Assignment 07: Backtracking and Helping Your Favorite Aunt

Due: Wed Apr 10, 2024 11:59pm

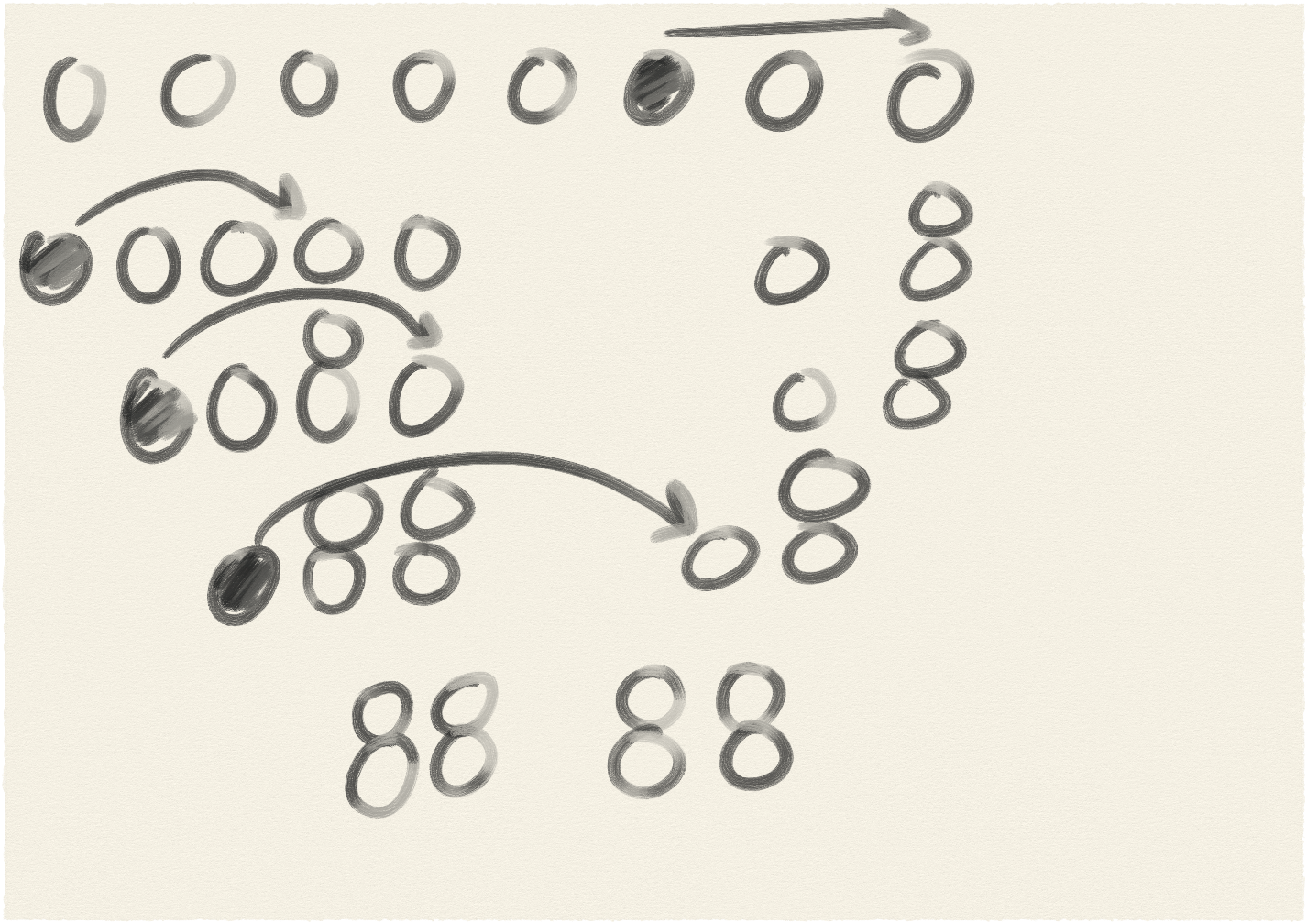
Kaylee Lewis

1. (40 points) There are n coins placed in a row. The goal is to form n/2 pairs of them by a sequence of moves. On each move a single coin can jump right or left over two coins adjacent to it (i.e., over either two single coins or one previously formed pair) to land on the next single coin; no triples are allowed. Any empty space between adjacent coins is ignored. Determine all the values of n for which the puzzle has a solution and devise an algorithm that solves the puzzle in the minimum number of moves for any even value of n.

You may use backtracking to find the minimum number of coins for which the problem has a solution. Note that the problem statement automatically rejects any odd value of n.

