COMP 6651: Worksheet 3

Note: you should always try to give as efficient algorithm as possible.

1. Draw a prefix tree generated by Huffman's algorithm for each of the following alphabets and the corresponding frequencies:
   (a) Alphabet containing symbols a, b, c, d, e with frequencies $30, 15, 110, 35, 20$, respectively.
   (b) Alphabet containing symbols $x, y, z, w, q, r, s, t, v$ with frequencies $20, 10, 300, 40, 15, 35, 10, 5, 600$, respectively.

2. Trit Computing Inc. has developed a new computer architecture that operates on trits instead of bits. Regular bits can assume values 0 or 1 only, while trits can assume values 0, 1, or 2. In particular, Trit Computing Inc. produces solid state drives that store trits instead of regular bits. You, as an expert in data compression, have been hired to develop a modified Huffman algorithm to take advantage of their new storage technology. Trit Computing Inc. needs to compress sequences of characters from an alphabet of size n ($n \geq 3$, n is odd), where the characters occur with known frequencies $f_1, f_2, \ldots, f_n$. Your algorithm should encode each character with a variable-length prefix-free code over the values 0, 1, 2 so as to obtain maximum possible compression. Your algorithm should return a ternary tree representation of such code.
   (a) Describe your algorithm in plain English (at most 5 short sentences).
   (b) Describe your algorithm in pseudocode. You may use priority queues in your algorithm.
   (c) Prove correctness of your algorithm.
   (d) State the running time of your algorithm. Justify it. State any assumptions on the implementation of abstract data types that are needed to achieve this running time.

3. Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ denote a set of n half-open intervals on the line, i.e., $I_j = [s_j, f_j)$ for some $s_j < f_j$. A proper coloring of $\mathcal{I}$ is a function $c : \mathcal{I} \to \mathbb{N}$ such that if $j \neq k$ and $I_j \cap I_k \neq \emptyset$ then $c(I_j) \neq c(I_k)$. In words, a coloring assigns a color to each interval (encoded as a natural number) and a coloring is proper if any two intersecting intervals receive different colors. Given $\mathcal{I}$, the goal is to efficiently compute a proper coloring that uses as few distinct colors as possible.

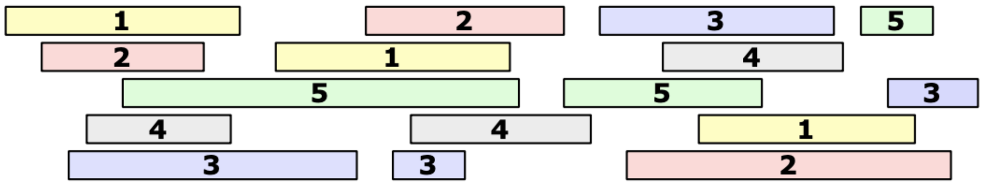

Figure 1: A proper coloring of a set of intervals using five colors.

   (a) Choose a representation for the input to this problem and state the problem formally using the notation **Input: ...**, **Output: ...**.
   (b) Design a greedy algorithm that solves this problem optimally.
   (c) Describe your algorithm in plain English.
   (d) Describe your algorithm in pseudocode.
   (e) Prove that your algorithm is correct.

(f) State and justify the running time of your algorithm.

4. A string w of parenthesis '(' and ')' is *balanced* if it satisfies one of the following conditions:

   - w is the empty string;
   - $w = (x)$ for some balanced string x;
   - $w = xy$ for some balanced strings x and y.

   For example, the string $w = ((())())(()((())))$ is balanced, because $x = ((())())$ and $y = (()((())))$ are balanced.

   (a) Describe (English description + pseudocode) and analyze (correctness + runtime) an algorithm to determine whether a given string of parenthesis is balanced.

   (b) Describe (English description + pseudocode) and analyze (correctness + runtime) a **greedy** algorithm to compute the length of a longest balanced subsequence of a given string of parenthesis.

   The input to each of the above problems is an array $W[1..n]$ such that for each i we have $W[i] =$'(' or $W[i] =$')'. Both of the algorithms should run in $O(n)$ time.

5. Suppose you are a shopkeeper in a country with k different types of coins with denominations

$$1 = c[1] < c[2] < \cdots < c[k].$$

   You keep an infinite supply of each type of a coin, but whenever making change you want to use the fewest number of coins. You need to make change for the value n.

