Romina Charles, Matt Elvidge, Dominic Faustino
COSC 336
10/24/2019

Assignment 5

Problem 1: Here is the solution for problem 1.

Mathematical text is written like this $a + b = c$.

This is how we can have subscripts and superscripts: $a^2 + b^2 = c^2 + d_1$.

For union of sets, write $A \cup B$; for intersection $A \cap B$; the empty set is denoted $\emptyset$.

Greek letters are easy to write: α, β, γ, θ, Θ, ω, Ω, and so on.

See `https://artofproblemsolving.com/wiki/index.php/LaTeX:Symbols` for a large list of latex symbols.

This is how we can write math equations on a separate line:

$$a + b = c^2 + \log n.$$

This is how we can write multi-line math equations on a separate line:

$$\begin{aligned} a + b &= c^2 + \log n \\ &\leq 5d + \sin x \\ &= A. \end{aligned}$$

Matrices can be written like this:

$$A = \begin{pmatrix} 1 & a & b \\ 2 & a^2 & b^3 \\ 3 & a_3 & b_5 \end{pmatrix}$$

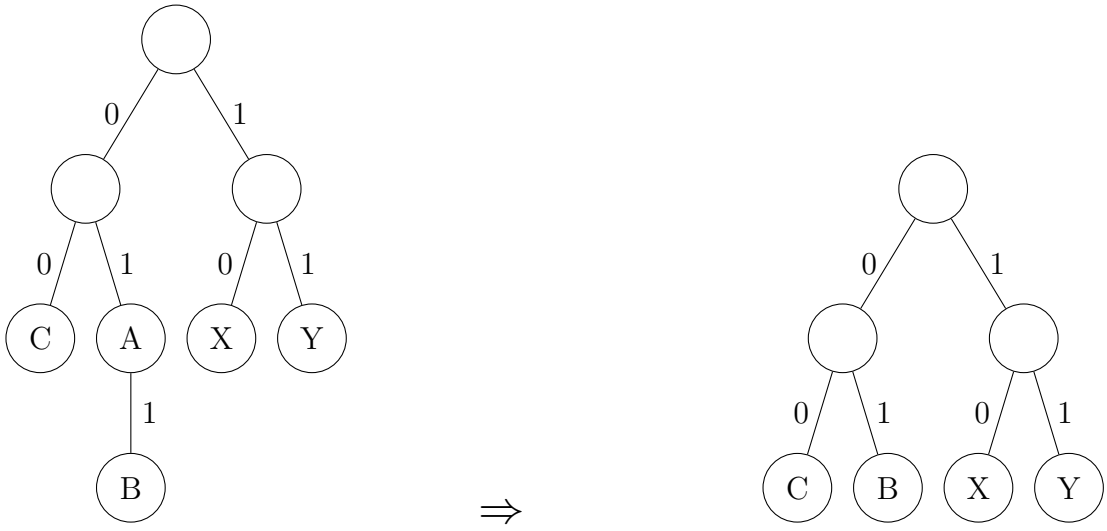
This is how to make a table:

1	2	3		5

Exercise 16.3-2, page 436: Prove that a binary tree that is not full cannot correspond to an optimal prefix code.

We can call some node A with one child node B. Node A causes the tree to not be full, since it only one child. If A only has one child B, we would

be better off simply taking away A, and B is assigned the code where A was. This new tree would have an optimal code.



Exercise 11.3-4, page 269:

$$h(61) = 1000(61 * [((5)^{\frac{1}{2}} - 1)/2] \ \ \%1) = 700$$
$$h(62) = 1000(62 * [((5)^{\frac{1}{2}} - 1)/2] \ \ \%1) = 318$$
$$h(63) = 1000(63 * [((5)^{\frac{1}{2}} - 1)/2] \ \ \%1) = 936$$
$$h(64) = 1000(64 * [((5)^{\frac{1}{2}} - 1)/2] \ \ \%1) = 554$$
$$h(65) = 1000(65 * [((5)^{\frac{1}{2}} - 1)/2] \ \ \%1) = 172$$

Programming Tasks 1 and 2

Programming tasks 1 and 2 are both functions in the following code. Programming task 1 finds the max sum throughout an array of numbers by comparing it to the max that it has currently stored. When it finds a greater sum, the max sum value will be updated.

Programming task 2 has a similar function to programming task one but compares the length and the sum of the sequence. Because the function is looking for the sum of the longest increasing subsequence, the length is the first value that must be checked. If two subsequences are the same length, the function compares the sum value of each and assigns the higher value to

the max sum.