Depth First Search Algorithm for Solving Nonogram Puzzles

Michael Ray/13517092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517092@std.stei.itb.ac.id

***Abstract*— Nonogram Puzzle is a logical picture puzzle played on a grid with numerical hints on the top and the left side of the grid. The hints are used to know which cell of the grid is supposed to be colored. In this paper, I will try to explain the use of Depth First Search (DFS) Algorithm to solve a Nonogram Puzzle.**

Keywords—Nonogram Puzzle; Depth First Search (DSF)

I. Introduction

Who doesn't love playing good old puzzles? Many puzzles that we know now are most likely to be a derivation of the ones older and way more simpler. One of those simple puzzles is the Nonogram Puzzle. It's a simple logical grid colouring puzzle that uses numbers as a guide to colour the grid. This somehow simple puzzle can be intimidating sometimes because this puzzle doesn't have size constraint.

Nonogram Puzzle first came about at 1897, when a Japanese graphics editor won a competition at Tokyo by designing a grid picture using a skyscrapers lights that were turned on or off. This led her to the idea of making a puzzle based on filling certain cell on a grid using numbers as hints.

The gameplay is very simple, there are going to be numbers at the left side of each row and the top each column. Using those numbers as a guide, start choosing which cell should be coloured and which cell should be left empty. For example, if the number on the left side of a row is "2 4", this means that there should be 2 consecutive cells and 4 other consecutive cells that should be coloured and at least one cell is empty between the two. Nonogram Puzzles are considered as NP-Complete problems, this means that this problem cannot be completed in polynomial time.

This puzzle is interesting in many ways, I myself have found this puzzle to be very interesting, and that is why I chose this as my paper topic.

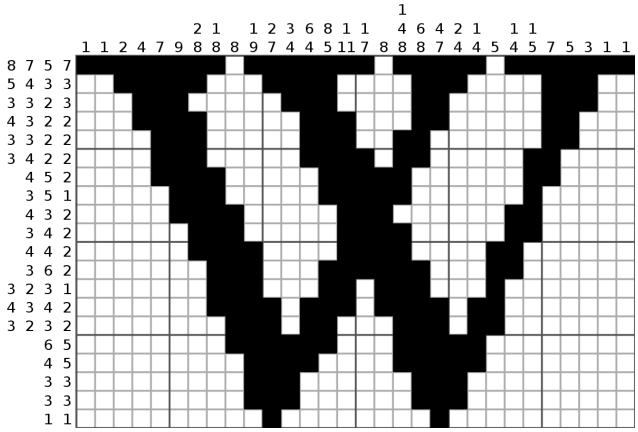


Figure 1 - Example of a nonogram puzzle
Source: https://en.wikipedia.org/wiki/Nonogram

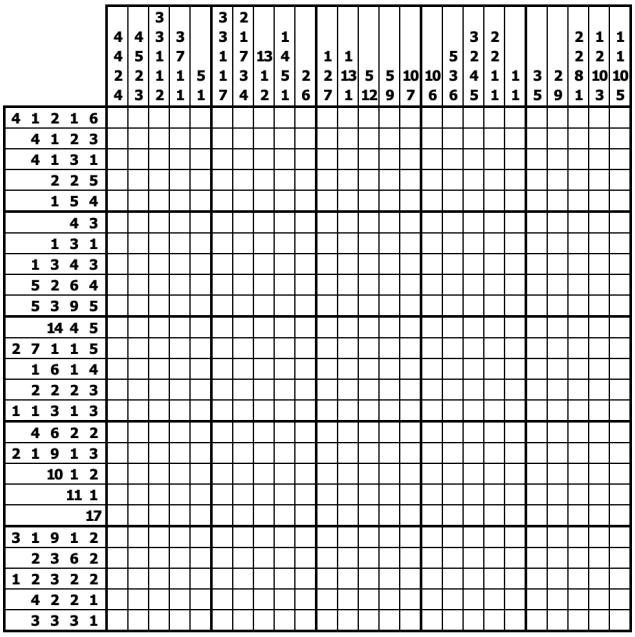


Figure 2 - Example of a 25x25 nonogram puzzle
Source: https://www.puzzle-nonograms.com/

