Design and Analysis of Algorithms **out:** March 5, 2015

Problem Set 5

Lecturer: Daniel A. Spielman **due: 2:30** PM, March 26, 2015

Instructions. Read Carefully

1. Make sure that your yale **netid** appears on **every** page of your problem set, and that it is written clearly.

2. The problem sets are due at the beginning of class. Solution sets will be provided in class. For this reason, late problem sets are not accepted.

3. You will **hand in each problem separately**, so make sure that they are not stapled together and that no problem appears on the back of another. There will be 4 boxes in the front of the class, one for each problem. Put each problem in the correct box.

4. Alternatively, you may submit the problem set electronically via the Classes server. If you do so, submit it as a **pdf** and make sure that your **netid** appears on **every** page. **Each problem must be in its own document.**

5. Cite any resources that you use, other than the textbook, TA and instructor.

6. You **must not** search the web for solutions to similar problems given out in other classes.

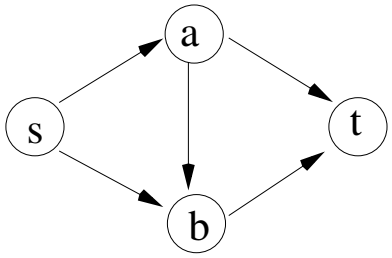
Collaboration Policy

You must write the solutions to the problems independently and in your own words.

However, you are allowed to discuss the problem sets in small groups of no more than 4 students. To ensure that you understand the solutions you submit, you are forbidden from taking written notes during your discussions. You must list at the top of the problem set everyone (other than course staff) with whom you have discussed the problem set. Failure to list people with whom you have discussed a problem set is considered a violation of academic honesty.

Problem 1: The failure of a greedy algorithm for Maximum Flow

When explaining the Ford-Fulkerson algorithm, I often gave examples like the following to show why the use of residual graphs is necessary.



Imagine that all edge have capacity one. In this case the maximum flow from s to t is 2.

Now, imagine an algorithm that does not use residual graphs, but which just tries to find s-t paths in the original graph that do not use saturated[1] edges. If this algorithm first finds the path $s - a - b - t$ and flows one unit along it, then there will no longer be any s-t paths with unsaturated edges. However, this is not a maximum flow. That's why the Ford-Fulkerson algorithm looks for s-t flows in the residual graph.

This example might suggest a simpler solution: always take the shortest s-t path in G consisting of unsaturated edges. This greedy path selection technique does find the maximum flow on this example graph. For concreteness, I state the proposed algorithm in pseudo-code. The algorithm will keep track of the set of unsaturated edges, called F.

0. Set $f(e) = 0$ for all $e \in E$. Set $F = E$.
1. While there is an s-t path in the graph (V, F)
   a. Let P be the shortest s-t path in (V, F) (breaking ties arbitrarily).
   b. Let $\mathsf{bot}(P)$ be the minimum over $e \in P$ of $c(e) - f(e)$.
   c. For all $e \in P$, increase $f(e)$ by $\mathsf{bot}(P)$.
   d. For every edge $e \in P$ for which $f(e) = c(e)$, remove e from F.

You will now show that this is a bad algorithm.

a. Give an example of a graph on which this algorithm does not find the maximum flow. Explain why. [5 Points]

[1] An edge is *saturated* if its flow equals its capacity. It is *unsaturated* otherwise.

b. Prove that this algorithm can be arbitrarily bad, even if all edge capacities are 1. That is, for each integer $k > 1$ prove that there is a graph G_k with all edge capacities 1 for which the flow found by this algorithm is less than the maximum flow divided by k. Explain why your examples work. [5 Points]

Problem 2: Flow with supplies and demands on trees

There are many variations of maximum flow problems. We will now consider a variation that has multiple vertices through which flow will enter and exit the graph, instead of just one source and sink. In fact, we will assume that flow can enter or exit through every vertex. We represent the flow that must enter or exit the vertices through a *demand vector* $d : V \to \mathbb{R}$. When $d(a) > 0$, it means that $d(a)$ units of flow must exit the graph from vertex a. Such vertices are sinks. When $d(a) < 0$, it means that $d(a)$ units of flow must enter the graph through vertex a. Vertices with $d(a) < 0$ are the sources of flow. In order for the demands to be satisfiable, it must be the case that $\sum_{a \in V} d(a) = 0$.

