CS 473 ✧ Fall 2024

☙ Homework 0 ❧

Due Wednesday, September 4, 2024 at 9pm Central Time

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms; fundamental graph problems and algorithms; and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
- **Every homework submission must include a list of sources and collaborators.** Yes, even if that list is empty.
- **Submit your solutions electronically on Gradescope as PDF files.**
  - Submit a separate PDF file for each numbered problem.
  - You can find a LaTeX solution template on the course web site; please use it if you plan to typeset your homework.
  - If you plan to submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), and use a high-quality scanner or scanning app to create a high-quality PDF for submission (not a raw cell-phone photo). We reserve the right to reject submissions that are difficult to read.

☞ Some important course policies ☜

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- **Avoid the Deadly Sins!** There are a few common writing (and thinking) practices that will be automatically penalized on every homework or exam problem. We're not just trying to be scary control freaks; history strongly suggests that people who commit these sins are more likely to make other serious mistakes as well. We're trying to break bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Write for humans, not compilers. In particular, don't submit code.
  - Every algorithm requires an English specification.
  - Never use weak induction. Why tie $n-2$ hands behind your back?!

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or online.

---

1. Describe and analyze algorithms[1] for the following problems. The input for each problem is an unsorted array $A[1..n]$ of n arbitrary numbers, which may be positive, negative, or zero, and which are not necessarily distinct.

   (a) Are there two distinct indices $i < j$ such that $A[i] + A[j] = 0$?

   (b) Are there three distinct indices $i < j < k$ such that $A[i] + A[j] + A[k] = 0$?

   For example, if the input array is $[2, -1, 0, 4, 0, -1]$, both algorithms should return TRUE, but if the input array is $[4, -1, 2, 0]$, both algorithms should return FALSE. You do ***not*** need to prove that your algorithms are correct.

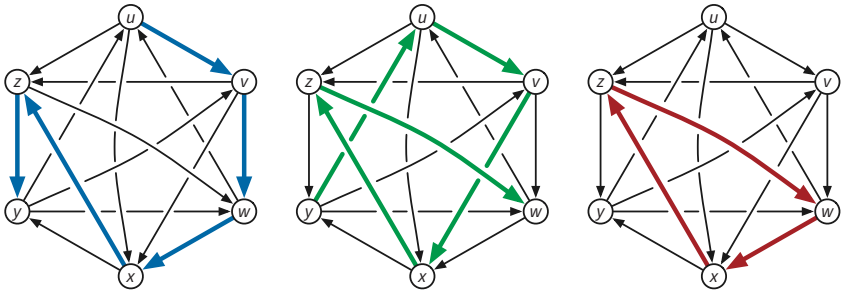
   [Hint: This question is intended to test your ability to describe algorithms clearly and precisely, without assuming that the reader knows your favorite programming language. The devil is in the details!]

2. A ***tournament*** is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices v and w, a tournament contains either an edge $v \to w$ or an edge $w \to v$, but not both. A ***Hamiltonian path*** in a directed graph G is a directed path that visits every vertex of G exactly once.

   (a) ***[Practice only. Do not submit solutions.]*** Describe and analyze an efficient algorithm that takes a tournament T as input and returns a Hamiltonian path in T as output. *[Hint: Why does such a path always exist?]*

   (b) Describe and analyze an efficient algorithm that takes a tournament T as input and returns as output either (1) the ***only*** Hamiltonian path in T or (2) a directed cycle of length 3 in T. Justify the correctness of your algorithm. *[Hint: Why is one of these two outputs always possible?]*

   To simplify both algorithms, assume that the input tournament T is represented as an adjacency *matrix*.



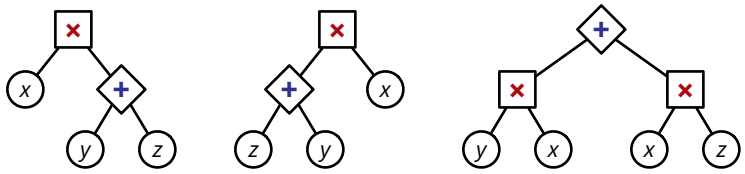
A tournament with two Hamiltonian paths $u \to v \to w \to x \to z \to y$ and $y \to u \to v \to x \to z \to w$ and a directed triangle $w \to x \to z \to w$.

[1] Whenever we ask for an algorithm in this class, we want the fastest algorithm you can find (even if the problem statement doesn't explicitly say "fast" or "efficient"), and we have a target time bound in mind. Correct algorithms that are slower than this target time will get partial credit; correct algorithms that are faster than this target time will get extra credit. *Completely incorrect algorithms are worth no credit, no matter how fast they are.* We generally don't reveal the target time bound, because when we do, a *lot* more people submit fast algorithms that don't work.

3. An ***arithmetic expression tree*** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and $\times$. Different leaves may or may not represent distinct variables.

   Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$-node is the sum of the values of its children. (2) The value of any $\times$-node is the product of the values of its children.

   Two arithmetic expression trees are ***equivalent*** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in ***normal form*** if the parent of every $+$-node (if any) is another $+$-node.



Three equivalent expression trees representing the function $f(x, y, z) = xy + xz$.
Only the third expression tree is in normal form.

   Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. *[Hint: Think recursively. Be careful; this is more subtle than it looks!]*