CS 6363.003 Final Exam

May 13, 2021

Please read the following instructions carefully before you begin.

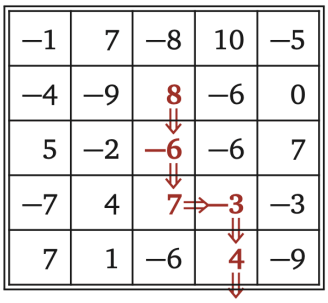
- The exam has 6 problems, some of which have multiple parts. Please start the solution to each problem on a new page.
- The exam is meant to take around 2 hours and 30 minutes, but you have some extra time to prepare the solutions for submission to eLearning and work through any related technical issues. You must finish uploading your solutions within 3 hours of beginning the exam or by 5:59am CST on Friday, May 14, *whichever comes first*. You may upload as many versions of your solutions as you would like, but you can only submit them once. Your solutions will be submitted automatically 3 hours after you begin the exam to prevent you from uploading but forgetting to actually submit your solutions.
- If you cannot successfully upload your solutions to eLearning before the end of the 3 hour time limit, email a copy to Kyle as soon as you possibly can.
- It is highly recommended that you take the exam between 9am and 5pm CDT on Thursday, May 13th. Kyle will be actively watching for questions via email during this time. He will also try to answer emailed questions at other times during the exam period, but he may be slower to respond, especially if he is asleep! Feel free to ask for clarification on any of the problems.
- Questions are not necessarily given in order of difficulty, so read through them all before you begin writing.
- This exam is closed book. No notes or calculators are permitted.
- If asked to describe an algorithm, you should state your algorithm clearly (preferably with pseudocode) and briefly explain its asymptotic running time in big-O notation in terms of the input size. Each individual problem will specify whether or not you must justify correctness.
- It's the end of the semester. Celebrate! (Throwing a party might be a bit much, though. See Problem 6.)

1. **(10 points)** Suppose are you given an array $A[1 .. n]$ of numbers. You are free to view and compare the numbers like normal, but the only way you are allowed to modify A is by calling a subroutine REVERSE(i) that reverses the order of the elements in $A[1 .. i]$. For example, if A initially equals $\langle 5, 9, 12, 43, 2, 6363, 19\rangle$, then the call REVERSE(4) would modify A so that it then equals $\langle \mathbf{43}, \mathbf{12}, \mathbf{9}, \mathbf{5}, 2, 6363, 19\rangle$.

   Describe and analyze an algorithm to sort the array A using $O(n)$ REVERSE operations. Your analysis only needs to discuss the number of REVERSE operation performed by the algorithm. You do not need to justify correctness of your algorithm. *[Hint: This is a problem about recursion. Try to perform a small number of flips so you can recursively finish sorting A.]*

2. Consider the following solitaire game played on an $n \times n$ square grid. The player starts by placing a token on any square of the grid. On each turn, the player moves the token either one square to the right or one square down. The game ends when the player moves the token off the edge of the board. Each square of the grid has a numerical value represented by a 2D array $A[1 .. n, 1 .. n]$ where $A[i, j]$ is the value for the grid square that is ith from the top and jth from the left (i.e., the top left grid square has value $A[1, 1]$, the bottom left has value $A[n, 1]$, etc.). ***Each of these values may be positive, negative, or zero.*** The player starts with a score of zero; whenever the token lands on a square, the player adds its value to their score. The object of the game is to score as many points as possible.

   For example, given the grid below, the player can score $8 - 6 + 7 - 3 + 4 = 10$ points by initially placing the token on the 8 in position $(2, 3)$, and then moving down, down, right, down, down. (This is *not* the best possible score for this grid of numbers.)



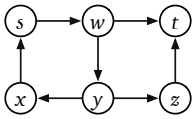
   (a) **(5 out of 10)** For $1 \le i \le n$ and $1 \le j \le n$, let $BestScore(i, j)$ denote the maximum score obtainable if the player starts by placing their token at position (i, j) (i.e., the grid square of value $A[i, j]$). For simplicity, we'll also define $BestScore(n + 1, \cdot)$ and $BestScore(\cdot, n + 1)$ to both be 0.

   Give a recursive definition for $BestScore(i, j)$. You do not need to justify correctness of your definition.

   (b) **(5 out of 10)** ***Using your recursive definition from part (a)*** and dynamic programming, describe and analyze an iterative algorithm that computes the maximum possible score obtainable given the array of values $A[1 .. n, 1 .. n]$. Unless you gave a recursive definition for which there is no efficient memoization method, you may assume your solution to part (a) is correct. You do not need to justify correctness of your algorithm.

3. **(10 points)** Suppose you are given a directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}_{\geq 0}$ and two designated vertices $s, t \in V$. Describe and analyze an algorithm that either returns the length of the shortest *walk* in G from s to t whose number of edges is divisible by 3 or reports that no such walk exists. Recall, a walk is allowed to repeat vertices and/or edges.

   For example, given the graph shown below, with indicated vertices s and t, and with all edges having weight 1, your algorithm should return 6, which is the length of the walk $s{\to}w{\to}y{\to}x{\to}s{\to}w{\to}t$.



   [Hint: Build a new graph G' with $O(V)$ vertices and run a black-box shortest paths algorithm on G'.]

