If only I had a child, 
curly-haired and dark 
to take by his small hand 
as we slowly walked through the park. 
A child. 
Uri I’d call him, 
a name clear and mild, 
a fragment of light. 
“Uri!” 
I’d call him, 
my small, dark child. 
Still, like Rachel 
the Mother, I mourn, 
like Hannah pray 
for the unborn, 
and wait, still wait 
for my child. 

- Barren, Rachel
Required Problems

1. **The good, the bad, and the middle.**
   
   **[10 Points]**

   Suppose you’re looking at a flow network $G$ with source $s$ and sink $t$, and you want to be
   able to express something like the following intuitive notion: Some nodes are clearly on the
   “source side” of the main bottlenecks; some nodes are clearly on the “sink side” of the main
   bottlenecks; and some nodes are in the middle. However, $G$ can have many minimum cuts, so
   we have to be careful in how we try making this idea precise.

   Here’s one way to divide the nodes of $G$ into three categories of this sort.

   - We say a node $v$ is *upstream* if, for all minimum $s$-$t$ cuts $(A, B)$, we have $v \in A$ — that
     is, $v$ lies on the source side of every minimum cut.
   - We say a node $v$ is *downstream* if, for all minimum $s$-$t$ cuts $(A, B)$, we have $v \in B$ — that
     is, $v$ lies on the sink side of every minimum cut.
   - We say a node $v$ is *central* if it is neither upstream nor downstream; there is at least one
     minimum $s$-$t$ cut $(A, B)$ for which $v \in A$, and at least one minimum $s$-$t$ cut $(A', B')$ for
     which $v \in B'$.

   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.

2. **Ad hoc networks**
   
   **[20 Points]**

   Ad hoc networks are made up of low-powered wireless devices, have been proposed for situations
   like natural disasters in which the coordinators of a rescue effort might want to monitor
   conditions in a hard-to-reach area. The idea is that a large collection of these wireless devices
   could be dropped into such an area from an airplane and then configured into a functioning
   network.

   Note that we’re talking about (a) relatively inexpensive devices that are (b) being dropped
   from an airplane into (c) dangerous territory; and for the combination of reasons (a), (b),
   and (c), it becomes necessary to include provisions for dealing with the failure of a reasonable
   number of the nodes.

   We’d like it to be the case that if one of the devices $v$ detects that it is in danger of failing,
   it should transmit a representation of its current state to some other device in the network.
   Each device has a limited transmitting range – say it can communicate with other devices
   that lie within $d$ meters of it. Moreover, since we don’t want it to try transmitting its state to
   a device that has already failed, we should include some redundancy: A device $v$ should have
   a set of $k$ other devices that it can potentially contact, each within $d$ meters of it. We’ll call
   this a back-up set for device $v$.

   (a) Suppose you’re given a set of $n$ wireless devices, with positions represented by an $(x, y)$
   coordinate pair for each. Design an algorithm that determines whether it is possible to
   choose a back-up set for each device (i.e., $k$ other devices, each within $d$ meters), with
   the further property that, for some parameter $b$, no device appears in the back-up set of
more than \( b \) other devices. The algorithm should output the back-up sets themselves, provided they can be found.

(b) The idea that, for each pair of devices \( v \) and \( w \), there’s a strict dichotomy between being “in range” or “out of range” is a simplified abstraction. More accurately, there’s a power decay function \( f(\cdot) \) that specifies, for a pair of devices at distance \( \delta \), the signal strength \( f(\delta) \) that they’ll be able to achieve on their wireless connection. (We’ll assume that \( f(\delta) \) decreases with increasing \( \delta \).)

We might want to build this into our notion of back-up sets as follows: among the \( k \) devices in the back-up set of \( v \), there should be at least one that can be reached with very high signal strength, at least one other that can be reached with moderately high signal strength, and so forth. More concretely, we have values \( p_1 \geq p_2 \geq \cdots \geq p_k \), so that if the back-up set for \( v \) consists of devices at distances \( d_1 \leq d_2 \leq \cdots \leq d_k \), then we should have \( f(d_j) \geq p_j \) for each \( j \).

Give an algorithm that determines whether it is possible to choose a back-up set for each device subject to this more detailed condition, still requiring that no device should appear in the back-up set of more than \( b \) other devices. Again, the algorithm should output the back-up sets themselves, provided they can be found.

3. [10 Points]

Give a polynomial-time algorithm for the following minimization analogue of the Maximum-Flow Problem. You are given a directed graph \( G = (V, E) \), with a source \( s \in V \) and sink \( t \in V \), and numbers (capacities) \( \ell(v, w) \) for each edge \((v, w) \in E\). We define a flow \( f \), and the value of a flow, as usual, requiring that all nodes except \( s \) and \( t \) satisfy flow conservation. However, the given numbers are lower bounds on edge flow — that is, they require that \( f(v, w) \geq \ell(v, w) \) for every edge \((v, w) \in E\), and there is no upper bound on flow values on edges.

