Spring 2025 Qualifying Exam—Algorithms (750)

Question 1 (A Recurrence). Find tight asymptotic bounds on any positive function $T(n)$ satisfying the following recurrence for all sufficiently large n:

$$T(n) = 2T(n/2) + T(\sqrt{n}) + \sqrt{n}\ .$$

You may assume that any implicit floors and ceilings are of no consequence.

Prove your upper bound by the substitution method. (You do not need to prove the matching lower bound, but if your upper bound is not tight, you will not receive credit even for a correct substitution proof.)

Answer: $T(n) = \Theta(n)$. This is suggested by case 1 of the Master Theorem, assuming the $T(\sqrt{n})$ term is negligible.

Proof of the upper bound: assume $T(m) \le cm - d\sqrt{m}$ for all $m < n$, where $c > 0$ and $d \ge 0$ are yet to be determined. Then

$$\begin{aligned} T(n) &= 2T(n/2) + T(\sqrt{n}) + \sqrt{n} \\ &\le 2\left(cn/2 - d\sqrt{n/2}\right) + c\sqrt{n} - d\sqrt[4]{n} + \sqrt{n} \\ &\le cn - \sqrt{2}d\sqrt{n} + c\sqrt{n} + \sqrt{n} \\ &= cn - (\sqrt{2}d - c - 1)\sqrt{n} \\ &\le cn - d\sqrt{n} \end{aligned}$$

provided

$$\begin{aligned} d &\le \sqrt{2}d - c - 1 \\ c + 1 &\le (\sqrt{2} - 1)d \\ d &\ge \frac{c+1}{\sqrt{2}-1}\ . \end{aligned}$$

(The value of c is determined solely by the base cases, which can be ignored.)

Question 2 (Optimal Change-Making). You are the cashier at a bodega, and your cash register has an unlimited supply of coins of various denominations $d_1, \ldots, d_k$ (in cents, say, but the currency unit is not important). A customer comes in and pays cash for a purchase, handing you a large bill. After subtracting the price of the purchased item, you need to give exact change back to the customer, say N cents, and you want to do this using as few coins as possible.

For example, let $k = 3$ and $\langle d_1, d_2, d_3 \rangle = \langle 5, 10, 25 \rangle$. If $N = 45$, then three coins suffice ($45 = 25 + 10 + 10$); if $N = 17$, then you cannot make exact change.

Describe an algorithm that takes as input an integer $N \ge 0$ (the amount of change required) and an array $d[1 \ldots k]$ of positive integers (the coin denominations; assumed pairwise distinct), and returns the least possible number of coins whose denominations add up to N. If making exact change for N is not possible, your algorithm should return some (any) number greater than N (including ∞).

Describe your algorithm in enough detail so that someone who did well in CSCE 750 can implement it without specific knowledge of the problem.

(Formally, your algorithm returns the least possible value of $\sum_{j=1}^{k} c_j$ subject to the constraints that $c_1, \ldots, c_k \geq 0$ are integers and $\sum_{j=1}^{k} c_j d[j] = N$. Here, each c_j is the number of coins of denomination $d[j]$ used for the change. If no such $c_1, \ldots, c_k$ exist, then some number greater than N is returned.)

Your algorithm only needs to return the total number of coins used, not the number of each denomination used.

For full credit, your algorithm must run in time $O(kN)$.

Answer: Maintain a table $c[0 \ldots N]$ such that $c[i]$ holds the minimum number of coins needed to make change for i cents. Then $c[N]$ is returned. Below, we use the convention that $\min \emptyset := N$ or any number $\geq N$.

1. Allocate a table $c[0 \ldots N]$ of integers.
2. Set $c[0] = 0$. // No coins for no change
3. For i going from 1 to N:
   (a) $c[i] = 1 + \min\{c[i - d[j]] : 1 \leq j \leq k \ \ \& \ \ i - d[j] \geq 0\}$. // The "last coin" has value $d[j]$.
4. Return $c[N]$.

Remarks: There's no need to expand step 3(a) in more detail. One can use ∞ in place of an integer larger than N.

Question 3 (Single-Source Shortest Paths). You are given a directed graph G with weight function $w : G.E \to \mathbb{R}$ such that $w(e)$ is an *integer* between 1 and 10 for all $e \in G.E$. You are also given a source vertex $s \in G.V$.

Show how to compute, in *linear time*, shortest distances (d-attributes) and shortest path information (π-attributes) to every vertex from s. Linear time means time $O(|G.V| + |G.E|)$ with the usual adjacency-list representation of G. Your explanation should be clear enough so that someone who did well in CSCE 750 can implement it.

Answer: There are at least two solutions I can think of:

1. For every edge e such that $w(e) = i \in \{1, 2, \ldots, 10\}$, chop up e into a directed path (in the same direction as e) consisting of i many edges, each with weight 1. This is done by adding $i - 1$ many intermediate, dummy vertices. This dilates the size of the input by a factor of at most 10 (a constant). Then
   (a) Run BFS with source s on the modified graph.
   (b) Adjust π-attributes of non-dummy vertices to point to the closest non-dummy predecessor.
   (c) Remove all the dummy vertices.

2. Run Dijkstra's algorithm on the original graph G with source vertex s, but modify the implementation of the priority queue to be an array of 11 doubly linked lists of vertices, each list consisting of vertices with the same current d-value. (The d-values are all integers, and the min and max d-values in the queue at any time differ by at most 10 (except for ∞-values, which can be kept on a separate list). This allows for constant-time INSERT and EXTRACT-MIN operations. The double linking in the lists allows for a constant-time DECREASEKEY operation.)

   This next point is more detail than necessary: A pointer must be associated with each vertex in the representation of G to point to its corresponding entry in the priority queue, so that accessing the queue element corresponding to a vertex can be done in constant time given the vertex.