COMP3010 Algorithm Theory

Assignment 1

Chaz Lambrechtsen

45426317

Table of contents

Structure

class Grid(int width, int height)

- **Cell get(int x, int y)** – return cell at given x and y coordinates on grid
- **Boolean isInBounds(int x, int y)** – return true for x and y values that are inside grid and false for coordinates out of bounds
- **Boolean placeRect(Rect rect, Cell base)** – place given rect on grid return true when placed successfully, return false when unsuccessful
- **ArrayList<Cell> getRectCells(Rect rect)** – return cells contained inside a rect on grid
- **ArrayList<Rect> fitRects(Cell base, Rect rect)** – return list of possible fitted rects for the given base cell.
- **String[][] toStringArray()** – Convert grid to String[][] array

class Cell(int x, int y, int value, String id, Boolean isBase)

- **Boolean isAvailable()** – return true when cell value is zero otherwise false
- **Replace(Cell c)** – replace cell id with input cell id

class Rect(Cell topleft, int width, int height, String id)

- **print()** – print rect values

Other methods and local variables

- **ArrayList<Rect> generateRects(Cell topleft, String id, int area)** – generate possible basic rectangles using factor pairs of area, where top left cell is base cell
- **Grid DFS(int base, ArrayList<Rect> rects, Grid grid)** – Recursive method used to find final solution using bruteforce DFS method to mark pathways that have already been tested. Returns final grid once base > cellBases.size()
- **solve[][] (String infile)** – returns final solution
- **ArrayList<Cell> cellBases** – contains list of all cell bases and their ids/values

Program Flow

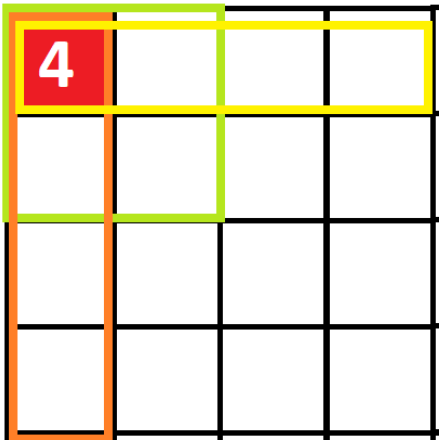
The solve() method is called after reading the input data of the grid.

Then basic rectangles are generated for each base cell in the grid using the generateRects() method. Basic rectangles are generated by finding factor pairs of the area. This will output basic width and height rectangles.

DFS() is then called and utilises the fitRects() method to find possible rectangle that fit in a given space on the grid for each basic rectangle. If fitRects returns rectangles that do not fit, they are marked as invalid and DFS is called back on base+1. The solution is found when base > cellBases.size() (total amount of bases in grid).

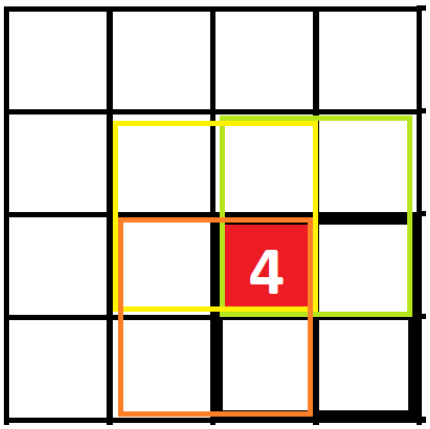
Main methods:

Basic Rectangles: generateRects()



