CS 473 ✧ Spring 2016
Homework 0

Due Tuesday, January 26, 2016 at 5pm

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms (at the level of CS 225); fundamental graph problems and algorithms (again, at the level of CS 225); 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.
- **Submit your solutions electronically on the course Moodle site as PDF files.**
  - Submit a separate 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 must submit scanned handwritten solutions, use a black pen (not pencil) on blank white printer paper (not notebook or graph paper), use a high-quality scanner (not a phone camera), and print the resulting PDF file on a black-and-white printer to verify readability before you submit.

☞ Some important course policies ☜

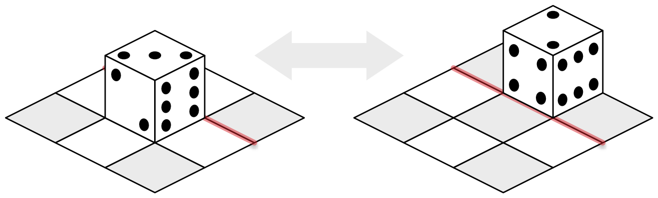
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer ***"I don't know"*** (and *nothing* else) is worth 25% partial credit on any problem or subproblem, on any homework or exam, except for extra-credit problems. We will accept synonyms like "No idea" or "WTF" or "¯\(•_•)/¯", but you must write *something*.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an ***automatic zero*** on any homework or exam problem, unless your solution is nearly perfect otherwise. Yes, we are completely serious.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English.
  - Never use weak induction.

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 on Piazza.

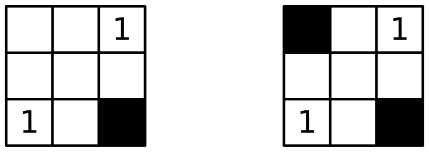
---

1. A ***rolling die maze*** is a puzzle involving a standard six-sided die (a cube with numbers on each side) and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to an adjacent square one step north, south, east, or west.


Rolling a die.

Some squares in the grid may be *blocked*; the die must never be rolled onto a blocked square. Other squares may be *labeled* with a number; whenever the die rests on a labeled square, the number of pips on the *top* face of the die must equal the label. Squares that are neither labeled nor marked are *free*. You may not roll the die off the edges of the grid. A rolling die maze is *solvable* if it is possible to place a die on the lower left square and roll it to the upper right square under these constraints.

For example, here are two rolling die mazes. Black squares are blocked; empty white squares are free. The maze on the left can be solved by placing the die on the lower left square with 1 pip on the top face, and then rolling it north, then north, then east, then east. The maze on the right is not solvable.



Two rolling die mazes. Only the maze on the left is solvable.

Describe and analyze an efficient algorithm to determine whether a given rolling die maze is solvable. Your input is a two-dimensional array $Label[1..n, 1..n]$, where each entry $Label[i, j]$ stores the label of the square in the ith row and jth column, where the label 0 means the square is free, and the label -1 means the square is blocked.

[Hint: Build a graph. What are the vertices? What are the edges? Is the graph directed or undirected? Do the vertices or edges have weights? If so, what are they? What textbook problem do you need to solve on this graph? What textbook algorithm should you use to solve that problem? What is the running time of that algorithm as a function of n? What does the number 24 have to do with anything?]

2. Describe and analyze fast algorithms 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: The devil is in the details.]*

3. A ***binomial tree of order k*** is defined recursively as follows:

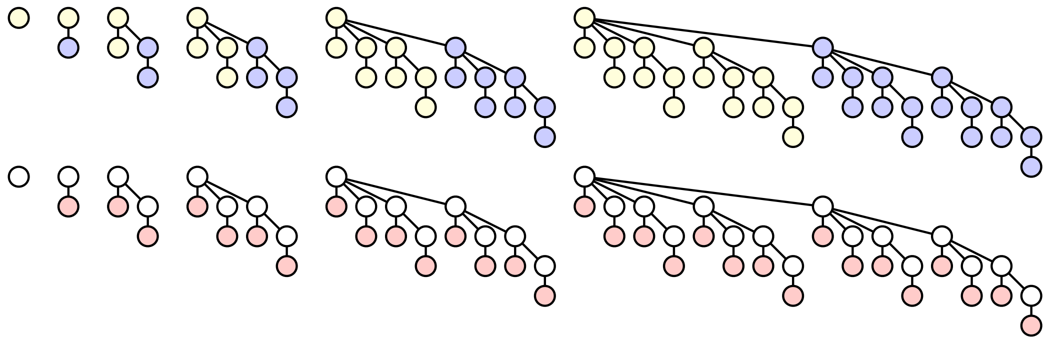
   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k - 1$, with the root of one tree connected as a new child of the root of the other. (See the figure below.)

   Prove the following claims:

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a new leaf to every node in a binomial tree of order $k - 1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d. (Hence the name!)

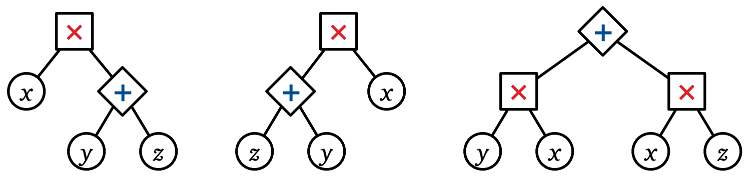


Binomial trees of order 0 through 5.
Top row: The recursive definition. Bottom row: The property claimed in part (b).

*4. *[Extra credit]* 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 different 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. 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: This is harder than it looks.]*