4. **(10 points)** Exam scheduling is back! On the one hand, it's easier than it was before, because all exams are to be done through the eUnderstanding online course management system using the MagicProctor student monitoring software. That means you no longer need to worry about placing students in cramped rooms or picking distractable humans to watch over them as they take their exams. On the other hand, it's harder than it was before, because all exams are to be done through the eUnderstanding online course management system which will fail spectacularly if too many students try to take an exam at the same time.

   To help ease the load on the eUnderstanding system, university administrators have decide that every exam will be taken during one of t different time slots, but no more than 2,000 exams may be taken during any one time slot. Administrators also know how many exams must be taken by each of the n students, and they have obtained a list of time slots during which each student is available to take an exam. A student may not take two exams during a single time slot.

   Formally, the input to the problem is

   - an integer array $E[1 \,..\, n]$ where $E[i]$ is the number of exams to be taken by student i, and
   - a Boolean array $A[1 \,..\, n, 1 \,..\, t]$ where $A[i, j] = \textsc{True}$ if and only if student i is available to take one of their exams during the jth time slot.

   Describe and analyze an algorithm to determine whether there is a way to assign each student i to $E[i]$ distinct time slots so that 1) $A[i, j] = \textsc{True}$ for each student i assigned to each time slot j and 2) at most 2,000 students are assigned to each time slot j. You do not need to justify correctness of your algorithm.

   (Your algorithm does not need to return the actual assignments, assuming they exist, but it will likely be clear to both of us how to compute the assignments if your algorithm is correct.)

5. Suppose are you given an undirected graph $G = (V, E)$. Recall, a subset of vertices $S \subseteq V$ is a ***vertex cover*** if every edge of G touches at least one vertex in S. The MinVertexCover problem asks for the smallest vertex cover in G.

   (a) **(4 out of 10)** Suppose the given graph G is a tree. Our goal is to design an efficient dynamic programming algorithm for this case of MinVertexCover. Let r be an arbitrary vertex of G, and imagine rooting G at r. Note that each vertex may have an arbitrary number of children.

   For any vertex v of G, let $MinVCNo(v)$ denote the minimum size of any vertex cover S of v's subtree *such that* $v \notin S$. Similarly, let $MinVCYes(v)$ denote the minimum size of any vertex cover S of v's subtree such that $v \in S$.

   Let $w \downarrow v$ denote "w is a child of v". Fill in the blanks to complete the following recursive definitions of both $MinVCNo(v)$ and $MinVCYes(v)$. Each blank is either an integer (positive, negative, or zero) or a recursive call to one of $MinVCNo$ or $MinVCYes$.

$$MinVCNo(v) = \_\_\_\_ + \sum_{w \downarrow v} \_\_\_\_$$

$$MinVCYes(v) = \_\_\_\_ + \sum_{w \downarrow v} \min\{MinVCNo(w), \_\_\_\_\}$$

   (b) **(3 out of 10)** Briefly describe and analyze an algorithm to compute the size of the smallest vertex cover in rooted tree G. For your description, it suffices to state what order to solve each of the subproblems $MinVCNo(v)$ and $MinVCYes(v)$ and how to compute the final return value of the algorithm using one or more of these subproblem solutions. You do not need to justify correctness of your algorithm.

   (c) **(3 out of 10)** As shown in class, the decision version of MinVertexCover (determine whether there a vertex cover at size of most k) is NP-hard. Why doesn't a polynomial time solution for part (b) imply P = NP?

6. Throwing a good party is hard!

   (a) **(5 out of 10)** Suppose you and a friend are hosting an end-of-semester party with n guests (it can be held virtually if you'd prefer). For one of the activities, you need to divide the guests into three groups. You don't want to make things too awkward for the guests, though, so you will only put two guests into a common group if they know each other ahead of time. All the guests must be put in exactly one of the groups for the activity to be a success.

   Fortunately, your friend has prepared a document listing every pair of guests that already know one-another. Unfortunately, figuring out the groups is still going to be a difficult task. Argue that, given a document listing which pairs of the guests know one-another, it is NP-hard to determine if there is a way to divide the guests into three groups as described above. Describing and analyzing a reduction from a known NP-hard problem is sufficient for full credit; you do not need to formally justify the correctness of your reduction. *[Hint: Try 3Color.]*

(b) **(5 out of 10)** Your party needs decorations, so you decide to purchase a bunch of balloons. Balloons come in a variety of shapes, but only two colors; green and orange. To keep things interesting, you've decided to buy at most one balloon of each shape.

Unfortunately, your once-helpful friend is now feeling very picky about what kinds of balloons you should purchase for the party, and they have provided a list of m requirements for your purchase. Each requirement lists one or more pairs of colors and shapes, and for each requirement, you must purchase at least one balloon that matches one of its color-shape pairs. For example, one of the many requirements might state "Buy a green star balloon or an orange square balloon or an orange balloon shaped like Temoc's head or a green balloon shaped like a pony."

Given that you want to buy at most one balloon of each shape, meeting all these requirements may not be easy. Argue that, given a list of requirements for what kinds of balloons to buy, it is NP-hard to determine if all the requirements can be met while still buying at most one balloon of each shape. Describing and analyzing a reduction from a known NP-hard problem is sufficient for full credit; you do not need to formally justify the correctness of your reduction. *[Hint: Try* 3Sat*.]*