George Washington University
Department of Computer Science

CS212 - Solution to Midterm Exam

2 hours and 30 minutes (closed book)
10/23/2013

Total 100 points.

1. (5 pts. each) Let $n \geq 1$.

(a) Using the definitions of O (big-Oh) and Ω (Omega) notations, prove that:

$$\text{If } g(n) = \Omega((f(n)) \text{ and } g(n) = O(h(n)), \text{ then } f(n) = O(h(n)).$$

sol: Note that $g(n) = \Omega((f(n))$ implies that $g(n) \geq c_1 f(n)$ for any $n \geq n_1$ where n_1 and c_1 are constants, and $g(n) = O(h(n))$ implies that $g(n) \leq c_2 h(n)$ for any $n \geq n_2$ where n_2 and c_2 are constants. Hence, $c_1 f(n) \leq g(n) \leq c_2 h(n)$, i.e., $f(n) \leq \frac{c_2}{c_1} h(n)$, for any $n \geq n_0$ where $n_0 = \max\{n_1, n_2\}$. Therefore, $f(n) = O(h(n))$.

(b) Let $T(n) = cn + T(\frac{n}{2})$ for $n > 1$ and $T(1) = c$ where c is a constant. Show that $T(n) = O(n)$. Assume $n = 2^k$ for some integer k.

sol: $T(n) = cn + (\frac{cn}{2} + T(\frac{n}{2})) = \cdots = c(n + n/2 + n/2^2 + \cdots + n/2^{k-1}) + T(\frac{n}{2^k}) = cn\frac{1-(\frac{1}{2})^k}{1-\frac{1}{2}} + c = O(n)$.

(c) Let $T(n) = cn + 2T(\frac{n}{2})$ for $n > 1$ and $T(1) = c$ where c is a constant. Show that $T(n) = O(n \log n)$. Assume $n = 2^k$ for some integer k.

sol: $T(n) = cn + 2T(\frac{n}{2}) = 2cn + 2^2 T(\frac{n}{2^2}) = \cdots = kcn + 2^k T(\frac{n}{2^k}) = kcn + 2^k c = kcn + nc = O(n \log n)$.

2. (5 pts) Construct a set of Hoffman codes for an alphabet with 7 characters $a_1, \cdots, a_7$ with relative frequencies $(q_1, \cdots, q_7) = (3, 3, 7, 7, 8, 15, 19)$.

3. (5 pts. each) Suppose you are given an array of n element in increasing order.

(a) Describe an $O(n)$ time algorithm to find two numbers from the list that add up to zero, or report there is no such pair.

sol: Let $a_1 \leq a_2 \leq \cdots \leq a_n$. We use two index variables i and j. Initially, $i = 1$ and $j = n$. Repeat: if $a_i + a_j = 0$, we are done. Otherwise, increase i by 1 if $a_i + a_j < 0$, and decrease j by 1 if $a_i + a_j > 0$.

(b) Describe an $O(n^2)$ time algorithm to find three numbers from the list that add up to zero, or report that there is no such triple.

sol: For each $k = 1$ to n do the following:
Let $A' = A - \{a_k\}$ (note that A' is a sorted list); Using two index variables, search for a pair of numbers whose sum is $-a_k$. This can be done similarly as in your algorithm in (a) in which if $a_i + a_j = -a_k$, we are done. Otherwise, increase i by 1 if $a_i + a_j < -a_k$, and decrease j by 1 if $a_i + a_j > -a_k$, and repeat.

4. (10 pts) Consider the fractional Knapsack problem discussed in class defined as: given a knapsack with capacity M and n items with weight function $w(i)$ and profit function $p(i)$ for each i, $1 \le i \le n$, the objective is to find a solution $X = (x_1, \cdots, x_n)$ such that (i) $0 \le x_i \le 1$, (ii) $\sum_{i=1}^{n} x_i w(i) \le M$, and (iii) $\sum_{i=1}^{n} x_i p(i)$ is maximized.

Now, we consider a slight modification of the problem such that we are given an upper $u(i)$ for each $x(i)$, $1 \le i \le n$, i.e., $0 \le x(i) \le u(i)$ for each i. Note that the original Knapsack problem is when $u(i) = 1$ for each i.

Design a greedy algorithm to solve this problem, and justify the correctness of your algorithm. (In your argument, you may assume the correctness of the greedy algorithm solving the original version of the Knapsack problem.)

sol: Let $p_1/w_1 \ge p_2/w_2 \ge \cdots \ge p_n/w_n$. Consider items starting from $1, 2, \cdots$; put item i as much as you can, i.e., let x_i be the largest number that is no larger than $u(i)$ without violating the capacity constraint.

To prove the correctness of this algorithm, you can apply almost same arguments as in the proof of the greedy algorithm with a few exceptions. Let j be the smallest index such that $x_j < u(j)$. (Note that j was defined to be the smallest index such that $x_j < 1$ in the proof of the original problem.) We then define k to be the smallest index such that $x_k \ne y_k$ where X is a greedy solution and Y is an optimal solution, and can show $y_k < x_k$. When constructing a new solution Z from Y, z_k is defined to be x_k and proceed the remaining arguments as in the proof of the original problem.

