CS 473 ✧ Spring 2017

Homework 1

Due Wednesday, February 1, 2017 at 8pm

Starting with this homework, groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

1. A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA.

   Any string can be decomposed into a sequence of palindromes. For example, the string BUBBASEESABANANA ("Bubba sees a banana.") can be broken into palindromes in the following ways (and many others):

   BUB • BASEESAB • ANANA

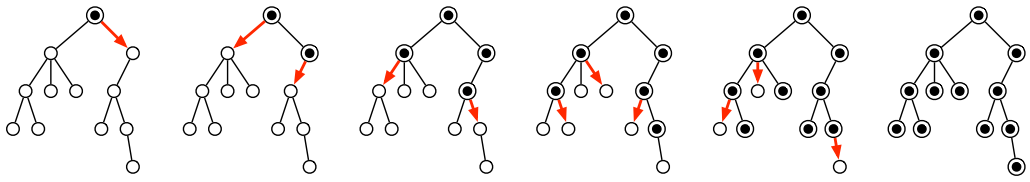
   B • U • BB • A • SEES • ABA • NAN • A

   B • U • BB • A • SEES • A • B • ANANA

   B • U • B • B • A • S • E • E • S • A • B • A • N • ANA

   Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string BUBBASEESABANANA, your algorithm would return the integer 3.

2. Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children.



A message being distributed through a tree in five rounds.

   (a) Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in a ***binary*** tree.

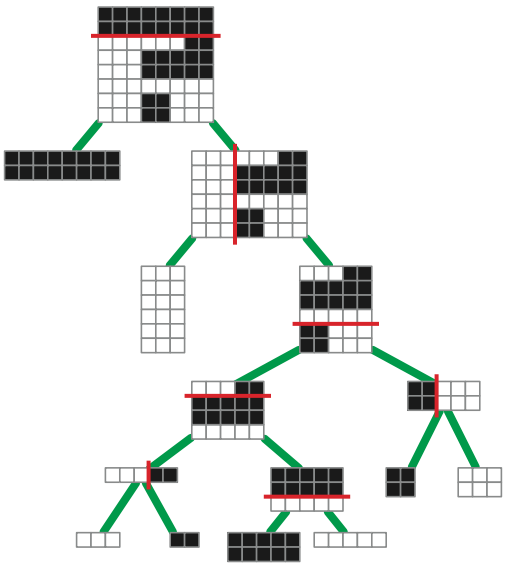
   (b) Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in an ***arbitrary rooted*** tree.

   [Hint: Don't forget to justify your algorithm's correctness; you may find the lecture notes on greedy algorithms helpful. Any algorithm for this part also solves part (a).]

3. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..m, 1..n]$ of 0s and 1s. A *solid block* in M is a contiguous subarray $M[i..i', j..i']$ in which all bits are equal.

   A *guillotine subdivision* is a compact data structure to represent bitmaps as a recursive decomposition into solid blocks. If the entire bitmap M is a solid block, there is nothing to do. Otherwise, we cut M into two smaller bitmaps along a horizontal or vertical line, and then decompose the two smaller bitmaps recursively.[1]

   Any guillotine subdivision can be represented as a binary tree, where each internal node stores the position and orientation of a cut, and each leaf stores a single bit 0 or 1 indicting the contents of the corresponding block. The *size* of a guillotine subdivision is the number of leaves in the corresponding binary tree (that is, the final number of solid blocks), and the *depth* of a guillotine subdivision is the depth of the corresponding binary tree.



A guillotine subdivision with size 8 and depth 5.

   (a) Describe and analyze an algorithm to compute a guillotine subdivision of M of minimum *size*.

   (b) Describe and analyze an algorithm to compute a guillotine subdivision of M of minimum *depth*.

[1] Guillotine subdivisions are similar to kd-trees, except that the cuts in a guillotine subdivision are *not* required to alternate between horizontal and vertical.

Standard dynamic programming rubric. For problems worth 10 poins:

- 3 points for a clear **English** specification of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - + 2 points for describing the function itself. (For example: "$OPT(i, j)$ is the edit distance between $A[1..i]$ and $B[1..i]$.")
  - + 1 point for stating how to call your recursive function to get the final answer. (For example: "We need to compute $OPT(m, n)$.")
  - + An English description of the ***algorithm*** is not sufficient. We want an English description of the underlying recursive ***problem***. In particular, the description should specify precisely the role of each input parameter.
  - + No credit for the rest of the problem if the English description is is missing. (This is a Deadly Sin.)
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for details of the iterative dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 1 point for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but iterative psuedocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. (But you still need to specify the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).