A white screen with black text

Description automatically generated



// Assignment 07: Backtracking and Helping Your Favorite Aunt

// Due : Wed Apr 10 , 2024 11 : 59pm

// Kaylee Lewis

// Problem 1: There are n coins placed in a row. The goal is to form n/2 pairs of them by a sequence of

// moves On each move a single coin can jump right or left over two coins adjacent to it ( i.e. , over either

// two single coins or one previously formed pair ) to land on the next single coin; no triples are allowed.

// Any empty space between adjacent coins is ignored.Determine all the values of n for which the puzzle

// has a solution and devise an algorithm that solves the puzzle in the minimum number of moves for

// any even value of n.

// You may use backtracking to find the minimum number of coins for which the problem has a solution.

// Note that the problem statement automatically rejects any odd value of n.

//

// www.geeksforgeeks.org/coin-puzzle-find-minimum-number-of-coins

// Algorithmic Puzzles By Anany Levitin , Maria Levitin

// https://www.youtube.com/watch?v=NNcN5X1wsaw

// https://www.youtube.com/watch?v=hhVNvi\_M-dA

#include <iostream>

#include <vector>

// Function to solve the puzzle

std::vector<int> solvePuzzle ( int n , std::vector<std::vector<int>> &moves )

{

// Check if n is a multiple of 4

if ( n % 4 != 0 )

{

// If not, print a message indicating no solution exists

std::cout << "No solution exists for n = " << n << ". Please provide an even number that is a multiple of 4." << std::endl;

return {};

}

// Initialize a vector representing the initial state of the puzzle

std::vector<int> puzzle;

for ( int i = 1; i <= n; ++i )

{

puzzle.push\_back ( i );

}

// Store the original arrangement of coins

moves.push\_back ( puzzle );

// Perform the first set of operations

for ( int i = 1; i < n / 4; ++i )

{

// Find the rightmost single coin

int rightmostSingle = puzzle.back ( );

puzzle.pop\_back ( );

// Find the coin to its left with i coins between them

int targetIndex = puzzle.size ( ) - i;

puzzle.insert ( puzzle.begin ( ) + targetIndex , rightmostSingle );

// Store the current arrangement of coins after the move

moves.push\_back ( puzzle );

}

// Perform the second set of operations

for ( int i = n / 4; i < n / 2; ++i )

{

// Find the leftmost single coin

int leftmostSingle = puzzle.front ( );

puzzle.erase ( puzzle.begin ( ) );

// Jump it over i coins to the right

int targetIndex = n / 2 + i - 1;

puzzle.insert ( puzzle.begin ( ) + targetIndex , leftmostSingle );

// Store the current arrangement of coins after the move

moves.push\_back ( puzzle );

}

return puzzle;

}

int main ( )

{

int n = 8;

std::vector<std::vector<int>> moves;

// Solve the puzzle for the given value of n

std::vector<int> solution = solvePuzzle ( n , moves );

if ( !solution.empty ( ) )

{

// If a solution exists, print the original arrangement of coins

std::cout << "Original arrangement of coins: ";

for ( int coin : moves.front ( ) )

{

std::cout << coin << " ";

}

std::cout << std::endl;

// Print each move made during the puzzle solving process

for ( size\_t i = 1; i < moves.size ( ); ++i )

{

std::cout << "Move " << i << ": ";

for ( int coin : moves [ i - 1 ] )

{

std::cout << coin << " ";

}

std::cout << " -> ";

for ( int coin : moves [ i ] )

{

std::cout << coin << " ";

}

std::cout << std::endl;

}

// Print the final arrangement of coins

std::cout << "Final arrangement of coins: ";

for ( int coin : solution )

{

std::cout << coin << " ";

}

std::cout << std::endl;

}

return 0;

}

2. (60 points) Sudoku puzzles are a logic-based combinatorial number-placement puzzle. The objective is fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub-grids that compose the puzzle contains all of the digits from 1 to 9. Completed puzzles must meet additional constraints on contents of individual regions. For example, the same integer cannot appear twice in the same row, column, or in any of nine 3x3 squares in the grid. The author of the puzzle provides a partially completed grid, which assuming a well-posed puzzle, has a unique solution.

Implement a Sudoku solver in your favorite programming language using the backtracking algorithm presented in class. Puzzles should be read from a text file that has one puzzle row per line in the file. Use a value of 0 to indicate an empty entry in the puzzle grid. Solutions should be output to standard output and written to a text file.

NOTE : I am aware that one can find a solution to this program through judicious application of Google-Fu. Remember: quote sources. And critically review any site you use for assistance as there are many WRONG solutions to this problem available on the Internet.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

// Assignment 07: Backtracking and Helping Your Favorite Aunt

// Due : Wed Apr 10 , 2024 11 : 59pm

// Kaylee Lewis

// Problem 2: Sudoku puzzles are a logic - based combinatorial number - placement puzzle.