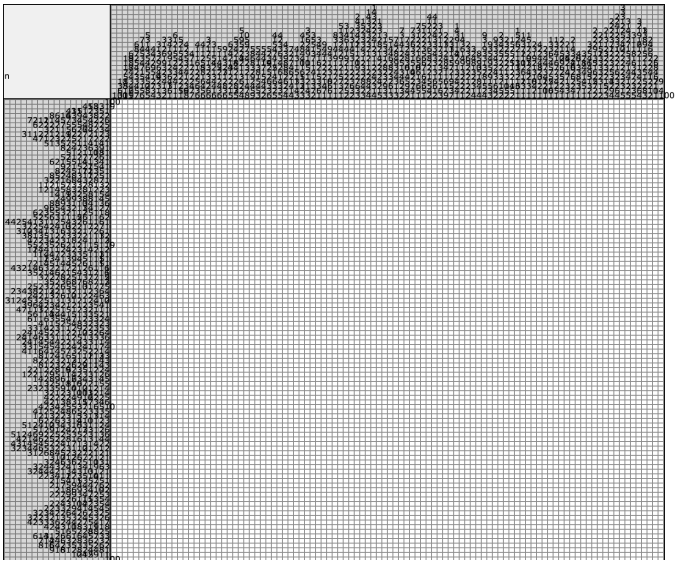


Figure 3 - Example of a 100 by 100 nonogram puzzle

II. Base Theory

A. Depth First Search (DFS)

Depth First Search is an algorithm mostly used to traverse or search along a tree or graph data structure. It starts at a particular node, usually called the root node, and explores a branch as far/deep as possible before going to another branch.

For example, on a graph

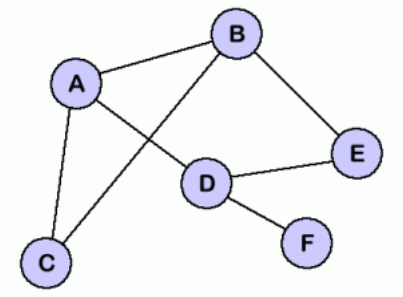


Figure 4 - A graph

If we use a DFS algorithm to traverse all the nodes on this graph with A as the root node, the function will get A-B-E-D-F then backtracks all the way to the B node then continues the search to C, because the function will search as deep as possible through a branch until there is no new nodes to search.

Using DFS as a solving algorithm means to make a tree data structure with the root node representing the starting condition, other nodes representing steps that are taken, and leaf nodes representing possible solutions.

DFS algorithm is considered as a recursive algorithm which uses backtracking.

B. NP-Complete Problems

NP-Complete problems or most commonly known as NPC problems are NP (Non-deterministic Polynomial) problems that are usually interesting and hard to solve in polynomial time.

This is how to prove that a problem is an NPC problem, first find an NP problem that is similar to the problem that you want to prove to be NPC, the find a way to reduce that NP problem to the problem you have in polynomial time.

The reason why can't we solve an NPC problem in polynomial time and considered as the hardest problems is because if somehow someone solved an NPC problem in polynomial time, it means that all NP problems supposedly could be solved in polynomial time.

C. Recursive

Recursive function is a function which call itself inside the function. On recursive functions, there are 2 important parts:

1. The base case
2. The recursive part

In the base case, the function is stopped at some specific point where the problem doesn't need to be traversed anymore. In the recursive part, the program does all the necessary work and reduces the problem so that it will reach the base case condition.

An example of recursive function is the Fibonacci sequence, here the base are fib(0) = 0 and fib(1) = 1, and the recursive part is fib(n) = fib(n-1) + fib(n-2). Here, the recursive part is slowly going to reach the base because the variable n will be decreased by 1 and 2 each time, which means it will reach the number 1 and 0 at some point.

D. Backtracking

Backtracking is a process used in DFS algorithm to track back the process to look for other possibilities. This process is also used by recursive functions as a way to ensure the function find every possible outcome.

