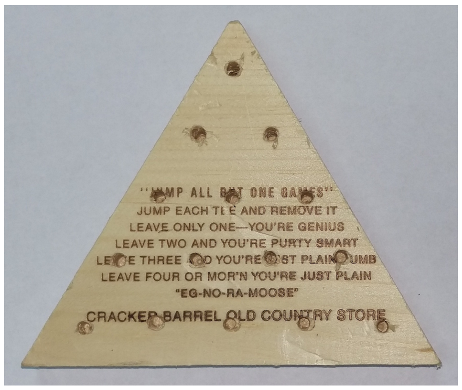
Final Project: Cracker Barrel Peg Game

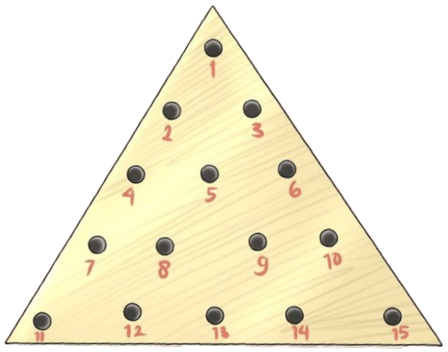
You must implement an algorithm to solve Peg Solitaire solely from an image of the game!

Introduction:

This project builds on the concepts learned thus far to solve the classic puzzle game of Peg Solitaire. The game consists of a board and pegs, and the objective is to remove all pegs but one by jumping various pegs in a particular order. There are actually numerous variations to the game, but we'll be focusing on the 15 hole triangular board shown below. The player is only allowed to jump an adjacent peg, moving to an empty location.


"JUMP ALL BUT ONE GAMES"
JUMP EACH TEE AND REMOVE IT
LEAVE ONLY ONE—YOU'RE GENIUS
LEAVE TWO AND YOU'RE PURTY SMART
LEAVE THREE AND YOU'RE JUST PLAIN DUMB
LEAVE FOUR OR MOR'N YOU'RE JUST PLAIN
"EG-NO-RA-MOOSE"
CRACKER BARREL OLD COUNTRY STORE

(a)



(b)

Figure 1: Peg game and typical numbering scheme.

The trick of the game is to plan out the best order to perform jumps so as only to leave one peg. It is easy to make a mistake that leaves 2, 3, or even more pegs on the board without another move possible!

Your goal is to solve the 15-hole Peg game from only its image! The image can be any size or scale but the board and pegs will always be fully visible. Your code must detect the initial location of the pegs and then generate all possible solutions using the provided algorithm (there are numerous solution methods but everyone must use the one detailed in this document!)

Part 1: Image Processing

Given various images of the game, your code must determine where the tees are located. This is a perfect image thresholding problem! The images you have to process will always follow a few rules:

- the board will always be fully visible and oriented upwards
- all pegs are blue
- all images have a white background
- images may be different sizes

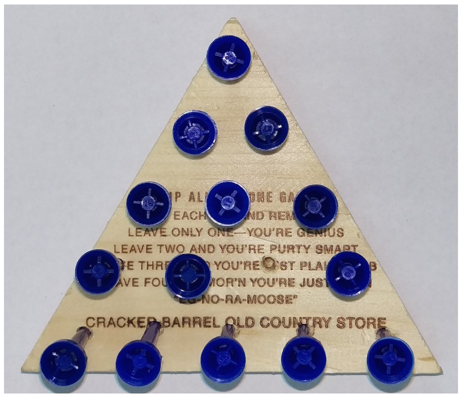

LEAVE ONLY ONE—YOU'RE GENIUS
LEAVE TWO AND YOU'RE PURTY SMART
NO-RA-MOOSE"
CRACKER BARREL OLD COUNTRY STORE

Figure 2: Example image to decode in software.

You must create a function called **AnalyzePegImage** that has an input for the name of an image and returns a vector representing peg positions. The vector will always be 15 numbers, where 0 represents an empty hole and 1 represents a peg. The order of the values must follow the numbering scheme shown in Figure 1. For example, Figure 2 corresponds to the vector:

1 1 1 1 1 1 1 1 0 1 1 1 1 1 1

Part 2: Solution Algorithm

To find every possible solution for a particular game, create a **PegAlgorithm** function which will use the **breadth-first** search to explore different jumping options. The difficult part here is that we actually need to track how we got to a solution, not just that a solution exists! We'll need to use an **explore list** to track the different jumping possibilities, but we also need to track what jump got us to this one. This is accomplished using a **parent id**; each move will be given the id of its parent move, which is just the index in explore list of the move that preceded this one.

The first step is to find the possible moves for a given board setup. Remember that there are only a finite amount of moves you can make, so you may simply want to write out the different possibilities. For each possible move you find, your code must remember the following pieces of information:

- current move
- parent id
- state of the board after the move

Repeat this process until there are no more possible moves!