We will consider the problem of determining whether or not an *undirected* graph has a valid flow that satisfies given demands. That is, the input will be an undirected graph $G = (V, E, c)$ where $c : E \to \mathbb{R}^+$ gives the capacity of each edge, along with a demand vector d. The problem is to find a valid flow f that satisfies the demands. The flow satisfies the demands if for every vertex a,

$$\sum_{(b,a) \in E} f(b, a) = d(a) \qquad \text{and} \qquad \sum_{(a,b) \in E} f(a, b) = -d(a). \tag{1}$$

I am doing something a little unusual by talking about undirected graphs. The easiest way to deal with this is to treat each undirected edge as a pair of directed edges going in opposite directions with the same capacities. You are free to do so. Alternatively, you can think of an undirected graph as a graph in which flow can move either direction on every edge. To be valid, the flow on each edge must be at most its capacity. However, you will need to be a little more careful when writing equations like (1) if you think about it this way, since the undirected edge (a, b) is the same as the undirected edge (b, a). I'll let you decide how to deal with this. The way that I usually deal with it is to assign a direction to every edge, and then allow negative flow. That is, I restrict the flow on edge (a, b) to be between $-c(a, b)$ and $c(a, b)$.

Now, here's your problem. You must design an algorithm that finds a valid flow that satisfies the demands, if such a flow exists. If such a flow does not exist, then you should return "no such flow". To make the problem easier, I only ask that you solve the problem in the case that G is a **tree**. To make the problem harder, I insist that your algorithm run in **linear time**. Give the algorithm, explain why it is correct, and explain why it runs in linear time.

Hint: When designing algorithms on trees, it helps to organize them from top to bottom. That is, we proclaim one node of the tree to be the root. You can pick this node arbitrarily. The neighbors of the root are called its "children". The root is called the "parent" of those nodes, and their other neighbors are their "children". So, every node other than the root has one parent, and every node other than the leaves have children. If you like, you may assume that the tree is given to you in this form. But, of course, you can easily put any tree in this form by using a breadth first search.

Problem 3: Replacing Capacities with Demands

In the previous problem I quickly claimed that one can transform a flow problem on undirected graphs into one on directed graphs by replacing each undirected edge with two directed edges. In this problem, you will show that a similar transformation allows you to use demands to eliminte the need for edge capacities. That is, you can make all the capacities infinite.

To make this precise, we will consider the decision version of the flow problem instead of the maximum flow problem. In the decision version, one is given as input the usual directed graph with capacities, $G = (V, E, c)$, vertices s and t, and a target flow F. The problem is to determine whether or not there is a valid flow of value at least F from s to t in G. If there is not, your algorithm can output "no". If there is, your algorithm should return "yes".

Your job is to turn this into the decision version of the demand flow problem. In that problem, you are given a directed graph $\widehat{G} = (\widehat{V}, \widehat{E})$ and a demand vector d. The answer is "yes" if there is a flow in $\widehat{G}$ that satisfies demands d. The answer is "no" otherwise. You will notice that I have not specified any capacities for edges in $\widehat{G}$. That is because the edges in $\widehat{G}$ are allowed to carry any nonnegative amount of flow.

When I say that you should transform the decision version of the maximum flow problem into the decision version of the demand flow problem, I mean that you should describe a linear-time algorithm that takes as input a weighted directed graph G, vertices s and t, and a target flow F, and outputs a directed graph $\widehat{G}$ and a demand vector d so that the answer to the demand flow problem is "yes" if and only if the answer to the maximum flow problem is "yes". You are welcome to describe this algorithm in any way reasonable. I think that pseudo-code would be overkill.

I am not asking for a linear time algorithm to be difficult. Rather, I just want to be sure that your first step is not "solve the maximum flow problem". I want you to make a very simple transformation of the graph. If you prefer, you can describe an algorithm that runs in time $O(n \log n)$.

The idea here is that you are going to construct $\widehat{G}$ by making small changes to G: for

example, you might replace each edge in G by a couple of edges and vertices in $\widehat{G}$. I suggest that you begin by considering the case of a graph G consisting of just one edge and two vertices. When we converted an undirected graph to a directed one, we replaced each undirected edge with two directed edges. This is similar, but you will need to introduce new vertices.

You should describe the algorithm that makes the transformation, and explain why it is correct.

Problem 4: The Residual Graph of Maximum Bipartite Matching

You will now prove a property of the residual graph that we obtain when we solve the maximum bipartite matching problem by a maximum flow computation. The bipartite graph will have two sets of vertices, L and R (for left and right), and all edges in the bipartite graph will have capacity 1 and will be directed from L to R.

In our reduction we turn this into a larger graph by adding two additional vertices, s and t. The vertex s has a capacity-1 edge directed out of it towards every vertex in L, and every vertex in R has a capacity-1 edge directed to t. Call this graph G.

Let f be a maximum integral flow in G. That is, f is a maximum flow in G and the flow on every edge is an integer.

Let G_f be the residual graph of G with respect to f, and let A be the set of vertices reachable in G_f from s. Let B be the set of vertices that are not reachable from s in G_f.

Let $a_L \in A \cap L$ and let $b_R \in B \cap R$. Prove that (a_L, b_R) is not an edge in G (Note: this is different from proving that it is not an edge in G_f).