VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
COMPUTER SCIENCE

EXERCISE REPORT
COMPLETE SEARCH - BACKTRACKING

Lecturer:	Son Nguyen Thanh McS
Class:	CS112.N21.KHTN
Members:	Hoang Ha Van - 21520033
	Anh Vo Thi Phuong - 21522883
Date:	08/05/2023

Contents

1. Exercise 1

1.1. Problem:

The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other. Given an integer n, return all distinct solutions to the n-queens puzzle. You may return the answer in any order. Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' Both indicate a queen and an empty space, respectively.

1.2. Solution:

The queens can attack each other if they are in the same row, column or diagonal. So we assign each queen to a row (queen i belongs to row i - 1), then choose column for each queen.

The backtrack function starts with first queen, we iterate through n columns and check if current position valid, then we place first queen there and call backtrack function on second queen and so on.

Algorithm:

1. Start the backtracking process with the initial row as 0.

2. If the current row is equal to n, it means all queens have been successfully placed on the board. Add the current board configuration to the solutions list and return.

3. Iterate through each column in the current row.

4. Check if placing a queen at the current position is valid. (A position is valid if no other queens can attack it: Check if there is a queen in the same column or on the same diagonal.)

5. If the position is valid, place a queen at the current position on the board, then recursively call the backtracking function with the next row (row + 1). After

returning from the recursive call, remove that queen from the current position.

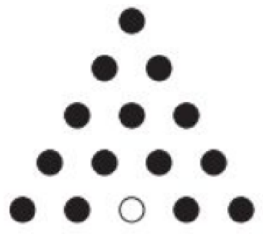
Implementation: Here

2. Exercise 2

2.1. Problem:

Puzzle Pegs:

This puzzle-like game is played on a board with 15 small holes arranged in an equilateral triangle. In an initial position, all but one of the holes are occupied by pegs, as in the example shown below. A legal move is a jump of a peg over its immediate neighbor into an empty square opposite; the jump removes the jumped-over neighbor from the board.



Design and implement a backtracking algorithm for solving the following versions of this puzzle.

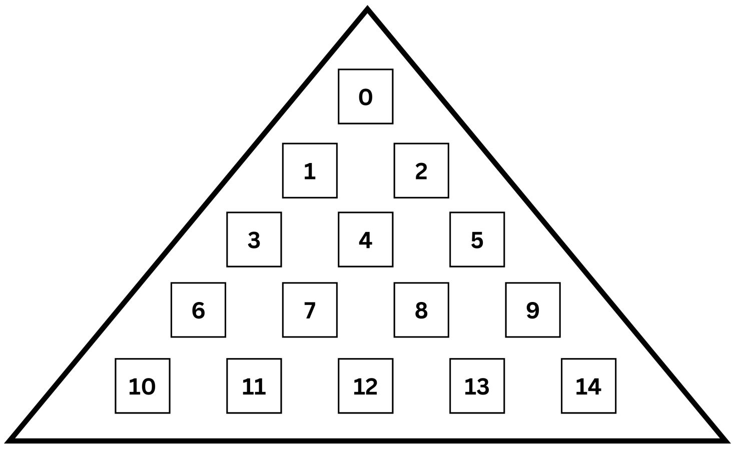
a. Starting with a given location of the empty hole, find a shortest sequence of moves that eliminates 14 pegs with no limitations on the final position of the remaining peg.

b. Starting with a given location of the empty hole, find a shortest sequence of moves that eliminates 14 pegs with the remaining peg at the empty hole of the initial board.

2.2. Solution:

Notice that if a solution exists, it has exactly 13 legal moves. Simply because for each legal move it removes exactly 1 peg from the table. Initially there were 14 pegs on the board, so after exactly 13 legal moves there was only 1 peg left. The problem is thus transformed into the problem of finding a satisfying solution.

You can create a table like below and store it as a bitmask:



a. To find a sequence of moves that eliminates 14 pegs with no limitations on the final position of the remaining peg, we can use the following backtracking algorithm:

1. If there is only one peg on the board, print the solution and return True

2. If not, iterate through each hole which is occupied by peg.

3. Check if it can make a legal move by jumping over its immediate neighbor into an empty square opposite.

4. If a legal move is possible, add this move to sequence of moves, create a copy of the current board and update it with this move, then recursively call the backtracking function to next move with the updated board.

5. If the recursive call return True, return True.

6. If not, remove this move from the sequence of moves, restore previous board configuration and try a different move.

7. If no solution is found after exploring all possible moves, return False.

You can see our implementation here: Link

b. To find a sequence of moves that eliminates 14 pegs with the remaining peg at the empty hole of the initial board, we can use the following backtracking algorithm:

1. If there is only one peg on the board, check if it occupies the empty hole of the initial board. If it occupies the empty hole of the initial board, print the solution and return True, else return False.

2. If there is more than one peg on the board, iterate through each cell which is occupied by peg.

3. Check if it can make a legal move by jumping over its immediate neighbor into an empty square opposite.

4. If a legal move is possible, add this move to sequence of moves, create a copy of the current board and update it with this move, then recursively call the backtracking function to next move with the updated board.

5. If the recursive call return True, return True.

6. If not, remove this move from the sequence of moves, restore previous board configuration and try a different move.

7. If no solution is found after exploring all possible moves, return False.

You can see our implementation here: Link