// The objective is fill a 9x9 grid with digits so that each column , each row , and each of the nine 3x3

// sub - grids that compose the puzzle contains all of the digits from 1 to 9.

// Completed puzzles must meet additional constraints on contents of individual regions.

// For example , the same integer cannot appear twice in the same row , column , or in any of nine 3x3

// squares in the grid.

// The author of the puzzle provides a partially completed grid , which assuming a well - posed puzzle ,

// has a unique solution.

// Implement a Sudoku solver in your favorite programming language using the backtracking algorithm

// presented in class.

// Puzzles should be read from a text file that has one puzzle row per line in the file.

// Use a value of 0 to indicate an empty entry in the puzzle grid.

// Solutions should be output to standard output and written to a text file.

// NOTE : I am aware that one can find a solution to this program through judicious application of Google -

// Fu.Remember : quote sources. And critically review any site you use for assistance as there are many

// WRONG solutions to this problem available on the Internet

// https://www.geeksforgeeks.org/sudoku-backtracking-7/

// https://en.wikipedia.org/wiki/Sudoku\_solving\_algorithms

// https://medium.com/analytics-vidhya/sudoku-backtracking-algorithm-and-visualization-75adec8e860c

#include <iostream>

#include <fstream>

#include <vector>

const int N = 9;

// Function to print the Sudoku grid

void printGrid ( std::vector<std::vector<int> > &grid )

{

for ( int i = 0; i < N; i++ )

{

for ( int j = 0; j < N; j++ )

{

std::cout << grid [ i ] [ j ] << " ";

}

std::cout << std::endl;

}

}

// Function to check if a number can be placed in the given position

bool isSafe ( std::vector<std::vector<int> > &grid , int row , int col , int num )

{

// Check if the number is not already present in the current row

for ( int i = 0; i < N; i++ )

{

if ( grid [ row ] [ i ] == num )

{

return false;

}

}

// Check if the number is not already present in the current column

for ( int i = 0; i < N; i++ )

{

if ( grid [ i ] [ col ] == num )

{

return false;

}

}

// Check if the number is not already present in the current 3x3 subgrid

int startRow = row - row % 3;

int startCol = col - col % 3;

for ( int i = 0; i < 3; i++ )

{

for ( int j = 0; j < 3; j++ )

{

if ( grid [ i + startRow ] [ j + startCol ] == num )

{

return false;

}

}

}

return true;

}

// Function to solve the Sudoku puzzle using backtracking

bool solveSudoku ( std::vector<std::vector<int> > &grid )

{

int row , col;

bool isEmpty = false;

for ( row = 0; row < N; row++ )

{

for ( col = 0; col < N; col++ )

{

if ( grid [ row ] [ col ] == 0 )

{

isEmpty = true;

break;

}

}

if ( isEmpty )

{

break;

}

}

if ( !isEmpty )

{

// All cells are filled, puzzle solved

return true;

}

// Try placing numbers from 1 to 9

for ( int num = 1; num <= 9; num++ )

{

if ( isSafe ( grid , row , col , num ) )

{

grid [ row ] [ col ] = num;

// Recursively solve the remaining puzzle

if ( solveSudoku ( grid ) )

{

return true;

}

// If placing num at (row, col) doesn't lead to a solution, backtrack

grid [ row ] [ col ] = 0;

}

}

// No solution found

return false;

}

int main ( )

{

// Open the input file

std::ifstream inputFile ( "input.txt" );

if ( !inputFile )

{

std::cerr << "Error: Unable to open input file." << std::endl;

return 1;

}

// Read the puzzle from the input file

std::vector<std::vector<int> > grid ( N , std::vector<int> ( N ) );

for ( int i = 0; i < N; i++ )

{

for ( int j = 0; j < N; j++ )

{

inputFile >> grid [ i ] [ j ];

}

}

inputFile.close ( );

// Solve the Sudoku puzzle

if ( solveSudoku ( grid ) )

{

// Open the output file

std::ofstream outputFile ( "output.txt" );

if ( !outputFile )

{

std::cerr << "Error: Unable to open output file." << std::endl;

return 1;

}

// Write the solution to the output file

for ( int i = 0; i < N; i++ )

{

for ( int j = 0; j < N; j++ )

{

outputFile << grid [ i ] [ j ] << " ";

}

outputFile << std::endl;

}

outputFile.close ( );

// Print the solution to standard output

printGrid ( grid );

} else

{

std::cout << "No solution exists." << std::endl;

}

return 0;

}