College of Computer & Information Science
Northeastern University
CSU690: Algorithms and Data

Spring 2005
Handout 17
15 April 2005

Sample Solution to Problem Set 6

1. (15 points) Bottleneck bandwidth

One can model a communication network by a weighted directed graph G, in which the vertices of G represent the nodes of the network, the edges of G represent the communication links of the network and the weight of an edge is the bandwidth of the link. We define the *bottleneck bandwidth* of a path p as the bandwidth of the minimum-bandwidth link in p.

Design and analyze an efficient algorithm to determine a path with the largest bottleneck bandwidth from a given node s to a given node t. If no path from s to t exists, then your algorithm must indicate so. Briefly justify the correctness of your algorithm. You may assume that the bandwidth of every link is positive. (*Hint:* Modify Dijkstra's algorithm.)

Answer: We can modify Dijkstra's algorithm by mainly replacing the "min" operation by "max", the "+" operation by "min", and the ">" operation by "<". These entail three kinds of changes in the algorithm.

First, consider the initialization routine INITIALIZE-SINGLE-SOURCE(G, s). We will set $d[v]$ to 0 for all $v \neq s$ and we set $d[s]$ to ∞.

Second, the RELAX(u, v, w) operation will change to the following code.

RELAX(u, v, w)
1. **if** $d[v] < \min\{d[u], w(u, v)\}$
2. $\quad d[v] = \min\{d[u], w(u, v)\}$
3. $\quad \pi[v] = u$

Finally, in DIJKSTRA(G, w, s), instead of extracting the vertex with the minimum d value, we extract the vertex with the maximum d value.

The proof of Dijkstra's algorithm for shortest paths proceeds by showing two claims: (a) $d[u] \geq \delta(s, u)$ for all u at all times, and (b) at the time u is extracted from Q, $d[u] = \delta(s, u)$. For establishing these two claims, the two main properties that we need are (i) if the last edge on the shortest path from s to v is (u, v), then $\delta(s, v) = \delta(s, u) + w(u, v)$, and (ii) for any edge (u, v), $\delta(s, v) \leq \delta(s, u) + w(u, v)$ (triangle inequality).

In our present situation, we are seeking the path with the largest bottleneck bandwidth – not the shortest path. Let $\Delta(s, u)$ denote the bottleneck bandwidth of the path from s to u with the largest bottleneck bandwidth. Claims (a) and (b) translate to the following two claims: (A) $d[u] \leq \Delta(s, u)$ for all u at all times, and (b) at the time u is extracted from Q, $d[u] = \Delta(s, u)$. We can establish these two claims by showing properties (I) and (II) analogous to the properties (i) and (ii) above: (I) if the last edge on the path with the largest bottleneck bandwidth from s to v is (u, v), then $\Delta(s, v) = \min\{\Delta(s, u), w(u, v)\}$, and (II) for any edge (u, v), $\Delta(s, v) \geq \min\{\delta(s, u), w(u, v)\}$. Note

that all we have done is to replace + by min and $\leq$ by $\geq$. Both claims (I) and (II) can be proved in a manner very similar to the proofs of (i) and (ii), that we went through in class.

One worthwhile remark is that the proof of claim (i) for the shortest-paths case can be done by establishing the optimal substructure property – that is, subpaths of shortest paths are shortest paths, something that is not always true for the bottleneck bandwidth case! Nevertheless, claim (I) can be proved by a simple argument based on contradiction.

2. (15 points) Arbitrage

Problem 24-3, page 615.

Answer: We view the given information as a weighted directed graph G in which the the currencies form the vertices and there is a directed edge between any two vertices. Thus, there are n vertices and $n(n-1)$ edges.

(a) Once we have specified the graph (without yet specifying the weights), there are two ways of solving this problem. One is by transforming the problem to one of finding the existence of a negative weight cycle in a graph – here we will judiciously choose the weights. And the other is to set the weight of the edge (i, j) to be the exchange rate $R(i, j)$ and then modify the Bellman-Ford algorithm to determine whether a cycle with product > 1 exists. In the latter approach, we also need to show that the modification of the Bellman-Ford algorithm works correctly.

Here we consider the first approach. For vertices (currencies) i and j, in G, we let the weight of edge (i, j) equal $-\lg(R(i, j))$. We now show that arbitrage exists if and only of there exists a negative weight cycle in G. This is easy to see since the weight of a cycle $i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_k \rightarrow i_1$ equals

$$\sum_{1 \leq j \leq k} -\lg(R(i_j, i_{j+1})) = -\lg\left(\prod_{1 \leq j \leq k} R(i_j, i_{j+1})\right),$$

thus implying that the weight of the cycle is negative if and only if $\prod_{1 \leq j \leq k} R(i_j, i_{j+1})$, which is the condition for arbitrage.

