Problem Set 6

This problem set is due **AT THE BEGINNING OF** class on *April 25, 2002*.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material and will be useful in solving the assessed problems. You are responsible for material covered by the exercises. Each problem is to be done on a separate sheet (or sheets) of paper. Mark the top of each sheet with your name, 6.046J/18.410J, the problem number, your TA and recitation section, the date, and the names of any students with whom you collaborated.

Problem 6-1. Six Degrees of Separation from Kevin Bacon

A *diameter* of a (possibly directed) graph $G = (V, E)$ is the minimum value Δ such that for all $p, q \in V$, the distance from p to q is at most Δ (the distance is simply the number of edges along the shortest path). If the graph is not connected (or is directed and not strongly connected), then $\Delta = \infty$. If the diameter is finite, then we know that any two nodes are within distance Δ from each other, which is a useful thing to know.

From the lecture we know that a diameter of a graph can be computed in time $O(|V||E|)$. However, if the graph is large, this is too costly. The goal of this problem is to design a faster (but approximate) algorithm for this problem,

(a) Assume that G is undirected. Consider an algorithm which computes the shortest path distances $D(u, v)$ from an arbitrary vertex u to all other $v \in V$. Then, the algorithm reports

$$\tilde{\Delta} = \max_{v \in V} D(u, v)$$

as an approximation to the diameter Δ. Show that $\tilde{\Delta} \leq \Delta \leq 2\tilde{\Delta}$, and that $\tilde{\Delta}$ can be computed in time $O(|E|)$.

Solution:
According to the definition of $\Delta = \max_{p,q \in V}\{D(p, q)\}$. That is Δ is the length of some maximal shortest path $p \leadsto q$ in G.

Since Δ is the length of a a maximal shortest path in G and $\tilde{\Delta}$ is the length of *some* shortest path in G, we know that $\tilde{\Delta} \leq \Delta$.

To see the other inequality, consider the shortest paths from u to p and from u to q, where u is the arbitrary node selected by our algorithm. We know that $D(u, p) = D(p, u) \leq \tilde{\Delta}$ and that $D(u, q) \leq \tilde{\Delta}$. Since, $D(p, q)$ is the length of a *shortest* path from p to q, $D(p, q)$ must be no more that the length of $p \leadsto u \leadsto q = D(p, u) + D(u, q)$. Therefore, $\Delta \leq D(p, u) + D(u, q) \leq 2\tilde{\Delta}$, as needed.

We can compute $\tilde{\Delta}$ using a very slight modification of breadth-first search. BFS has running time $O(|V| + |E|)$. If $|V| < |E|, O(|V| + |E|) = O(|E|)$. So we are done. If $|V| \geq |E|$ then the graph cannot be connected. We know that if G is unconnected, then $\Delta = \infty$. So we can short circuit BFS and simply return $\Delta = \infty$ in constant time.

(b) Modify the algorithm and result from part (a) to apply to *directed* graphs.

Solution:

Remember that G^T is the graph G with its edges reversed. Out new algorithm runs BFS to compute the shortest path distances $D(u, v)$ from an arbitrary vertex u to all other $v \in V$ for G. Then we compute G^T and run BFS to compute all the shortest path distances $D^T(u, v)$ from the same vertex u to all other $v \in V$ for G^T. The algorithm reports

$$\tilde{\Delta} = \max\{\max_{v \in V}\{D(u, v), \max_{v \in V}\{D(v, u)\}$$

as the approximation of the diameter Δ.

Our argument for correctness for the previous part can be used nearly unchanged once we notice that $D^T(u, v) = D(v, u)$.

As before, since Δ is the length of a a maximal shortest path in G and $\tilde{\Delta}$ is the length of *some* shortest path in G, we know that $\tilde{\Delta} \leq \Delta$.

To show that $\Delta \leq 2\tilde{\Delta}$ consider a maximal shortest paths $p \rightsquigarrow q$ in G and the shortest paths from u to p and from u to q, where u is the arbitrary node selected by our algorithm. We know that $D(u, p) = D^T(p, u) \leq \tilde{\Delta}$ and that $D(u, q) \leq \tilde{\Delta}$. Since, $D(p, q)$ is the length of a *shortest* path from p to q, $D(p, q)$ must be no more that the length of $p \rightsquigarrow u \rightsquigarrow q = D(p, u) + D(u, q)$. Therefore, $\Delta \leq D(p, u) + D(u, q) \leq 2\tilde{\Delta}$, as needed.

By the same arguments as in the previous part, we can compute G^T in $O(|E|)$ time and run each of the BFS in $O(|E|)$. So our new algorithm also runs in $O(|E|)$.

(c) Go to the web page `http://www.cs.virginia.edu/oracle/` and click on `Bacon numbers`. Based on the just acquired wisdom and the information stored at the web page, give an estimation for the diameter of the *Bacon graph*, defined over the set of all actors reachable from Kevin Bacon (see the web page for details).

Solution:
Since the longest path length from Kevin to any of the other 506244 (reachable) actors in the Internet Movie Database (`imdb.com`) has length 10, we apply the given algorithm to assert that the diameter of the *Bacon graph* to be between 10 and 20.

Problem 6-2. Shortest Path With Small Weights

You are given a weighted graph, where the weights are integers in the range from $\{1 \ldots 10\}$. Give a linear time algorithm for finding the shortest path from a given node u to a given node v in such a graph.

Solution:
Given weighted graph $G = (V, E)$. Construct graph G' such that every edge (u, v) is extended to be $w(u, v)$ edges. So an edge with weight 3 in G will turn into a path of length 3 in G'.

