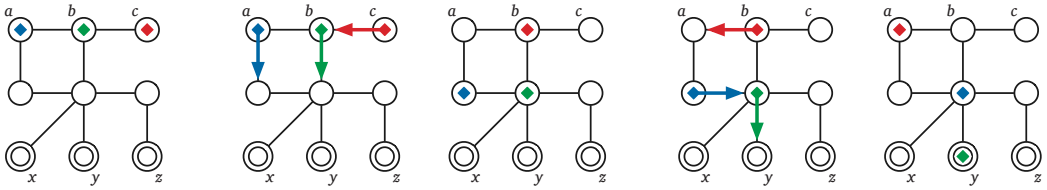
# CS/ECE 374 A ✧ Spring 2018

# Homework 7

Due Tuesday, March 27, 2018 at 8pm
(*after* Spring Break)

---

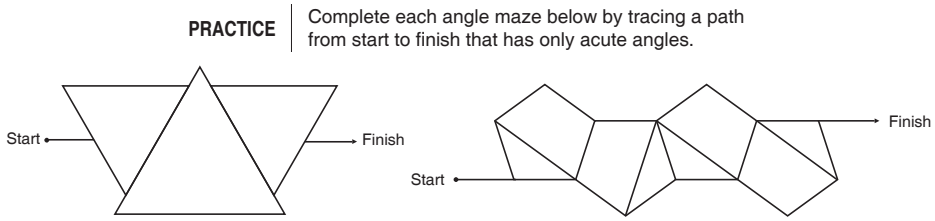
1. Consider the following solitaire game, played on a connected undirected graph $G$. Initially, tokens are placed on three start vertices $a, b, c$. In each turn, you *must* move *all three* tokens, by moving each token along an edge from its current vertex to an adjacent vertex. At the end of each turn, the three tokens *must* lie on three different vertices. Your goal is to move the tokens onto three goal vertices $x, y, z$; it does not matter which token ends up on which goal vertex.




The initial configuration of the puzzle and the first two turns of a solution.

Describe and analyze an algorithm to determine whether this puzzle is solvable. Your input consists of the graph $G$, the start vertices $a, b, c$, and the goal vertices $x, y, z$. Your output is a single bit: TRUE or FALSE. *[Hint: You've seen this sort of thing before.]*

2. The following puzzles appear in my daughter's elementary-school math workbook.[1]




Describe and analyze an algorithm to solve arbitrary acute-angle mazes.

You are given a connected undirected graph $G$, whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the first maze above has 13 vertices and 15 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if $G$ contains a walk from Start to Finish that has only acute angles, and FALSE otherwise. Formally, a walk through $G$ is valid if, for any two consecutive edges $u \to v \to w$ in the walk, either $\angle uvw = \pi$ or $0 < \angle uvw < \pi/2$. Assume you have a subroutine that can determine in $O(1)$ time whether two segments with a common vertex define a straight, obtuse, right, or acute angle.

---

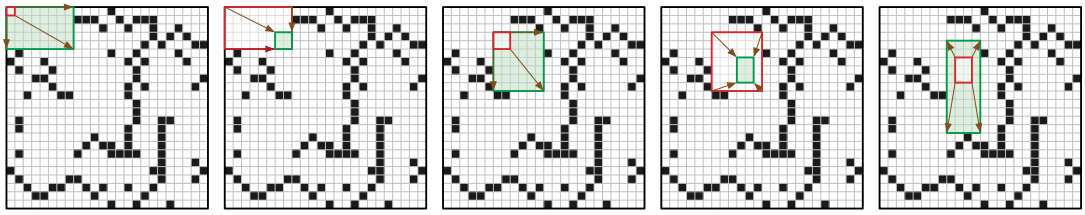
[1] Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See https://www.beastacademy.com/resources/printables.php for more examples.

