A to B in a minimum cut (A, B), then *every maximum* flow f uses the full capacity of e, that is $f(e) = c_e$.

   Solution: Let (A, B) be a minimum cut. Consider any maximum flow f. By Fact from Problem 1, we have that $f^{out}(A) - f^{in}(A) = c(A, B)$ (since (A, B) is a min-cut). But now if there is an edge $e \in (A, B)$ for which $c_e > f(e)$, it would directly follow that

   $$f^{out}(A) - f^{in}(A) := \sum_{\substack{uv\in E(A,B) \\ u\in A, v\in B}} f(uv) - \sum_{\substack{uv\in E(A,B) \\ u\in B, v\in A}} f(uv) < c(A, B),$$

   which is a contradiction.

4. (15 Points) Is there a polynomial time algorithm that decides whether a flow network has a *unique* minimum cut?

   Solution: Yes, here is one such algorithm: Run the Ford-Fulkerson algorithm to find a mininum cut (A_0, B_0). Then for every edge e in (A_0, B_0), increase the capacity of that edge and compute the value of the max-flow again. If the value does not increase, then it means that there is another min-cut that there was another min-cut in the original graph that did not include this edge, and hence the min-cut was not unique. If for all the edges in (A_0, B_0) the value of the max-flow increases, then the min-cut is unique. Clearly this algorithm is polynomial (we run FF at most # of edges in (A_0, B_0) times).

5. (15 Points) Consider a flow network.

- We say that a node v is *upstream* if for all minimum cuts (A, B), we have $v \in A$.
- We say that a node v is *downstream* if for all minimum cuts (A, B), we have $v \in B$.
- We say that a node v is *central* if it is neither upstream nor downstream.

Give an algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a single maximum flow. You must prove that your algorithm is correct.

Solution: First let us design an auxiliary algorithm A which computes the minimum cut (S, T) where S if of minimum cardinality. Run the Ford-Fulkerson algorithm to find the maximum flow and look at the final residual graph G_f. On this graph G_f run DFS (depth first search) from the source s. Suppose DFS observed the vertex set S (it is clear that $S \neq V$, at least $t \notin S$), we claim that $(S, V - S)$ is the desired cut. Indeed, notice that for every vertex in S which is a leaf found by BFS, there is an edge of full capacity that comes in from some vertex of $V - S$ and these are the only non-zero capacity edges between S and $V - S$ (if there was an edge going out from the leaves, the BFS would not have stopped). Hence, indeed this cut is a minimum one. And S has the minimum cardinality because of the property of DFS - we found a cut with S that has the first vertices in the path from s from which only saturated edges are going further. Algorithm A clearly is polynomial.

Now run algorithm A twice. First on our original flow network, say the output cut is $(S, V - S)$. We claim that S is the set of upstream vertices. Run it second time on the "reversed network" (i.e. all edges are reversed and t becomes the source) and obtain the cut $(T, V - T)$. T is the set of downstream vertices. It is not hard to show that all the remaining vertices (i.e. $V - S - T$) are central.

6. (15 Points) Let A be an $n \times n$ matrix with each entry equal to either 0 or 1. Let a_{ij} denote the entry in the row i and column j. Entries a_{ii} are called the *diagonal* entries. We call A re-arrangeable if it is possible to re-arrange the rows and then re-arrange the columns so that after the rearrangements all the diagonals entries of the matrix are equal to 1.

   - Give an example of an $n \times n$ matrix that is not rearrangeable but for which at least one entry in each row and each column is equal to 1.
   - Give a polynomial algorithm based on Max-Flow that determines whether a matrix M with $0 - 1$ entries is re-arrangeable.

   Solution: For example for every n one can take the matrix to be the following - let the first row be $11 \ldots 10$ and all other rows $00 \ldots 01$. It is easy to see that this is not re-arrangeable.

   This problem can be deduced to finding a perfect matching in a bipartite graph. Indeed, construct the following bipartite graph (A, B). Let A be the vertices corresponding to rows and B be the vertices corresponding to columns. Join $a \in A$ and $b \in B$ with an edge only when the corresponding entry in the matrix is 1. It is easy to see that matrix will be re-arrangeable if and only if this bipartite graph has a perfect matching. And how to find a perfect matching in a bipartite graph using flows you have seen during the class (Lecture 4).

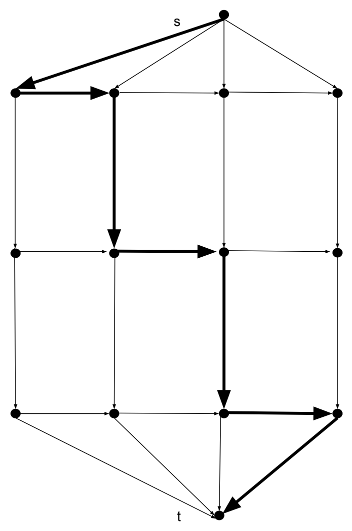
7. (15 Points) Consider the variant of the maximum flow problem where every node v also has an integer capacity $c_v \geq 0$. We are interested in finding the maximum flow as before, but now with the extra restriction that $f^{\text{in}}(v) \leq c_v$ for every node v. Solve this problem using the original maximum flow problem.

Solution: Split every vertex into two vertices v^{in} and v^{out} and put a directed edge from v^{in} to v^{out} with capacity c_v. The input edges of the original vertex v now are pointed to v^{in} and the output edges of the original v leave v^{out}.

8. (10 Points) The goal of this exercise is to show that if we modify the Ford-Fulkerson algorithm so that it does not decrease the flow on any of the edges (i.e. we do not add the opposite edges to the residual graph), then the algorithm might perform very poorly.

   For every positive integer m, construct an example of a flow network whose maximum flow is equal to m, but the modified Ford-Fulkerson algorithm terminates after finding a flow of value 1. In other words, in your flow network, there is an augmenting path with bottleneck 1 such that after augmenting this path there are no augmenting paths left that do not push back flow on any edges.

Solution: Construct the following network. Take an $(m-1) \times m$ grid and join s to the first row vertices and join the last row vertices to t. Let all the edges to have capacity one and all vertical edges to be directed down and horizontal ones to the right. Now if FF algorithm picks at the very first step the diagonal path going from s through the left-most corner to the bottom right-most corner and ending at t (as shown in the figure on the right, for $m = 4$) then there is no $s-t$ path anymore, thus the algorithm stops with max flow being 1 while in fact it is m (on can send all m value using m disjoint columns of the grid).



9. (15 Points) Show that there is always a sequence of at most m augmenting paths that leads to maximum flow where m is the number of edges in the flow network. (Hint: Use the maximum flow to find the paths).

   Solution: To prove this, we apply the following algorithm to decompose a maximum flow f into a set of flow-paths:

   - Repeat the following two steps until there is no such s, t-path:
   - Find an s, t-path P with positive flow, i.e., $f(e) > 0$ for all $e \in P$.
   - Let Δ be the minimum value of the flow f on edges of P. Decrease the flow f on each edge $e \in P$ by Δ.

   In every iteration the value of f on at least one edge becomes 0. Hence the number of iterations is at most m. Using these paths as augmenting paths (with augmenting value Δ) leads to the desired result.