Formally $G' = (V', E')$ is the graph such that

- $V \subseteq V'$
- If $(u, v) \in E$ with weight 1 then $(u, v) \in E'$
- If $w > 1$ then vertices $uv_1, uv_2, \ldots, uv_{w-1} \in V'$ and edges $(u, uv_1), (uv_1, uv_2), \ldots, (uv_{w-1}, v) \in E'$.

Now run breadth-first search on G' to find the shortest paths in G'. Ignore all paths starting at vertices not in V. Since all edge weights are integers in the range from $\{1 \ldots 10\}$, $10|V| \geq |V'|$ and $10|E| \geq |E'|$. Thus, running BFS on G' takes time $\Theta(|V| + |E|)$.

Problem 6-3. Paintball^{TM}

You and your friends thought that playing Paintball^{TM} over a network of platforms situated in the jungle would be a good way to forget about 6.046 for a few hours. No such luck! The platforms are connected by rickety bridges, each of which has an independent failure probability. As every Paintballer^{TM} knows, the goal of each player to get from platform s to platform t in one piece in order to gain a better defensive or offensive position. You've created a graph with each platform as a node and each bridge as an edge in order to better analyze your situation.

Give a polynomial time algorithm to find the path from node s to node t with minimum failure probability.

Solution:
We assume the probabilities that each bridge will fail are independent. The probability P_S that a path with edges of failure probabilities $p_1, p_2, \cdots, p_k$ will NOT fail is $P_S = (1-p_1)(1-p_2)\cdots(1-p_k)$. Therefore, the probability of failure P_F is $1 - P_S$. We want to minimize P_F for the nodes in the graph, so we can do something akin to a shortest paths algorithm, which attempts to minimize the failure probability. In fact, we can reduce the problem to a classical shortest path problem with nonnegative weights by observing that minimizing P_F is equivalent to minimizing $-\log(P_S) = \sum_{i=1}^{k} - \log(1 - p_i)$. Setting the weight of an edge to be $w_e = -\log p_e \geq 0$, we observe that a path of minimum failure probability is a path of minimum total weight (with respect to the weights w_e). We could thus simply apply Dijkstra with the weights w_e. Since we haven't really discussed whether our computational model allows the computation of logarithms we can simply modify Dijkstra to work directly on the p_e's instead of the w_e's but perform exactly the same updates as Dijkstra's algorithm would do on the w_e's. Correctness will simply follow from Dijkstra's correctness. Our variant of Dijkstra's algorithm will maintain estimates on the failure probabilities which we will relax. Here is the pseudocode.

```
NOT-FALL(G, w, s)
Initialize-NF-Single-Source(G, s)
S ← NIL
Q ← V[G]
while Q ≠ NIL
      do u ← EXTRACT-MIN(Q)
         S ← S ∪ {u}
         for each vertex v ∈ Adj[u]
             do NF-RELAX(u, v, w)
```

```
INITIALIZE-NF-SINGLE-SOURCE(G, s)
for each vertex v ∈ V[G]
      do d[v] ← 1
         pi[v] ← NIL
d[s] ← 0
```

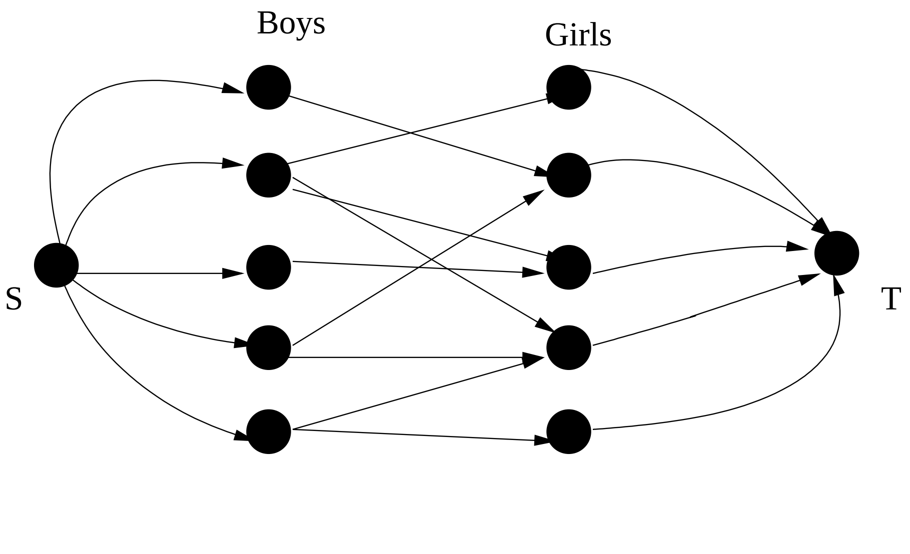
```
NF-RELAX(u, v, w)
if d[v] > 1 − (1 − d[u])(1 − w(u, v))
   then d[v] ← 1 − (1 − d[u])(1 − w(u, v))
        π[v] ← u
```

Running time and correctness follow directly from Dijkstra's algorithm as stated earlier by using the relationship between p_e and w_e.

Problem 6-4. Boy-Girl Matching Revisited

Last week we attempted to match students up for tango dancing based on their heights h_i. Having done so, we realized that matching students of similar ages a_i would also be useful. Design an efficient algorithm that maximizes the number of student pairs, such that in each pair, the partners' heights differ by no more than c and their ages differ by no more than d. (Hint: A good place to start on this problem would be to read section 26.3 in CLRS on maximum bipartite matching).

Solution:
We can formulate this problem as a max flow problem as follows. Each boy and each girl is a node in the graph. There is an edge between each boy and every girl he is eligible to dance with. Each of these edges has capacity one. Then draw a source node with edges of infinite capacity to each of the boys and a sink node with edges of infinite capacity from each of the girls. Finding the max flow of this graph will give you the maximal matching, since each boy will be paired with exactly one girl with whom he matches (unless he matches no one).


Boys
Girls
S
T