3. ***Rectangle Walk*** is a new abstract puzzle game, available for only 99¢ on Steam, iOS, Android, Xbox One, Playstation 5, Nintendo Wii U, Atari 2600, Palm Pilot, Commodore 64, TRS-80, Sinclair ZX-1, DEC PDP-8, ILLIAC V, Zuse Z3, Duramesc, Odhner Arithmometer, Analytical Engine, Jacquard Loom, Horologium mirabile Lundense, Leibniz Stepped Reckoner, Antikythera Mechanism, and Pile of Sticks.

   The game is played on an $n \times n$ grid of black and white squares. The player moves a rectangle through this grid, subject to the following conditions:

   - The rectangle must be aligned with the grid; that is, the top, bottom, left, and right coordinates must be integers.
   - The rectangle must fit within the $n \times n$ grid, and it must contain at least one grid cell.
   - The rectangle must not contain a black square.
   - In a single move, the player can replace the current rectangle $r$ with any rectangle $r'$ that either contains $r$ or is contained in $r$.

   Initially, the player's rectangle is a $1 \times 1$ square in the upper right corner. The player's goal is to reach a $1 \times 1$ square in the bottom left corner using as few moves as possible.



The first five steps in a Rectangle Walk.

   Describe and analyze an algorithm to compute the length of the shortest Rectangle Walk in a given bitmap. Your input is an array $M[1..n, 1..n]$, where $M[i, j] = 1$ indicates a black square and $M[i, j] = 0$ indicates a white square. You can assume that a valid rectangle walk exists; in particular, $M[1, 1] = 0$ and $M[n, n] = 0$. For example, given the bitmap shown above, (I think) your algorithm should return the integer 18.

## Solved Problem

4. Professor McClane takes you out to a lake and hands you three empty jars. Each jar holds a positive integer number of gallons; the capacities of the three jars may or may not be different. The professor then demands that you put exactly $k$ gallons of water into one of the jars (which one doesn't matter), for some integer $k$, using only the following operations:

   (a) Fill a jar with water from the lake until the jar is full.

   (b) Empty a jar of water by pouring water into the lake.

   (c) Pour water from one jar to another, until either the first jar is empty or the second jar is full, whichever happens first.

   For example, suppose your jars hold 6, 10, and 15 gallons. Then you can put 13 gallons of water into the third jar in six steps:

   - Fill the third jar from the lake.
   - Fill the first jar from the third jar. (Now the third jar holds 9 gallons.)
   - Empty the first jar into the lake.
   - Fill the second jar from the lake.
   - Fill the first jar from the second jar. (Now the second jar holds 4 gallons.)
   - Empty the second jar into the third jar.

   Describe and analyze an efficient algorithm that either finds the smallest number of operations that leave exactly $k$ gallons in any jar, or reports correctly that obtaining exactly $k$ gallons of water is impossible. Your input consists of the capacities of the three jars and the positive integer $k$. For example, given the four numbers $6, 10, 15$ and $13$ as input, your algorithm should return the number 6 (for the sequence of operations listed above).

   **Solution:** Let $A, B, C$ denote the capacities of the three jars. We reduce the problem to breadth-first search in the following directed graph:

   - $V = \{(a,b,c) \mid 0 \le a \le A \text{ and } 0 \le b \le B \text{ and } 0 \le c \le C\}$. Each vertex corresponds to a possible ***configuration*** of water in the three jars. There are $(A+1)(B+1)(C+1) = O(ABC)$ vertices altogether.
   - The graph has a directed edge $(a,b,c) \to (a',b'c')$ whenever it is possible to move from the first configuration to the second in one step. Specifically, there is an edge from $(a,b,c)$ to each of the following vertices (except those already equal to $(a,b,c)$):
     - $(0,b,c)$ and $(a,0,c)$ and $(a,b,0)$ — dumping a jar into the lake
     - $(A,b,c)$ and $(a,B,c)$ and $(a,b,C)$ — filling a jar from the lake
     - $\begin{Bmatrix} (0, a+b, c) & \text{if } a+b \le B \\ (a+b-B, B, c) & \text{if } a+b \ge B \end{Bmatrix}$ — pouring from jar 1 into jar 2
     - $\begin{Bmatrix} (0, b, a+c) & \text{if } a+c \le C \\ (a+c-C, b, C) & \text{if } a+c \ge C \end{Bmatrix}$ — pouring from jar 1 into jar 3

