Homework 03 — Solution

CS 624, 2022 Fall

Review the course homework policies before you start!

1. Problem 9.3-8 on p223.

   Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\lg n)$-time algorithm to find the median of all $2n$ elements in arrays X and Y.

   For simplicity, I'll ignore the issues of rounding $\frac{n}{2}$ initially.

   Compare the medians of X and Y: x_m and y_m. The following cases are possible:

   - If $x_m = y_m$, then both are $\geq X[1..\frac{n}{2}]$ and $Y[1..\frac{n}{2}]$ and both are $\leq X[\frac{n}{2}..n]$ and $Y[\frac{n}{2}..n]$, so it is also the median of the combination of X and Y.
   - If $x_m < y_m$, then every element in $X[1..\frac{n}{2}]$ is in the lower half ($\leq$ the median), because it is $\leq$ both $X[\frac{n}{2}..n]$ and $y_m < Y[\frac{n}{2}..n]$. Similarly, every element in $Y[\frac{n}{2}..n]$ is in the upper half ($\geq$ the median), because it is $\geq$ both $Y[1..\frac{n}{2}]$ and $x_m > X[1..\frac{n}{2}]$. So we discard the lower half of X and the upper half of Y, and we recur on $X[\frac{n}{2}..n]$ and $Y[1..\frac{n}{2}]$. To work out the rounding, we just need to make sure that we always discard the same number of below-median and above median elements, and we must avoid discarding the medians themselves.
   - If $x_m > y_m$, then the situation is like the previous case, just with X and Y swapped.

   The size of the problem is roughly halved each time, so the total time is $O(\lg n)$ time.

2. Problem 12.2-5 on p293.

   Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

3. Problem 15.4-5 on p397.

   Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

   The easy way: Let the original array be called A. Make a copy, sort it, and delete any duplicates; call the sorted, de-duplicated array B. Then the longest common subsequence of A and B is the longest monotonic increasing subsequence of A.

   Sorting the copy takes $O(n \log n)$ time, and finding the LCS takes $O(n^2)$ time, so the total cost is $O(n^2)$.

   The hard way:

   Optimal substructure: Suppose that X is a sequence of n numbers ($n > 0$) and Z is a LMIS of X with length m. There are two possibilities:

   - $X[n] = Z[m]$, and $Z[1..m-1]$ is a longest (monotonically increasing subsequence of $X[1..n-1]$ that ends in a value strictly less than $X[n]$); or

- $X[n] \neq Z[m]$, and Z is a LMIS of $X[1..n-1]$.

Note that the first case does *not* refer back to the unrestricted LMIS problem; it refers to a constrained version of it. That suggests that we need to generalize the problem to take an upper bound on the end of the subsequence. The relevant upper bounds will either be elements of X or ∞, so let's "extend" X with $X[n+1] = \infty$ so all upper bounds can be represented via array indexes.

Define $\text{llmis}(i, j)$ to be the length of the longest monotonic increasing subsequence within $X[1..i]$ that ends in a value strictly less than $X[j]$. Its cases reflect the possibilities listed above:

$$\text{llmis}(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ \max(\text{llmis}(i-1,j), \underline{1 + \text{llmis}(i-1,i)}) & \text{if } X[i] < X[j] \\ \text{llmis}(i-1,j) & \text{otherwise} \end{cases}$$

The underlined term represents the situation where $X[i]$ is part of the subsequence.

Then $\text{llmis}(n, n+1)$ is the length of the longest monotonic increasing subsequence in $X[1..n]$ (the original input), since it must end in a value less than ∞. We can use an array of size $n \times (n+1)$ as a memo table for llmis and calculate $\text{llmis}(n, n+1)$ in either bottom-up or top-down fashion. The array has $O(n^2)$ elements; each llmis calculation taks $O(1)$ time once its dependencies are filled in; so the whole calculation takes $O(n^2)$ time.

After calculating the length, an actual sequence can be constructed from the memo table as follows: Start with $i = n$ and $j = n+1$. While $i > 0$, compare the memo table's stored value for $\text{llmis}(i, j)$ with the underlined term above; if they match, then recur on $(i-1, i)$ and then emit $X[i]$; otherwise recur with $(i-1, j)$. This takes $O(n)$ additional time, because i decreases from n by one each step, and each step takes $O(1)$ time.

Alternatives: There is also a solution based on a different generalization: $\text{llmis}'(i)$ is the length of the longest monotonic increasing subsequence of $A[1..i]$ ending with $A[i]$. With the extension $A[n+1] = \infty$, $\text{llmis}'(n+1)$ is one plus the length of the LMIS for the original input. The memo table is one dimensional with $O(n)$ elements, but the cost to fill each slot is no longer constant but linear, so the total cost is still $O(n^2)$. Constructing the actual sequence is also more complicated, but it can be simplified using an auxiliary array.