III. Using DFS to Solve Nonogram Puzzle

My method of trying to solve the Nonogram Puzzle can be narrowed down to 3 steps, try to solve a row of the grid then check if that finished row complies to the column hints, if it is fulfilled then continue to the next row else find another possible solution for the current row, if the program can't find another solution for that row backtrack to the previous row. For this paper I will use an example grid like so:

	1		
	1	5	1
1			
2			
1			
1			
3			

Figure 5

With each row there is a hint on the left side. Using that hint try to fill the row from the leftmost cell.

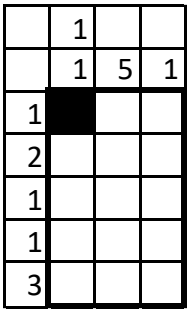


Figure 6

Then check the column hint, if it is fulfilled continue to the next row, here the column hint is fulfilled.

On the next row, fill the cells from the leftmost cell using the hint.

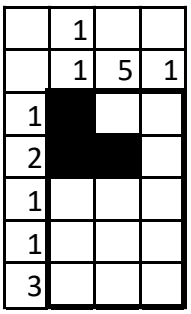


Figure 7

Here as you can see, the column hints are no longer fulfilled, so the program finds another solution for the row. For checking the column hints, the program will only check if there is some cell that is more than the constraint, in this example the first column has a group of 2 cells together, while at the column hint says there should only be 1 cell with another cell separated.

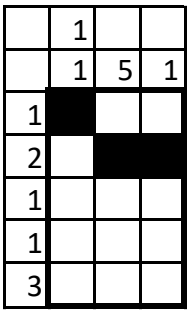


Figure 8

Here the constraints are fulfilled, so the program continues to the next row.

On the next row the program will fill the cells again from the leftmost cell using the hints. I will skip the process to the second last row because the column hints will not be fulfilled at that row.

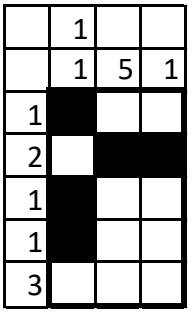


Figure 9

Here as you can see, the left column hint is not fulfilled, so the program backtracks to the previous row and finds an alternative by shifting the first group of cells to the right, then continues to the next row.

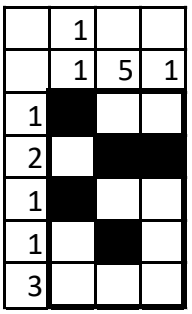


Figure 10

Here, the first and the third column hints are both violated, so the program will backtrack to the previous row and finds another alternative, then continues to the current row.

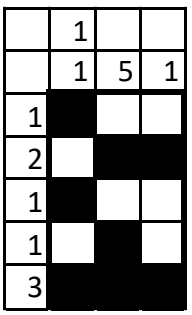


Figure 11

Here once again the column hints are violated as you can see on the first and third column. Here the program backtracks

finds that it is still violated, then backtracks again, and this process is repeated until the program reaches the first row.

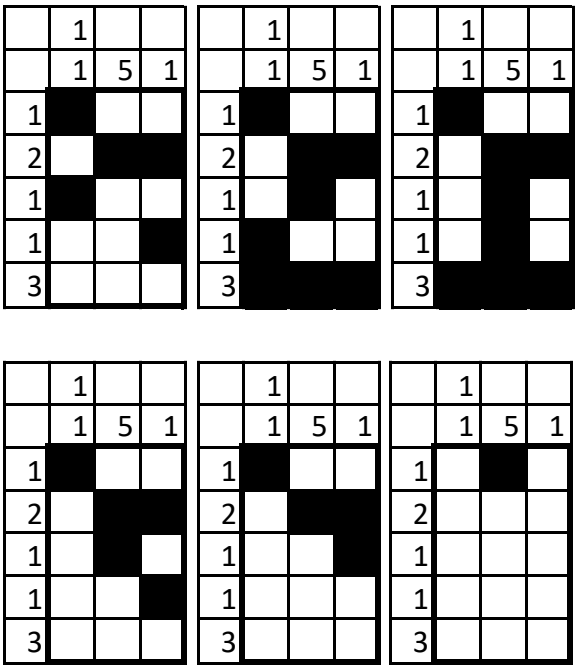


Figure 12

Here finally the program continues forward. The program will fill the cell all over again from the second row.