5. (5 pts each) (Show all your work.)

(a) Apply the Partition algorithm to $A = (3, 3, 8, 5, 3, 8, 10, 3, 2, 3, 3, 1, 4, 3)$ using the first element as the pivot. In your output, elements with the same value of the pivot element **must** be placed in the left of the pivot element upon completion of the Partition algorithm.

(b) Apply the linear-time Select algorithm to the following example:
$L = \{1, 3, 13, 14, 2, 6, 18, 5, 15, 16, 17, 18, 12, 7, 8, 18, 12, 18, 19, 20, 21, 22\}$ and $k = 18$.

6. Consider a sequence $x_1, x_2, \cdots, x_n$ of numbers such that for some unknown i ($1 \leq i \leq n$, $x_1 < x_2 < \cdots < x_i$ and $x_i > x_{i+1} > \cdots > x_n$.

(a) (5 pts) Describe an $O(\log n)$ time algorithm to find the kth smallest element for an arbitrary integer k.

(b) (5 pts) Apply your algorithm in (a) to the following example: $X = (3, 5, 6, 8, 10, 9, 7, 4, 2, 1)$ and $k = 6$. (Show all your works.)

7. (5 pts.) Given n segments of line (into the X axis) with coordinates $[l_i, r_i]$, i.e., left and right end points. You are to choose the minimum number of line segments that cover the segment $[0, M]$. Design a greedy algorithm to solve this problem and argue why your algorithm is optimal.

   sol: Let $c = 0$; Among the lines that cover point c, choose one, say l_i, with the largest right end point; let $c = r_i$. Remove all lines whose right end point is less than or equal to c, and repeat the above process.

   To prove the optimality, we claim that there always exists an optimal solution that includes a line that covers point 0 and has the largest right end point. Suppose there is an optimal solution that does not include such a line, say l_k. Since point 0 must be also covered, there must be a line, say l_j, in the optimal solution that covers point 0 but $r_j < r_k$. It is clear that the optimal solution when l_j is replaced by l_k also covers the segment $[0, M]$ with the total number of lines in the new solution remained same as in the optimal solution. Therefore the claim is true; hence, our algorithm must generate an optimal solution.

8. (5 pts) A *forest* is defined to be a graph without a cycle, i.e., a graph that may have more than one component, but each component does not include a cycle. Let $G(V, E)$ be a connected edge-weighted graph. Let k be an arbitrary integer in $1 \leq k \leq |V|$.

(a) Design an algorithm for finding a forest with exactly k components such that the sum of the weights of edges in the forest is minimum.

   sol: Apply Kruskal's MST algorithm until you have exactly k components. Recall that Kruskal's algorithm stops when a tree with n nodes (i.e., a single component) is constructed.

(b) Let $V_0 = \{v_1, v_2, \cdots, v_k\}$ be a subset of k vertices in V. Design an algorithm to find a forest with exactly k components such that each component contains exactly one of the vertices in V_0.

   sol: We construct a new graph $G'$from G by adding a new node, say v_0 and adding k edges $v_0, v_i)$ for each i, $1 \leq i \leq k$ with $w(v_0, v_i) = \epsilon$. Then, find a MST T of G'. (Note that any MST must include all of new edges.) The, remove k new edges from T. The resulting graph is a desired forest.

9. (10 pts) Assume that edge weights are distinct in a given graph. Prove that there exists an unique minimum spanning tree of G.

10. (5 pts each) Consider the following graph G. Show all your work for each of the following problems.

(a) Find a minimum spanning tree of G obtained using the Prim's algorithm starting from node 3.

(b) Find a minimum spanning tree of G obtained using the Kruskal's algorithm.

(c) Apply the Dikstra's shortest path algorithm and find a shortest path tree of G where node 3 is the start node.

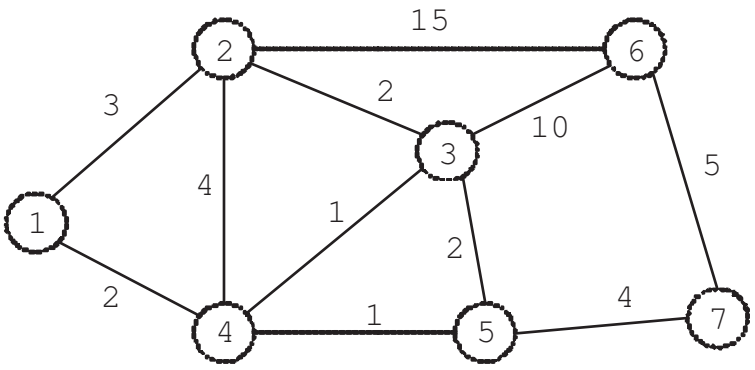

Figure 1: G