CSC 545 Homework 3 (version 2) Spring 2021

This homework is due Wednesday, April 14, at 3:30pm MST. Please upload a single PDF file containing your submission (ensuring scans of handwritten work are legible) on Gradescope by that time.

The questions are drawn from the material in the lectures, and in Chapters 16 and 17 of the text, on greedy algorithms and amortized algorithms.

The homework is worth a total of 100 points. When point breakdowns are not given for the parts of a problem, each part has equal weight.

In grading the required problems on the homework, only a subset of two required problems will be graded, whose points together add up to a total of 100 points.

For questions that ask you to design a greedy algorithm, prove that your algorithm is correct making use of a lemma of the following form:

**Lemma** If a partial solution $P$ is contained in an optimal solution, then the greedy augmentation of $P$ is still contained in an optimal solution.

Prove the lemma using an exchange-style argument, where you transform the optimal solution to contain the augmentation and argue that this transformation does not worsen the solution value.

Please remember to start each problem on a new page, and mark the corresponding pages in your submission for each homework problem on Gradescope. Conciseness counts!

(1) **(Room scheduling)** (50 points) Suppose you have $n$ classes that you want to schedule in rooms. Each class is offered at a fixed time interval, and classes whose times overlap cannot be scheduled in the same room. There are enough rooms to schedule all the classes.

Design a greedy algorithm to find an assignment of classes to rooms that minimizes the total number of rooms used, in $O(n \log n)$ time. Prove that your algorithm finds an optimal assignment using a lemma of the required form.

(2) **(Least change)** (50 points) Suppose that for a given amount of cents $n$, you want to choose coins whose total value adds up to $n$, where the coins have denominations that come from a fixed assortment of values. The problem is to choose coins that add up to $n$, while minimizing the number of coins used. In other words, you want to make $n$ cents in change, using the fewest possible number of coins. You may assume that there are as many coins as you wish of each denomination.

(a) (30 points) Design a greedy algorithm that makes change for the US system of coins, in other words, where the coin denominations are quarters, dimes, nickels, and pennies. Prove that your algorithm finds an optimal solution with the fewest number of coins, using a lemma of the required form.

(b) (10 points) Prove that your greedy algorithm from Part (a) does not make optimal change for all systems of coins. More precisely, demonstrate a system of coin denominations that does contain the penny, and an explicit value of $n$, such that your greedy algorithm from Part (a) does not use the fewest number of coins.

(c) (10 points) Suppose that the system of coins has denominations $\{a^0, a^1, \ldots, a^k\}$ for integers $a > 1$ and $k \geq 1$. Prove that your greedy algorithm makes optimal change for this system.
(d) **(bonus)** (10 points) Using dynamic programming, design an algorithm that makes optimal change for any system of \( k \) coins that contains the penny, and which runs in \( O(kn) \) time.

(Note: Since by Part (b), a greedy change-making algorithm will in general not be optimal for arbitrary coin denominations, your proof of correctness for Part (a) will have to make use of special properties of the US coin denominations.)

(3) **(Constant amortized time extract)** (50 points) Show that, by an appropriate choice of a potential function, the standard implementation of the implicit heap used in heap sort takes \( O(1) \) amortized time for an Extract, and \( O(\log n) \) amortized time for an Insert.

(Note: In your solution, (a) specify how you concretely measure the real time for these two operations, (b) specify your potential function for the heap, and (c) analyze the amortized time for both operations. Implicit heaps are described in Section 6.5 of the text.)

(4) **(Amortized search trees)** (50 points) For binary search tree \( T \) and node \( x \) in \( T \), let

- \( s(x) \) be the size of the subtree rooted at \( x \),
- \( \ell(x) \) be the left child of \( x \), and
- \( r(x) \) be the right child of \( x \).

Given a constant \( \alpha \), where \( \frac{1}{2} \leq \alpha < 1 \), a binary tree \( T \) is said to be \( \alpha \)-balanced if at every node \( x \) of \( T \), both of the following hold:

\[
\begin{align*}
\quad & s(\ell(x)) \leq \alpha s(x), \\
\quad & s(r(x)) \leq \alpha s(x).
\end{align*}
\]

(a) (10 points) Show that an arbitrary \( n \)-node tree can be made \( \frac{1}{2} \)-balanced in \( \Theta(n) \) time using \( \Theta(n) \) space.

(b) (10 points) Show that performing a Find operation in an \( n \)-node \( \alpha \)-balanced binary search tree takes \( O(\log n) \) worst-case time.

(c) (20 points) Consider the following amortized approach for supporting the Insert and Delete operations on a search tree. Suppose Insert and Delete are implemented in the standard way in an ordinary search tree that is not balanced, except that now after an Insert or Delete, the tree is rebalanced in the following way. After the Insert or Delete, find the highest node \( x \) in the tree that is not \( \alpha \)-balanced, and rebuild the subtree rooted at \( x \) so it becomes \( \frac{1}{2} \)-balanced, using your solution to Part (a). We call this task of rebuilding the subtree at \( x \) in this way, rebalancing the tree.

Prove that rebalancing an \( n \)-node \( \alpha \)-balanced tree, where \( \alpha > \frac{1}{2} \), takes \( O(1) \) amortized time.

To prove this, use the potential method with the following potential function \( \Phi(T) \). For a node \( x \) in \( T \), let

\[
d(x) := |s(\ell(x)) - s(r(x))|.
\]

Then

\[
\Phi(T) := \frac{1}{2\alpha - 1} \sum_{x \in T, d(x) \geq 2} d(x).
\]
(d) (10 points) Using your answer to Part (c), show that an Insert or a Delete on an $n$-node $\alpha$-balanced tree, where $\alpha > \frac{1}{2}$, takes $O(\log n)$ amortized time.

Note that Problem (2)(d) is a bonus question. It is not required, and its points are not added to regular points.