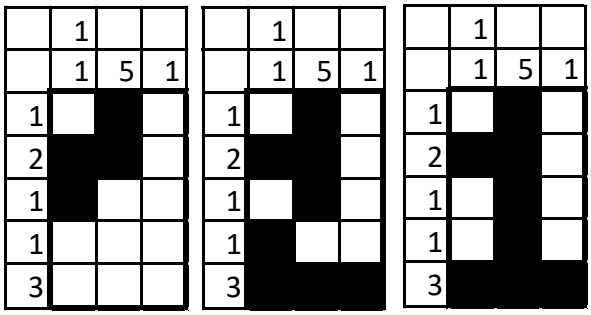


Figure 13

For this algorithm, the complexity that I have calculated is $T(n) = O(n^2 \log(n))$

The tree structure that is generated should be something like this:

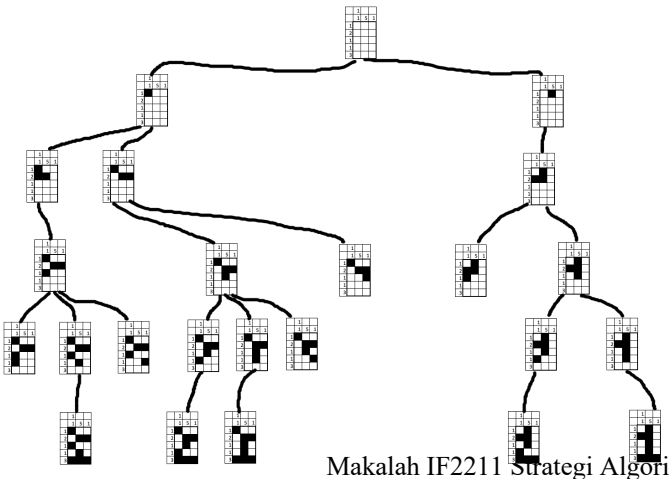


Figure 14 – State tree

Even though the process takes a lot of time and memory space, this algorithm is pretty much effective to solve most of the puzzle.

Here is the execution result of the program with the input of the grid I used in this paper.

