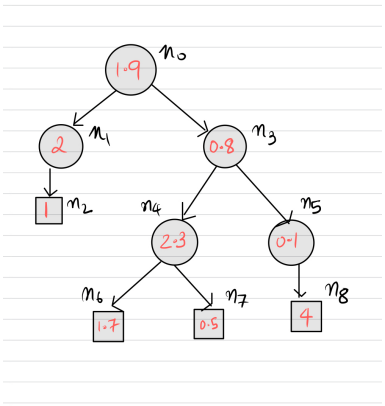
CSCI 5454 Spring 2023: Assignment #6
Due Date: Friday, April 7th

Topics: Dynamic Programming, Greedy Algorithm and NP-hardness.

P1 (20 points) Suppose we are given a binary tree where each node is either a leaf, has one child or two children. Furthermore, each node n_i has a weight w_i. We wish to choose a subset of nodes from the tree such that (a) if a node is chosen then its children/parent cannot be chosen; and (b) the total weight of all the chosen nodes must be maximum possible. For instance consider the tree below with leaf nodes shown as rectangles and internal nodes as circles. We show some of the possible ways we can select nodes in the tree and corresponding total weights.



$n_0, n_2, n_4, n_8 \rightarrow 9.2$
$n_1, n_4, n_8 \rightarrow 8.3$
$n_0, n_2, n_6, n_7, n_8 \rightarrow 9.1$
$n_0, n_1, n_4, n_8 \rightarrow -\infty$ (constraint violated)

(A, 15 points) Design a dynamic programming algorithm that computes the subset of nodes with maximum weight with the constraint that when a node is chosen, its parents/children cannot be chosen. What is the running time of your algorithm in terms of the total number of nodes in the tree?

Proceed as follows:

1. For each node n of the tree with weight w_n, write down a recurrence for $\mathsf{maxWeightNotIncluding}(n)$ that calculates the maximum weight subset for the subtree rooted at n such that the node n itself is forced not to be included and $\mathsf{maxWeightIncluding}(n)$ wherein the node n itself is forced to be included. Your recurrence will be different for leaf nodes and for non-leaf nodes, it will express these quantities in terms of the corresponding quantities for the children nodes.

2. How would you memoize the recurrence?

3. How would you recover the maximum weight subset starting from the root of the tree?

Hint: In the beginning, we decide whether to include the root or not. If the root is included, then what are the "left over" problems? If the root is not included what are the "left over" problem?

(B, 5 points) Suppose we formulate a greedy solution that goes as follows:

- Start at the root.
- When visiting the node n compare the weight of the node w_n with the sum of the weight of its children. For a leaf node, sum of weight of children is taken to be zero.

- If w_n is greater than the weight of the children, force node n to be included. Then recursively skip the children of n and visit the children of children.
- Else, n is excluded and recursively visit the children.

Formulate an example where the greedy algorithm produces a sub-optimal result. Show the example along with what the greedy and dynamic programming approaches produce.

P2 (20 points) You are given a string s with n elements and asked to make a palindrome out of it. This is done by inserting characters at various places in the string.

Eg., to make a palindrome out of the string "STAIR", we proceed as follows:

- Insert characters "RIA" at position 0.
- Insert character "S" at position 2.

The result is the string "RIASTSAIR"which is a palindrome.

There are other ways to make "STAIR" a palindrome but the string "RIASTSAIR" is the smallest such palindrome made from inserting characters at various places in the original string.

Here are a few more examples:

- "MOTHER" $\rightarrow$ "MOREHTHEROM"
- "FAMILY" $\rightarrow$ "FAMILYLIMAF"
- "BEAKER" $\rightarrow$ "BREAKAERB"
- "BARBARIAN" $\rightarrow$ "BNAIRABARIANB"

Formulate a dynamic programming algorithm for finding the smallest number of insertions that can make a given string a palindrome as well as the resulting palindrome.

(A) For any string of the form $s[0], \ldots, s[n-1]$ of length n, focus on $s[0]$ and $s[n-1]$. If they are the same, then we can simply recurse on $s[1] \ldots s[n-1]$. Otherwise, we need to either insert $s[0]$ at the end and recurse on $s[1], \ldots, s[n-1]$ OR insert $s[n-1]$ at the beginning and recurse on $s[0], \ldots, s[n-2]$.

Write a recursive definition of the function $\mathsf{MinPalindromeIns}(s)$ that counts the minimum number of insertions needed to make a string s into a palindrome. Note that strings of length $0, 1$ are automatically palindromes.

(B) Memoize the recurrence above by building a memo table. How would you fill the memo table? What is the overall running time?

(C) How can we read off the solution from the memo table.