ASSIGNMENT 3

DUE: Wednesday October 7, 5 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read http://www.student.cs.uwaterloo.ca/~cs341 for general instructions and policies.

Exercises. The following exercises are for practice only. You may ask about them in office hours. Do not hand them in.

1. You want to create a given number \( n \) by starting with the number \( t = 1 \) and then using steps where you either double \( t \) or add 1 to \( t \). For example, you can get 7 by starting with 1 and applying the sequence of steps \( +1, \times 2, +1, +1, +1 \) or the sequence \( \times 2, +1, \times 2, +1 \). Give a greedy algorithm to find the shortest possible sequence to produce the number \( n \).

2. Does greed bring happiness? The happiest interval scheduling problem is defined as follows: Given a set of \( n \) intervals defined by start and finish times \( (s_1, f_1), \ldots, (s_n, f_n) \) and \( n \) positive integers \( h_1, \ldots, h_n \) denoting the happiness value of each interval, find a subset \( I \subseteq \{1, 2, \ldots, n\} \) of disjoint intervals that maximizes \( \sum_{i \in I} h_i \).

Determine which of the following greedy algorithms correctly solves the happiest interval scheduling problem. Justify each answer with a proof of correctness or a counter-example.

(a) Earliest finish time. Choose the interval \( i \) with the earliest finish time, discard all intervals that overlap with \( i \), and recurse.

(b) Highest happiness per minute. Choose the interval \( i \) with the maximum happiness per minute ratio \( \frac{h_i}{f_i-s_i} \), discard all intervals that overlap with \( i \), and recurse.

(c) Kondo’s algorithm. If no intervals overlap, select them all. Otherwise, discard the interval \( i \) with the smallest happiness value \( h_i \). Repeat.

Problems. To be handed in.

1. [10 marks] Greedy Algorithms. Suppose you have an array \( A[1..n] \) of positive integers. We say that an index \( i \) covers the subsequence from index \( i \) to index \( i + A[i] - 1 \) (or to the end of the array in case the index is larger than \( n \)). For example, in the array \( A = [4, 3, 1, 5, 4, 2, 2] \) index 1 covers \( A[1..4] \), index 3 covers \( A[3] \), and index 4 covers \( A[4..7] \) (truncated at the end of the list).

The problem is to cover the whole array by choosing as few indices as possible. For example, a solution for the above array is the set of indices \( \{1, 4\} \).

(a) [2 marks] Clearly we must include index 1 in any solution. One greedy idea is to next choose the index of a maximum value in the array. Find an example where this greedy choice is wrong (i.e., does not lead to an optimum solution no matter which further choices are made).
(b) [8 marks] Give a greedy $O(n)$ time algorithm to solve the problem. In order to help you and the markers, we have broken this down into several parts.

i. [1 mark] Describe the idea of your algorithm in English. Use the following notation: $G[1..t]$ is an array that stores the indices found by your greedy algorithm in increasing order; $t$ is the number of indices found by your greedy algorithm.


iii. [1 marks] Analyze the run time.

iv. [4 marks] Prove that your algorithm is correct. Use an exchange proof where you compare your greedy solution $G[1..t]$ to an optimum solution $O[1..s]$.

2. [10 marks] **Huffman Coding.** In CS 240 you learned about Huffman encoding to build an optimal prefix code for a set of characters with given frequencies. This is a good example of a greedy algorithm. Remind yourself of how the algorithm works (use any source you like).

(a) [6 marks] A proof of the correctness of the algorithm can be found in Section 4.4 of Jeff Erickson’s book, [https://jeffe.cs.illinois.edu/teaching/algorithms/book/04-greedy.pdf](https://jeffe.cs.illinois.edu/teaching/algorithms/book/04-greedy.pdf) and in Section 16.3 of the [CLRS] textbook (the link to the online library version of [CLRS] is on the CS 341 course web page). Which proof do you prefer? Briefly support your choice. We are expecting you to write approximately one paragraph here.

(b) [0 marks, optional] Find a proof you like better on the internet (or from Huffman’s original paper, which you can find from the wikipedia article). Post any answers to this question on Piazza.

(c) [4 marks] At first, it might seem that divide-and-conquer would be a good approach for building optimal prefix codes. Suppose that the set of frequencies splits into two equal halves. More precisely, suppose the set of frequencies is $F = \{f_1, f_2, \ldots, f_n\}$ with $\sum_{i=1}^{n} f_i = 1$, and suppose that $\{1, 2, \ldots, n\}$ can be split into two sets $A$ and $B$ with $\sum_{i \in A} f_i = \sum_{i \in B} f_i = \frac{1}{2}$. A divide and conquer approach is to find optimal trees for each of $A$ and $B$, and then join the two trees with one new root node. Find an example where this approach does not produce an optimal prefix code.

**Challenge Questions.** This is for fun and enrichment only. Do not hand it in.

See 2(b) above.