CS 473 ✧ Spring 2020

Homework 6

Due Wednesday, March 25, 2020 at 9pm

(after spring break)

For the rest of the semester, you are welcome to use randomized algorithms in your homework and exam solutions, but please report your running times carefully. Without further qualification, “$O(n^2)$ time” means $O(n^2)$ time *in the worst case*. If you mean $O(n^2)$ *expected* time, or $O(n^2)$ time *with high probability*, you must write that explicitly.

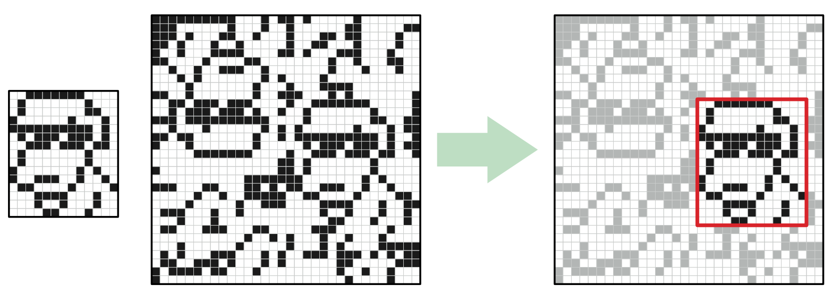
---

1. Describe and analyze an *even faster* algorithm to find the length of the longest substring that appears both forward and backward in an input string $T[1..n]$. The forward and backward substrings must not overlap. Here are several examples:

   - Given the input string `ALGORITHM`, your algorithm should return 0.
   - Given the input string `RECURSION`, your algorithm should return 1, for the substring `R`.
   - Given the input string `REDIVIDE`, your algorithm should return 3, for the substring `EDI`. (Remember: The forward and backward substrings must not overlap!)

   Yes, this *exact* problem appeared in Midterm 1, so you should already know how to solve it in $O(n^2)$ time. You can do better now.

2. Describe an efficient algorithm to determine if a given $p \times q$ rectangular pattern of bits appears anywhere in an $m \times n$ bitmap. (The pattern may be shifted horizontally and/or vertically, but it may not be rotated or reflected.)



3. Describe an efficient algorithm to decide, given two rooted ordered trees P and T, whether P (the “pattern”) occurs anywhere as a subtree of T (the “text”).

   A *rooted ordered tree* is a rooted tree where every node has a (possibly empty) *sequence* of children. The order of these children matters: Two rooted ordered trees are identical if and only if their roots have the same number of children and, for each index i, the subtrees rooted at the ith children of both roots are identical.

   For purposes of this problem, a *subtree* of T contains some node and ***all*** its descendants in T, along with the edges of T between those vertices.

There is no data stored in the nodes, only pointers to children (if any). We want an algorithm that compares the *shapes* of the trees.

For example, in the figure below, P appears exactly once as a subtree of T.