   Example: there are $k = 3$ different types of coins with denominations $c[1] = 1, c[2] = 5, c[3] = 25$. You need to make change for 117. You could use 4 coins of denomination $c[3] = 25$, 3 coins of denomination $c[2] = 5$, and 2 coins of denomination $c[1] = 1$. This uses a total of 9 coins and this is optimal. Another solution could be to use 117 coins of denomination $c[1]$, but it is clearly suboptimal.

   (a) Assume that all denominations are consecutive powers of a common base $b \geq 2$ (b is an integer), i.e., $c[1] = b^0 = 1, c[2] = b^1 = b, c[3] = b^2, \ldots, c[k] = b^{k-1}$. Describe (English description + pseudocode) and analyze (correctness + runtime) an efficient **greedy** algorithm to solve this problem. Carefully prove the optimality of the algorithm.

   (b) Give an example of currency denominations and the value n such that the greedy algorithm does not return an optimal solution.

6. Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of n closed intervals, i.e., $I_j = [s_j, f_j]$ for some $s_j < f_j$. We say that a set of points P *covers* $\mathcal{I}$ if every interval in $\mathcal{I}$ contains at least one point in P. Describe (English description + pseudocode) and analyze (correctness + runtime) an efficient **greedy** algorithm that computes the smallest set of points that covers $\mathcal{I}$.

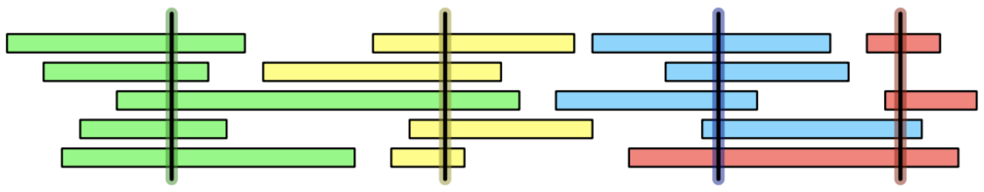

Figure 2: A set of intervals covered by four points shown as vertical line segments.

7. You are given a set of m constraints over n Boolean variables $x_1, \ldots, x_n$. The constraints are of two types:

   - **equality constraints**: $x_i = x_j$ for some $i \neq j$;
   - **inequality constraints**: $x_i \neq x_j$ for some $i \neq j$.

   Describe (English description + pseudocode) and analyze (correctness + runtime) an efficient **greedy** algorithm that given the set of equality and inequality constraints determines if it is possible or not to satisfy all the constraints simultaneously. If it is possible to satisfy all the constraints, your algorithm should output an assignment to the variables that satisfies all the constraints.

8. Recall, that Horn formulas are formulas over Boolean variables with two types of constraints:

   - **implication clauses**: the left-hand side is an AND of arbitrary many (including zero) positive literals and the right-hand side is a single positive literal, e.g., $(x_1 \wedge x_2 \wedge \cdots \wedge x_k) \Rightarrow x_\ell$.
   - **pure negative clauses**: an OR of arbitrary many negative literals, e.g., $\overline{x_1} \vee \overline{x_2} \vee \cdots \overline{x_k}$.

   Also, recall the greedy algorithm for this problem:

```
procedure Satisfiability($\phi$)                                    ▷ $\phi$ is a Horn formula
    set all variables to false
    while there exists a violated implication clause do
        set the variable on the right-hand side of the violated implication clause to true
    if all purely negative clauses remain satisfied by the current assignment then
        return the assignment
    else
        return “not satisfiable”
```

   The above description is at a very high level. It doesn't include a discussion of what data structures are used to represent the input and output. It doesn't explain how to perform each of the steps efficiently.

   Show how to implement the above algorithm in time that is linear in the length of the formula (the number of occurrences of literals in it). Include all the details (including low level data structures used to represent the data) and argue why your implementation runs in linear time.

9. There is a list of n jobs that need to be scheduled on a single machine. Job j is described by a single number – its processing time $p_j \geq 0$, which is how long this job takes to execute on the machine. The jobs are scheduled one after another without any gaps and without preemption. The goal is to schedule jobs in an order that minimizes the cumulative waiting time. The waiting time of job j, denoted by $wait(j)$, is defined as

$$wait(j) = p_{i_1} + p_{i_2} + \cdots + p_{i_k}, \text{ where } i_1, i_2, \ldots, i_k \text{ are all the jobs scheduled before } j.$$

   If no jobs are scheduled prior to j then $wait(j) = 0$. The cumulative waiting time of the schedule is the sum of waiting times of all jobs, i.e., $\sum_{j=1}^{n} wait(j)$.

   Describe (English description + pseudocode) and analyze (correctness + runtime) an efficient **greedy** algorithm that computes a schedule minimizing the cumulative waiting time.