- $\left\{\begin{array}{ll}(a+b,0,c) & \text{if } a+b \le A \\ (A, a+b-A, c) & \text{if } a+b \ge A\end{array}\right\}$ — pouring from jar 2 into jar 1
- $\left\{\begin{array}{ll}(a,0,b+c) & \text{if } b+c \le C \\ (a, b+c-C, C) & \text{if } b+c \ge C\end{array}\right\}$ — pouring from jar 2 into jar 3
- $\left\{\begin{array}{ll}(a+c,b,0) & \text{if } a+c \le A \\ (A, b, a+c-A) & \text{if } a+c \ge A\end{array}\right\}$ — pouring from jar 3 into Jar 1
- $\left\{\begin{array}{ll}(a,b+c,0) & \text{if } b+c \le B \\ (a, B, b+c-B) & \text{if } b+c \ge B\end{array}\right\}$ — pouring from jar 3 into jar 2

Since each vertex has at most 12 outgoing edges, there are at most $12(A+1) \times (B+1)(C+1) = O(ABC)$ edges altogether.

To solve the jars problem, we need to find the ***shortest path*** in $G$ from the start vertex $(0,0,0)$ to any target vertex of the form $(k,\cdot,\cdot)$ or $(\cdot,k,\cdot)$ or $(\cdot,\cdot,k)$. We can compute this shortest path by calling ***breadth-first search*** starting at $(0,0,0)$, and then examining every target vertex by brute force. If BFS does not visit any target vertex, we report that no legal sequence of moves exists. Otherwise, we find the target vertex closest to $(0,0,0)$ and trace its parent pointers back to $(0,0,0)$ to determine the shortest sequence of moves. The resulting algorithm runs in $O(V+E) = \boldsymbol{O(ABC)}$ ***time***.

We can make this algorithm faster by observing that every move either leaves at least one jar empty or leaves at least one jar full. Thus, we only need vertices $(a,b,c)$ where either $a=0$ or $b=0$ or $c=0$ or $a=A$ or $b=B$ or $c=C$; no other vertices are reachable from $(0,0,0)$. The number of non-redundant vertices and edges is $O(AB+BC+AC)$. Thus, if we only construct and search the relevant portion of $G$, the algorithm runs in $\boldsymbol{O(AB+BC+AC)}$ ***time***. ■

**Rubric:** 10 points: standard graph reduction rubric (see next page)

- Brute force construction is fine.

— 1 for calling Dijkstra instead of BFS

- max 8 points for $O(ABC)$ time; scale partial credit.

**Standard rubric for graph reduction problems.** For problems out of 10 points:

- + 1 for correct vertices, *including English explanation for each vertex*
- + 1 for correct edges
  - −½ for forgetting "directed" if the graph is directed
- + 1 for stating the correct problem (in this case, "shortest path")
  - − "Breadth-first search" is not a problem; it's an algorithm!
- + 1 for correctly applying the correct algorithm (in this case, "breadth-first search from $(0,0,0)$ and then examine every target vertex")
- + 1 for time analysis in terms of the input parameters.
- + 5 for other details of the reduction
  - – If your graph is constructed by naive brute force, you do not need to describe the construction algorithm; in this case, points for vertices, edges, problem, algorithm, and running time are all doubled.
  - – Otherwise, apply the appropriate rubric, *including Deadly Sins*, to the construction algorithm. For example, for a solution that uses dynamic programming to build the graph quickly, apply the standard dynamic programming rubric.