CS41 Homework 7

This homework is due at 11:59pm, November 8. Write your solution using LATEX. Submit this homework using github. For this homework, you will work with a partner. It’s ok to discuss approaches with others at a high level, but most of your discussions should just be with your lab partner. The only exception to this rule is work you’ve done with a lab partner while in lab. In this case, note who you’ve worked with and what parts were solved during lab. If there are questions about academic integrity, please visit the section on Academic Integrity on the course website.

0. Before final submission, make sure to fill out the README file.

1. Canonical Coins.

On homework 5, you provided a greedy algorithm for making change, and proved that for one set of coins (Euro denominations) the greedy algorithm is optimal, while for another set of coins (the counterexample you came up with), the greedy algorithm is not optimal. In this problem, you will automate this proof process. Your task is to design an algorithm that takes a set of coins as input, and determines whether those coins are canonical. A set of coins is canonical if making change with the greedy algorithm always succeeds (it is possible to make any amount of change), and is always optimal (it never uses more coins than necessary).

(a) Prove that for any set of coins $C = \{c_1, c_2, \ldots, c_k\}$, there exists an upper bound $n$ on the values that need to be checked in order to prove that $C$ is canonical. That is, there is some finite $n$, depending on the set of coins, where if $\text{greedy}(C, i) = \text{optimal}(C, i) \neq \text{FAIL}$ for all $i \leq n$, then $C$ is canonical.

(b) Describe an algorithm that takes a set of coins and determines whether they are canonical. What is the running time of your algorithm? Explain why your algorithm is correct (formal proof not required).

(c) Implement your algorithm from part (b) using the starter code in coins.py. You should submit this program along with your LATEX writeup. Your program should read from standard input, and write to standard output (this is already handled by the starter code).

Input: The input contains $k$ coin denominations $c_1, c_2, \ldots, c_k$ as space-separated integers, in increasing order: $0 < c_1 < c_2 <\cdots c_k < 10^6$.

Output: If the set of coins is canonical, print the string CANONICAL, and the range of numbers you had to check to prove it. If the set of coins is not canonical, print the string NON-CANONICAL, and the smallest number on which the greedy algorithm fails.

Sample Inputs and Outputs:
2. **Optimization vs Decision Problems.** Recall that a decision problem requires a **yes/no** answer, and an optimization problem requires the “best possible answer”, which often means maximizing or minimizing over some *cost* or *score*. For most optimization problems, there is an obvious analogue as a decision problem. For example, consider the following problem:

**VC-Opt:** Given a graph $G = (V, E)$, return the size of the smallest vertex cover in $G$.

VC-Opt has a natural decision problem, namely **Vertex-Cover**. In fact, every optimization problem can be converted to a decision problem in this way.

(a) Show that **Vertex-Cover** $\leq_P$ **VC-Opt**.

(b) Let $B$ be an arbitrary optimization problem, and let $A$ be the decision version of $B$. Show that

$$A \leq_P B.$$  

(c) Show that **VC-Opt** $\leq_P$ **Vertex-Cover**.

3. **Polynomial-time Verifiers.** Recall the definition of a *polynomial-time verifier*: Call $V$ an efficient *verifier* for a decision problem $L$ if

(a) $V$ is a polynomial-time algorithm that takes two inputs $x$ and $w$.

(b) There is a polynomial function $p$ such that for all strings $x, x \in L$ if and only if there exists $w$ such that $|w| \leq p(|x|)$ and $V(x, w) = \text{YES}$.

Give a polynomial-time verifier for **Factoring** (and prove its correctness).

4. **Memoization vs Tabulation.** *(Extra Credit)* You’ve seen two different methods of implementing a dynamic program—memoization and tabulation. In practice, which performs better? Is one always a better performer?

Pick a dynamic programming problem we’ve seen already (or select a new one!) and implement the DP both using memoization and tabulation. Next, test your programs on varying inputs of varying sizes. Is one implementation method always better? Is memoization always better on certain kinds of inputs (which ones?) Does the relative performance of memoization
vs tabulation depend only on the input size, or are there examples where one technique is better on some inputs, but worse on others? How much better/worse can memoization be vs tabulation?

Include a description of what you implemented, along with any observations/conclusions, in your hw7.tex file.

5. **Optimization vs Decision Problems. (Extra Credit)** Let $B$ be an arbitrary optimization problem, and let $A$ be the decision version of $B$. Does it *always* hold that

$$B \leq_P A?$$

Answer YES or NO. Justify your response.