Due: In tutorial on February 17 by 12:10pm

1. [15 marks]

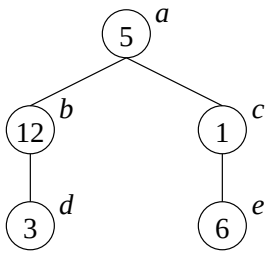
   An *independent set* S is a set of vertices of a graph such that no two vertices in S are adjacent (no pair of vertices are joined by an edge).

   Consider the following "weighted IS on trees" problem:

   Input: an undirected tree $T = (V, E)$ with vertex weights $w(v)$ for $v \in V$.

   Output: an independent set S of maximum total weight.

   For example, in the tree on the right, $S = \{b, e\}$ is an independent set of weight 18 and $S' = \{a, d, e\}$ has weight 13.



   (a) Give an example to show that the following "heaviest first" greedy algorithm *does not* always find an independent set of maximum total weight.

   $S \leftarrow$ the empty set
   while *some vertex remains in* G **do**
   pick a vertex v of maximum weight in G
   add v to S
   delete v and all its neighbours from G
   end
   return S

   (b) Give an efficient algorithm that solves the weighted IS on trees problem using dynamic programming. Briefly justify that your recursive formulation is correct, and state the running time of your algorithm. The running time should be polynomial in n, the number of vertices in the tree, and should be independent of the vertex weights.

2. [15 marks]

   The people living on the island of Zorapia have a unique number system that includes only three digits. Our computer font doesn't have the characters to illustrate these digits, so we will call them `a`, `b` and `c`. Any two of these digits can be combined (multiplied together) to yield a single digit, according to the following table (the row is the left-hand symbol and the column is the right-hand symbol).

	a	b	c
a	c	b	a
b	a	c	a
c	b	b	c

   For example, `ab` = `b`, `ba` = `a`, and so on. Note that the "multiplication" given by this table is neither associative nor commutative.

   The Zorapians believe that strings that can be parenthesized to yield the value `b` are good luck. For example, $w =$ `abba` is a lucky string, since either (`a`((`bb`)`a`)) = `b` or (((`ab`)`b`)`a`) = `b`.

   Design an efficient algorithm that, when given a string $w = w_1 w_2 \cdots w_n$ of characters each of which is either `a`, `b` or `c`, determines whether it is possible to parenthesize w such that the value of the resulting expression is `b`. Prove your algorithm correct and justify its running time.

3. [15 marks]

   In a word processor, the goal of "pretty-printing" is to take text with a ragged right margin, like this,

   ```
   Call me Ishmael.
   Some years ago,
   never mind how long precisely,
   having little or no money in my purse,
   and nothing particular to interest me on shore,
   I thought I would sail about a little
   and see the watery part of the world.
   ```

   and turn it into text whose right margin is as "even" as possible, like this,

   ```
   Call me Ishmael. Some years ago, never
   mind how long precisely, having little
   or no money in my purse, and nothing
   particular to interest me on shore, I
   thought I would sail about a little
   and see the watery part of the world.
   ```

   To make this precise enough for us to start thinking about how to write a pretty-printer for text, we need to figure out what it means for the right margins to be "even." So suppose our text consists of a sequence of *words*, $W = \{w_1, w_2, \ldots, w_n\}$, where word w_i consists of c_i characters. We have a maximum line length of L characters. We will assume we have a fixed-width font and ignore issues of punctuation or hyphenation.

   A *formatting* of W consists of a partition of the words in W into *lines*. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

   $$\left(\sum_{i=j}^{k-1}(c_j + 1)\right) + c_k \leq L$$

   We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the *slack* of the line—that is, the number of spaces left at the right margin.

   Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the *squares* of the slacks of all lines (including the last line) is minimized. Prove your algorithm correct and justify its running time.

   [This is exercise number 6 on pages 317–318 of the textbook.]

4. [15 marks]

   You're consulting for a group of people (who would prefer not to be mentioned here by name) whose jobs consist of monitoring and analyzing electronic signals coming from ships in coastal Atlantic waters. They want a fast algorithm for a basic primitive that arises frequently: "untangling" a superposition of two known signals. Specifically, they're picturing a situation in which each of two ships is emitting a short sequence of 0s and 1s over and over, and they want to make sure that the signal they're hearing is simply an *interleaving* of these two emissions, with nothing extra added in.

   This describes the whole problem; we can make it a little more explicit as follows. Given a string x consisting of 0s and 1s, we write x^k to denote k copies of x concatenated together. We say that a string x' is a *repetition* of x if it is a prefix of x^k for some number k. So $x' = 10110110110$ is a repetition of $x = 101$.

   We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) subsequences s_1 and s_2, so that s_1 is a repetition of x and s_2 is a repetition of y. (So each symbol in s must belong to exactly one of s_1 or s_2.) For example, if $x = 101$ and $y = 00$, then $s = 100010101$ is an interleaving of x and y, since characters 1,2,5,7,8,9 form 101101—a repetition of x—and the remaining characters 3,4,6 form 000—a repetition of y.

   In terms of our application, x and y are the repeating sequences from the two ships, and s is the signal we're listening to: We want to make sure s "unravels" into simple repetitions of x and y. Give an efficient algorithm that takes strings s, x and y and decides if s is an interleaving of x and y. Prove your algorithm correct and justify its running time.

   [This is based on exercise number 19 on page 329 of the textbook.]