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

Spring 2005
Handout 19
21 March 2005

Sample Solution to Final

Problem 1. (2 + 8 = 10 points) Bitonic sequences

A *bitonic* sequence is a sequence of numbers that monotonically decreases and then monotonically increases. That is, a sequence $X = x_1, x_2, \ldots, x_n$ is bitonic if there exists a t in $[1, n]$ such that $x_{i+1} < x_i$ for $1 \leq i < t$ and $x_{i+1} > x_i$ for $t \leq i < n$. For example, $11, 9, 6, 5, 1, 4, 7$ is bitonic since the sequence first decreases from 11 to 1 and then increases from 1 to 7. On the other hand, the sequence $11, 8, 7, 9, 10, 6$ is not bitonic since it decreases, then increases, and then again decreases.

(a) Give an $O(1)$ time algorithm to find the maximum of a given bitonic sequence.

Answer: The maximum of the first and last elements yield the maximum of the sequence.

(b) Give an $O(\lg n)$ time algorithm to find the minimum of a given bitonic sequence of length n. Briefly justify the correctness of your algorithm.

Answer: The minimum of a bitonic sequence is the unique element x_i in X that satisfies one of the following three conditions: (i) $i = 1$ and $x_i < x_{i+1}$; (ii) $i = n$ and $x_{i-1} > x_i$; (iii) $1 < i < n$, $x_{i-1} > x_i$, and $x_{i+1} > x_i$.

We use a recursive binary-search procedure. If the length of the list is 1, then simply return the lone element. Otherwise, if any of the conditions (i) or (ii) hold, we simply return the appropriate element. If none of these two conditions hold, we probe the middle element, say x_i of the list. If x_i satisfies condition (iii), we return this element. Otherwise, either $x_{i-1} < x_i < x_{i+1}$ or $x_{i-1} > x_i > x_{i+1}$. In the former case, we make a recursive call on the right half of the list. In the latter case, we make a recursive call on the left half of the item.

The recurrence for the running-time is $T(n) \leq T(\lfloor n/2 \rfloor) + \Theta(1)$, which yields an $O(\lg n)$ worst-case running time.

Problem 2. (10 points) Fairly splitting a heist

Two art thieves share a collection of n paintings, numbered 1 through n. They have agreed on a positive integer value for each painting, say $x_1, x_2, \ldots, x_n$. They would like to split the paintings into two halves of exactly equal value.

Design an algorithm that determines whether it is possible for the thieves to fairly split the paintings. Your algorithm should run in $O(nT)$ time, where T is the sum of the x_i's.

(*Hint:* Maintain an $n \times T$ matrix S, where $S[i, j]$ is 1 if one of the thieves can get a total value of i using a subset of the paintings 1 through j. Obtain a recurrence for computing S and use S to determine whether the paintings can be split equally.)

Answer: We can calculate $S[i, j]$ as follows. First we set $S[i, 0]$ to be 0 for all i. For any other i and j, we set $S[i, j]$ to 1 if either $i = x_j$, or $i > x_j$ and $S[i - x_j, j - 1] = 1$; otherwise, we set $S[i, j]$ to 0.

The desired problem can be solved as follows now. If T is odd, then the answer is trivial: it is not possible to fairly split the paintings. If T is even, then we would like each thief to have a total value of $T/2$. So we check whether $S[T/2, n]$ is 1. If it is, then the answer is yes; otherwise, the answer is no.

Problem 3 ($4 \times 3 = 12$ points) Graph properties

For each of the following statements, indicate whether it is true or false. In each case, briefly justify your answer.

(a) If vertex u appears before vertex v in a topologically sorted order of a directed acyclic graph G, then there is a directed path from u to v in G.

Answer: False. Consider the dag with three vertices x, u, and v and edges (x, u) and (x, v). One topologically sorted order is x, u, v. There is no edge (u, v), however.

(b) Every n-vertex undirected tree has exactly $n - 1$ edges.

Answer: True. The proof is by induction. For $n = 1$, the claim is trivially true. Assume the induction hypothesis that a tree T with n vertices has $n-1$ edges. For the induction step, consider an $n + 1$-vertex tree. Remove a leaf and the associated edge. We have an n-vertex tree with $n - 1$ edges. So the $n + 1$-vertex tree has n edges.

(c) Let G be an undirected connected graph with weights on edges. Assume that the edge weights are all distinct. Then, the edge with the smallest weight is always in the minimum spanning tree.

Answer: True. If not, then we can add the smallest-weight edge to the tree, remove the heaviest weight edge on the resulting cycle and obtain a spanning tree with lower weight.

(d) Let G be a weighted directed graph and s be a vertex in G. Let p denote a shortest path from s to t. If we increase the weight of every edge in the graph by 1, then p remains a shortest path from s to t.

Answer: False. Suppose there are two paths from s to t in G, one with 3 hops and total weight 3, and another with 1 hop and weight 4. When we increase all edges weights by 1, the 3-hop path gets weight 6, while the 1-hop path gets weight 5. Thus, the original shortest path is no longer the shortest path.

Problem 4. (8 points) Updating all-pairs shortest path lengths on addition of a new edge

Applications that maintain all-pairs shortest paths among various physical locations should be able to address changes in the network efficiently. For instance, when a new edge is added, one should be able to change the all-pairs shortest paths without having to compute these paths all over again.

Suppose we are given an n-vertex directed graph G with adjacency weight matrix W for which we have computed the all-pairs shortest path lengths. (Recall that the entry in the ith row and jth column of W is the weight of the edge (i, j), if it exists, and ∞, otherwise.) Assume that there are no negative weights in the graph. Suppose we are also given D, the all-pairs shortest path length matrix for G: for vertices i and j, the entry d_{ij} (ith row and jth column of the matrix D) gives the shortest path length between i and j.

Suppose a new edge (x, y) with weight c is added to G, where x and y are two vertices in G: thus, the entry w_{xy} in W changes from ∞ to c. Give an efficient algorithm to compute the new all-pairs shortest path length matrix, and analyze its worst-case running time. You need not compute the actual shortest paths.

(*Hint*: An $O(n^2)$ worst-case running time is achievable.)

Answer: We set the new shortest-path length d'_{ij} to $\min\{d_{ij}, d_{ix} + c + d_{yj}\}$, in parallel, for all i and j. Originally, d_{ij} specified the length of the shortest path. With the addition of the edge (x, y), we have two cases to consider. Either the shortest path contains the edge (x, y), or not. In the first case, the shortest such path will have total weight $d_{ix} + c + d_{yj}$. In the second case, the shortest such path will have weight d_{ij}.

Bonus Problem. (4 bonus points) Data compression

Suppose you are asked to compress large files containing characters from a 4-element alphabet $\{a, b, c, d\}$. You are told that each character of the file is independently drawn at random from the alphabet with probability distribution $(0.99, 1/300, 1/300, 1/300)$, respectively. Which of the following three compression schemes would you use and why: Huffman coding, Run Length coding, LZ77 compression with a dictionary of size 4 and lookahead 4. Credit will be given only if appropriate justification is provided.

Answer:

Huffman coding will assign 1 bit to a, 2 bits to b, and 3 bits to each of c and d. The average number of bits per character is close to 1 bit.

Run length encoding will replace a run of ℓ as by (a, ℓ). The average length of a run will be 100, and most of the runs will be concentrated around this number. So number of bits per character will be approximately 7 (the number of bits needed to code the most likely run lengths) plus a couple of additional bits, divided by 100.

An LZ77 code with dictionary 4 and lookahead buffer has to output at least 1 bit every 8 characters (or less).

Hence, run length coding will yield the most compression.