(a) Give a polynomial-time algorithm that finds a feasible flow of minimum possible values.

(b) Prove an analogue of the Max-Flow Min-Cut Theorem for this problem (i.e., does min-flow = max-cut?).

4. [20 Points]

You are trying to solve a circulation problem, but it is not feasible. The problem has demands, but no capacity limits on the edges. More formally, there is a graph \( G = (V, E) \), and demands \( d_v \) for each node \( v \) (satisfying \( \sum_{v \in V} d_v = 0 \)), and the problem is to decide if there is a flow \( f \) such that \( f(e) \geq 0 \) and \( f^{\text{in}}(v) - f^{\text{out}}(v) = d_v \) for all nodes \( v \in V \). Note that this problem can be solved via the circulation algorithm seen in class by setting \( c_e = +\infty \) for all edges \( e \in E \). (Alternately, it is enough to set \( c_e \) to be an extremely large number for each edge — say, larger than the total of all positive demands \( d_v \) in the graph.)

You want to fix up the graph to make the problem feasible, so it would be very useful to know why the problem is not feasible as it stands now. On a closer look, you see that there is a subset \( U \) of nodes such that there is no edge into \( U \), and yet \( \sum_{v \in U} d_v > 0 \). You quickly realize that the existence of such a set immediately implies that the flow cannot exist: The set \( U \) has a positive total demand, and so needs incoming flow, and yet \( U \) has no edges into it. In trying to evaluate how far the problem is from being solvable, you wonder how big the demand of a set with no incoming edges can be.

Give a polynomial-time algorithm to find a subset \( S \subset V \) of nodes such that there is no edge into \( S \) and for which \( \sum_{v \in S} d_v \) is as large as possible subject to this condition.
5. [20 Points]

Consider an assignment problem where we have a set of \( n \) stations that can provide service, and there is a set of \( k \) requests for service. Say, for example, that the stations are cell towers and the requests are cell phones. Each request can be served by a given set of stations. The problem so far can be represented by a bipartite graph \( G \): one side is the stations, the other the customers, and there is an edge \((x, y)\) between customer \( x \) and station \( y \) if customer \( x \) can be served from station \( y \). Assume that each station can serve at most one customer. Using a max-flow computation, we can decide whether or not all customers can be served, or can get an assignment of a subset of customers to stations maximizing the number of served customers.

Here we consider a version of the problem with an addition complication: Each customer offers a different amount of money for the service. Let \( U \) be the set of customers, and assume that customer \( x \in U \) is willing to pay \( v_x \geq 0 \) for being served. Now the goal is to find a subset \( X \subseteq U \) maximizing \( \sum_{x \in X} v_x \) such that there is an assignment of the customers in \( X \) to stations.

Consider the following greedy approach. We process customers in order of decreasing value (breaking ties arbitrarily). When considering customer \( x \) the algorithm will either “promise” service to \( x \) or reject \( x \) in the following greedy fashion. Let \( X \) be the set of customers that so far have been promised service. We add \( x \) to the set \( X \) if and only if there is a way to assign \( X \cup \{x\} \) to servers, and we reject \( x \) otherwise. Note that rejected customers will not be considered later. (This is viewed as an advantage: If we need to reject a high-paying customer, at least we can tell him/her early.) However, we do not assign accepting customers to servers in a greedy fashion: we only fix the assignment after the set of accepted customers is fixed. Does this greedy approach produce an optimal set of customers? Prove that it does, or provide a counterexample.

6. [20 Points]

Some friends of yours have grown tired of the game “Six Degrees of Kevin Bacon” (after all, they ask, isn’t it just breadth-first search?) and decide to invent a game with a little more punch, algorithmically speaking. Here’s how it works.

You start with a set \( X \) of \( n \) actresses and a set \( Y \) of \( n \) actors, and two players \( P_0 \) and \( P_1 \). Player \( P_0 \) names an actress \( x_1 \in X \), player \( P_1 \) names an actor \( y_1 \) who has appeared in a movie with \( x_1 \), player \( P_0 \) names an actress \( x_2 \) who has appeared in a movie with \( y_1 \), and so on. Thus, \( P_0 \) and \( P_1 \) collectively generate a sequence \( x_1, y_1, x_2, y_2, \ldots \) such that each actor/actress in the sequence has costarred with the actress/actor immediately preceding. A player \( P_i \) \((i = 0, 1)\) loses when it is \( P_i \)'s turn to move, and he/she cannot name a member of his/her set who hasn’t been named before.

Suppose you are given a specific pair of such sets \( X \) and \( Y \), with complete information on who has appeared in a movie with whom. A strategy for \( P_i \), in our setting, is an algorithm that takes a current sequence \( x_1, y_1, x_2, y_2, \ldots \) and generates a legal next move for \( P_i \) (assuming it’s \( P_i \)'s turn to move). Give a polynomial-time algorithm that decides which of the two players can force a win, in a particular instance of this game.