Finally, go back through explore list looking for moves that resulted in only 1 peg left on the board. If you follow the parents of this move, you will eventually get back to the original game, showing you the path of how to win. **Please check out the class video from 11/24 to get a detailed example of this!**

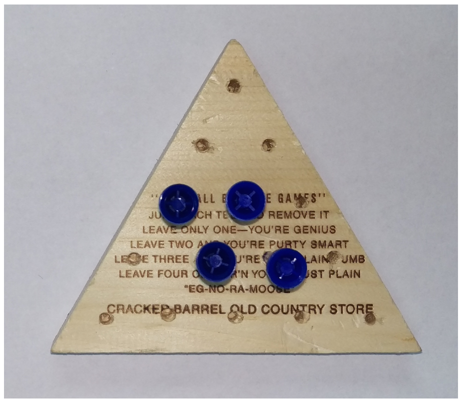
Key steps to solution:

Since this is a final project, you are mostly free to implement the solution in any way you like, but there are some key points that everyone must follow. **First, your code must be well commented!** Since you are not turning in a written report with the code, comments are the best way to explain various portions of your code. Second, you must create **at least 2 functions: AnalyzePegImage and PegAlgorithm**. **Lastly, you may not use any pre-built/external code to automatically solve the problem!** If you find a new command you would like to use but are unsure if it is allowed, just ask!

- AnalyzePegImage function
  - Summary: Given the name of an image, determine the positions of the pegs
  - Inputs: imageName (string)
  - Outputs: pegs (1D vector)
- PegAlgorithm function
  - Summary: Given a vector of peg positions, return a 2D vector of all possible solutions (only 1 peg left!)
  - Inputs: pegs (1D vector)
  - Outputs: solutions (2D matrix)

Sample Output:

Image Processing Examples:


"ALL E GAMES"
JU CH TE REMOVE IT
LEAVE ONLY ONE—YOU'RE GENIUS
LEAVE TWO A YOU'RE PURTY SMART
LE E THREE 'RE LAIN UMB
LEAVE FOUR 'R'N YO ST PLAIN
"EG-NO-RA-MOOSE"
CRACKER BARREL OLD COUNTRY STORE
(a) 000110011000000

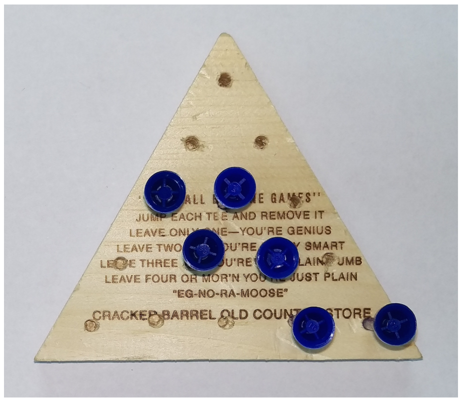

"ALL E GAMES"
JUMP EACH TEE AND REMOVE IT
LEAVE ONLY NE—YOU'RE GENIUS
LEAVE TWO U'RE SMART
LE E THREE U'RE LAIN UMB
LEAVE FOUR OR MOR'N YOU JUST PLAIN
"EG-NO-RA-MOOSE"
CRACKER BARREL OLD COUN STORE
(b) 000110011000011

Figure 3: Example images and corresponding pegs vector. More test images are provided in Canvas.

Solution Algorithm Examples:

```
Given board layout: {0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0}
2 solutions found!
Solution 1: 9 2
Solution 2: 5 14

Given board layout: {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0}
2 solutions found!
Solution 1: 14 5 8 3
Solution 2: 14 5 5 12

Given board layout: {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0}
1 solution found!
Solution 1: 14 5 5 12 11 13

Given board layout: {0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0}
11 solutions found!
Solution 1: 14 5 7 9 9 2 4 1
Solution 2: 14 5 7 9 9 2 2 7
Solution 3: 14 5 7 9 4 6 6 13
Solution 4: 14 5 7 2 2 9 9 7
Solution 5: 14 5 7 2 2 9 8 10
Solution 6: 14 5 5 12 4 11 11 13
Solution 7: 14 5 4 11 5 12 11 13
Solution 8: 14 5 4 6 7 9 6 13
Solution 9: 7 2 14 5 2 9 9 7
Solution 10: 7 2 14 5 2 9 8 10
Solution 11: 4 11 14 5 5 12 11 13

Given board layout: {0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0}
2 solutions found!
Solution 1: 6 15 15 13 13 11 11 4 4 1
Solution 2: 6 15 15 13 13 11 11 4 2 7

Given board layout: {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
29760 solutions found!
....

Given board layout: {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
1550 solutions found!
....
```