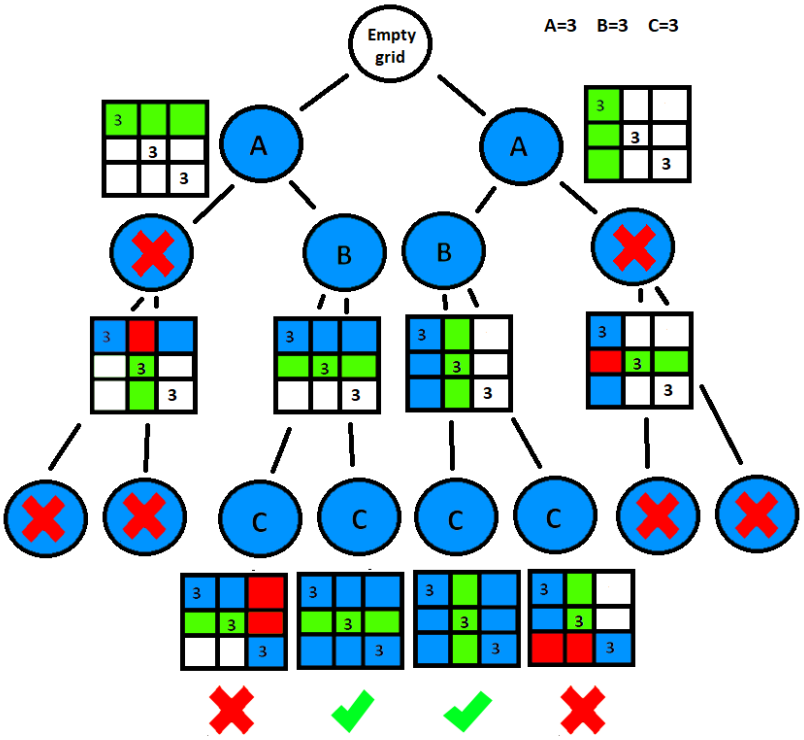
For each base cell, Basic rectangles will be generated using the generateRects() method. First, this method will find factors of the area which are the width and height of rectangles. For example, factors of area 4 can be 2x2 or 4x1. These factors are also swapped to generate rotated rectangles excluding identical pairs e.g. 4x4. This will then generate the last factor pair, 1x4.

Fitted rectangles: fitRects()



For each of the previously generated basic rectangles, the fitted rectangles are generated in the method fitRects() by decrementing the x and y axis of the top left cell. For example, x-1 then y-1 and x-1 y-1. The rectangle will not be created if the top left is out of bounds or a cell that is occupied.

DFS implementation: DFS()



There are 2 solutions, but this is just an example case.

In this DFS, both possibilities for A are basic rectangles generated by the generateRects() method. Both rectangles in B are fitted rectangles generated by fitRects, because the base is not the top left. The solution is found once all bases have been covered and all different combinations are tried, C is the final base.

As shown in this diagram the DFS brute force is a process of elimination

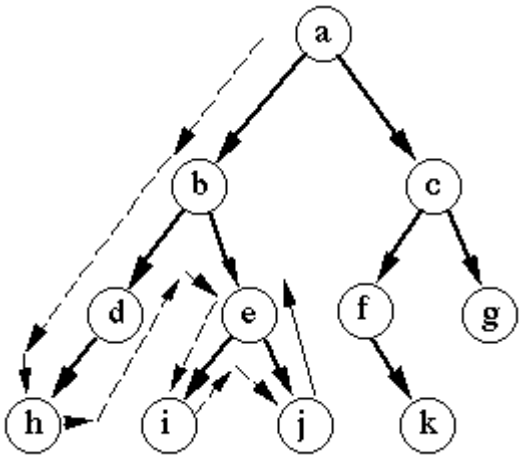
Algorithm explanation

Overview

The solution used is an implementation of the depth first search algorithm. This algorithm will traverse a data structure by starting at the root node and explore as far as possible along each branch before backtracking. Similarly, in my solution graphs will be filled with rectangles with a base at each iteration starting with an empty graph (root node) before backtracking.

Example

[1] fig 3.1: Depth first search



As shown in this example the depth first search algorithm will search the tree from the left branch and backtrack after the end of each branch.

References:

[1] S Even, Depth-First Search. In G. Even (Ed.), Graph Algorithms (pp. 46-64). Cambridge: Cambridge University Press. 2011. doi:10.1017/CBO9781139015165.006