We have thus reduced our problem to that of finding whether there exists a negative-weight cycle in G. We simply invoke the Bellman-Ford algorithm starting from any source. Since every vertex is accessible from every other vertex, if a -ve weight cycle exists in G, we will find its existence using the algorithm. The running time of the algorithm is $O(VE) = O(n \cdot n^2) = O(n^3)$.

(b) In order to extract the cycle of currencies that yields arbitrage, we employ $n(n-1)+1$ iterations of the Bellman-Ford algorithm. In iteration 0, we run the Bellman-Ford algorithm on G. If no negative weight cycle is found, then there is nothing to do further. If the existence of a negative-weight cycle is indicated, then we perform the next $n(n-1)$ iterations (one associated with each of the $n(n-1)$ edges).

We number the edges arbitrarily from 1 to $n(n-1)$. We also select a source s arbitrarily. In iteration 1, we run the Bellman-Ford algorithm on the subgraph of G obtained after removing edge 1. If we find no negative-weight cycle, then this implies that 1 lies in all negative-weight

cycles or lies along a path from the source that leads to all negative-weight cycles. In this case, we keep edge 1 in all future iterations. Otherwise, we remove 1 from the graph.

So, in general, at the start of iteration i, we have a graph G_{i-1} which is a subgraph of G with a subset of the edges $\{1, 2, \ldots, i-1\}$ removed. In iteration i, we run the Bellman-Ford algorithm on the subgraph G' of G_{i-1} in which edge i has been removed. If a negative-weight cycle exists, then we set G_i to G'; otherwise, we set G_i to G_{i-1}.

Thus, after iteration $n(n-1)$, we have a subgraph G_n which consists of a set of edges, each of which belongs to every negative weight cycle in G_n or lies on a path from s to every negative-weight cycle. Thus, the graph G_n is simply a (possibly empty) path from source s to a negative-weight cycle. We can extract the negative-weight cycle from G_n by a search from s (depth-first or breadth-first).

Since there are $O(n^2)$ iterations, each involving a Bellman-Ford algorithm of $O(n^3)$ running time, the running time of the complete algorithm equals $O(n^5)$.

3. (10 points) Shortest paths on vertex-weighted graphs

Suppose you are given a connected undirected graph with weights on vertices (rather than on edges) and you are asked to compute the single-source shortest paths from a given source vertex. Here, the length of a path is defined as the sum of the weights on the vertices comprising the path. Give an algorithm for solving this problem by reducing it to the standard single-source shortest paths problem on directed graphs with weights on edges.

Answer: Given the undirected graph $G = (V, E)$, we construct a new directed graph $G' = (V, E')$, which has the same set V of vertices as G and has directed edges (u, v) and (v, u) for every undirected edge (u, v) in G. We set the weight of the edge from u to v in G' to be the weight of the vertex v. Now, the shortest path p from u to v in G is the same as the shortest path from u to v in G'; the weight of the path in G is the weight of the path in G' plus the weight of the vertex u. Thus, we can calculate the shortest paths from a given source to all other vertices in G by calculating the shortest paths from the same source to all other vertices in G'.

4. (10 points) Bellman-Ford algorithm

Exercise 25.1-5, page 628.

Answer: Let W denote the matrix with all the edge weights. Thus w_{ij} is the weight of the edge from i to j; if $i = j$, then $w_{ij} = 0$. Since we are interested in single-source shortest paths, we only need to compute a vector. We will compute this vector in a series of $n-1$ iterations, like in Bellman-Ford. Let $D^{(k)}$ the vector after i iterations. So the final output will be $D^{(n-1)}$. The ith entry in this vector will be the shortest path distance from the source to i.

We can set $D^{(0)}$ to be the vector with entries $(\infty\infty \ldots 0\infty \ldots \infty)$, where the 0 appears in the position of the source s. Assuming we write this vector as a column-vector, we can compute $D^{(i)}$ for $i \geq 1$ as

$$D^{(i)} = W \oplus D^{(i-1)},$$

where $\oplus$ indicates the matrix multiplication in which we have replaced $+$ and $\cdot$ by min and $+$,

respectively (as we discussed for the all-pairs shortest paths case). Thus, D^{n-1} equals $W^{n-1}D^{(0)}$, where the power of W is being taken with respect to the different notion of matrix multiplication.

5. (10 points) Floyd-Warshall algorithm

Exercise 25.2–1, page 634.

$$D^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix} \quad D^{(1)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 3 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix} \quad D^{(5)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 3 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$D^{(6)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$