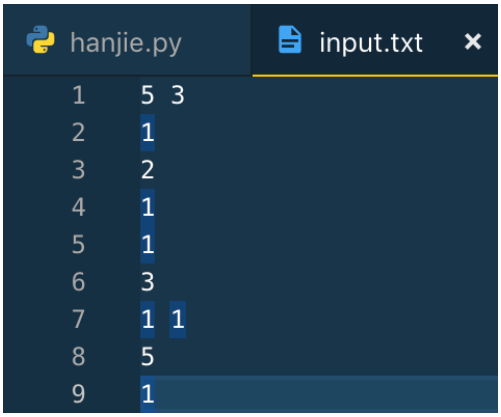


Figure 15 - Input example

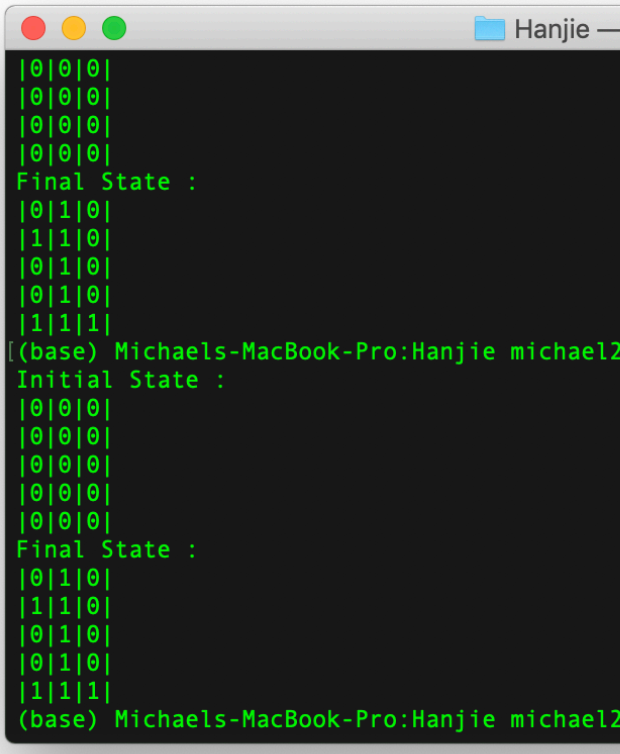


Figure 16 - Example of output from previous input

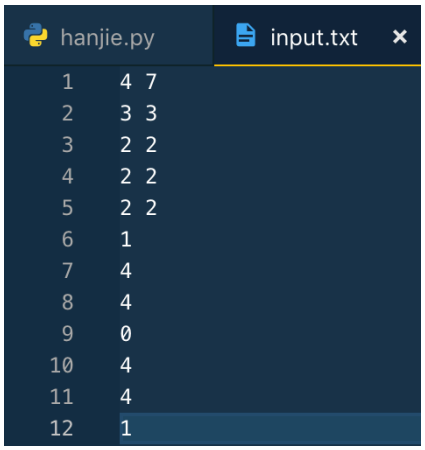


Figure 17 - Example of input 2

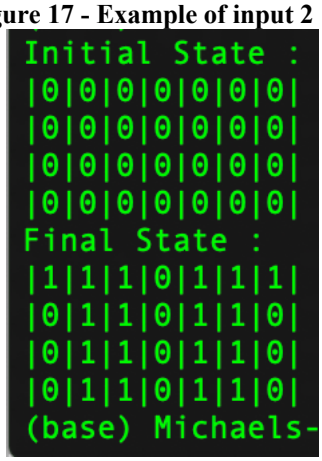
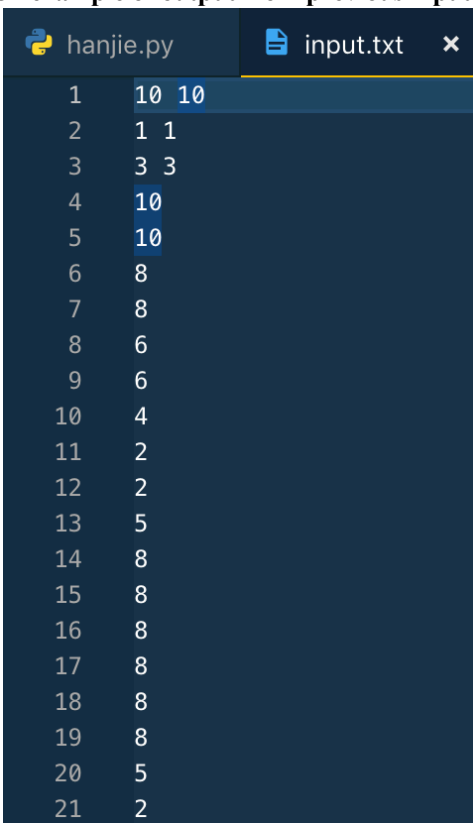


Figure 18 - example of output from previous input



IV. Results

Using the DFS algorithm for solving Nonogram Puzzles could be efficient if the board size is not too big, for bigger sizes, the program will take a lot of time backtracking which is really inefficient.

As I have said before, this problem can't be solved in polynomial time. Human logic is far too complicated for a program to copy, this is why this problem won't be fully solved using this algorithm, this algorithm will at most solve 80-90% of all the puzzles.

Acknowledgment

I would like to express my special thanks of gratitude to my teacher who gave me the golden opportunity to do this wonderful project on the topic, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them.

Secondly I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

References

[1] https://stackoverflow.com/questions/813366/solving-nonograms-picross

[2] https://www.puzzlemuseum.com/griddler/gridhist.htm Accessed at 25 April 2019

[3] Munir, Rinaldi. 2006. *Strategi Algoritma*. Bandung: Institut Teknologi Bandung.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

Michael Ray/13517092