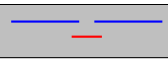
4. Problem 16.2-2 on p427.

Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack.

5. Problem 16.1-3 on p422.

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities.

(a) Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work.



(b) Do the same for the approach of always selecting the compatible activity that overlaps the fewest other remaining activities.

(c) Do the same for the approach of always selecting the compatible remaining activity with the earliest start time.

6. Problem 16.2-5 on p428.

Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Let $X = \{x_1, x_2, \ldots, x_n\}$. A solution can be described as a set of m starting points for unit intervals—that is, as $\{a_1, \ldots, a_m\}$ such that $X \subseteq \bigcup_{i=1}^{m} [a_i, a_i + 1]$.

Optimal substructure: Suppose that A is an optimal solution for X, and suppose that $a_k \in A$. Then $A = A_L \cup \{a_k\} \cup A_R$, where A_L is an optimal solution for $\{x \mid x \in X, x < a_k\}$ and A_R is an optimal solution for $\{x \mid x \in X, x > a_k + 1\}$.

Greedy choice property: If $x_1 = \min(X)$, then there is an optimal solution for X that contains x_1 as the starting point of an interval. *Proof:* Suppose that A is an optimal solution and $x \notin A$; then x must be covered by some interval starting at $a \in A$, and in fact $a < x$. Then the part of a's interval before x is useless (since x is the minimum element of X), and we could move it over to start at x. That interval will cover at least as much of X as before (possibly more), so the solution is at least as good as the A, which was assumed to be optimal, so the adjusted solution is also optimal.

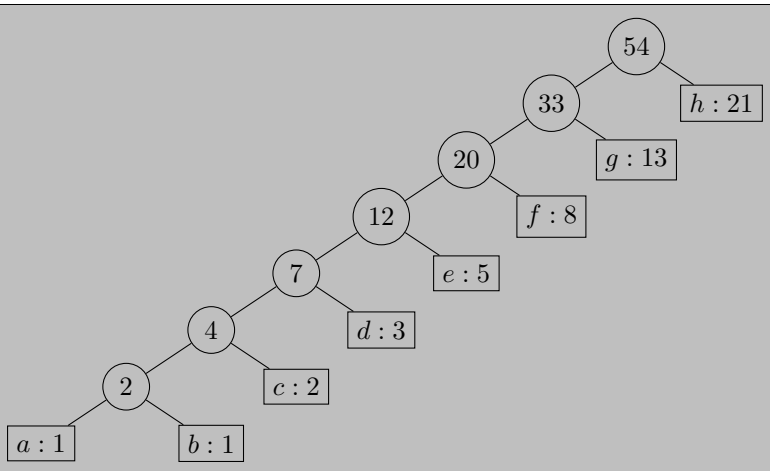
Algorithm: Always take the least element ($x_{\min}$) as the starting point of an interval and recur on $X - [x_{\min}, x_{\min} + 1]$. We can do this efficiently as follows: Start by sorting the input array X. Start at $i = 1$. While $i < n$, take $a = X[i]$ as an interval starting point, and repeatedly increment i until either we hit the end of the array or $X[i] > a + 1$. The total cost is $O(n \log n + n) = O(n \log n)$, or just $O(n)$ if we are given the input array already sorted.

7. Problem 16.3-3 on p436.

What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a : 1 \quad b : 1 \quad c : 2 \quad d : 3 \quad e : 5 \quad f : 8 \quad g : 13 \quad h : 21$$

Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?



(for some labeling of edges with 0 and 1)

Generalization: When we run the pairing loop, after the first iteration, the smallest two nodes will always be the paired node just created and the leaf with the next Fibonacci number. In fact:

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1$$

and since $f_{n+1} \leq \sum_{k=1}^{n} f_k < f_{n+2}$, we will always pair the accumulated node and the next leaf node (f_{n+1}) together.

Theorem: $f_{n+2} = 1 + \sum_{k=1}^{n} f_k$
Proof: By induction on n.

Base case ($n = 0$):
Goal: $f_2 = 1 + \sum_{k=1}^{0} f_k = 1 + 0 = 1$.
Trivial, by definition of f_2. Done.

Inductive case:
IH: $f_{n+2} = 1 + \sum_{k=1}^{n} f_k$.
Goal: $f_{(n+1)+2} = 1 + \sum_{k=1}^{n+1} f_k$.

$$\begin{aligned} f_{(n+1)+2} &= f_{n+2} + f_{n+1} && \text{by Fibonacci recurrence} \\ &= 1 + \sum_{k=1}^{n} f_k + f_{n+1} && \text{by IH} \\ &= 1 + \sum_{k=1}^{n+1} f_k && \text{absorb term into sum} \end{aligned}$$

Done.