HITORI SOLVER

Presented by:

Pama, Jose Arniel

Rodriguez, Icel Ann

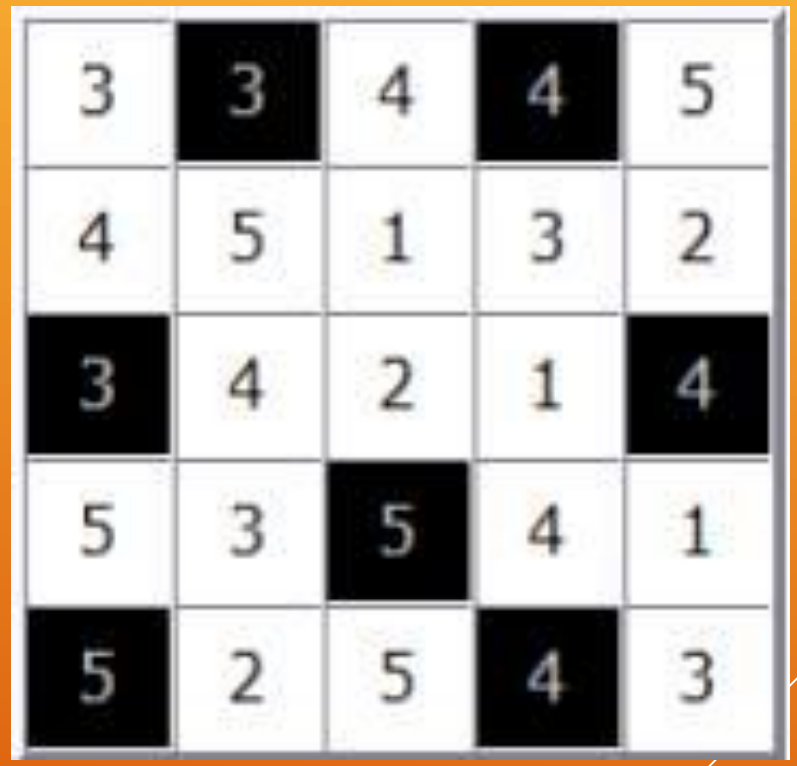
Arsolon, Lovely Grace

Tahadlangit, Chezkah Kate

OBJECTIVE

- To solve a given Hitori puzzle in a short period of time than manually solving the puzzle.

3	3	4	4	5
4	5	1	3	2
3	4	2	1	4
5	3	5	4	1
5	2	5	4	3



SCOPES AND LIMITATIONS

- This Hitori Solver is in C language.
- This Hitori Solver is limited to some Hitori puzzle.
- This is only applicable to Hitori puzzles with equal length of rows and columns.
- This is only applicable to Hitori Puzzles with solutions.
- This provides only one of the many possible solutions of the given Hitori Puzzle.
-

HITORI

Hitori (Japanese for: Alone or one person) (ひとりにしてくれ Hitori ni shite kure; literally "leave me alone") is a type of logic puzzle published by Nikoli.

A Hitori Puzzle always have an equal row length and column length. Therefore, a Hitori puzzle is always a square.

A normal Hitori Puzzle has dimension of 3x3 as the smallest and 20x20 as the biggest Hitori Puzzle known. Other dimensions include 5x5, 6x6, 7x7, 8x8, 9x9, 10x10, 12x12, 15x15 17x17.

Hitori is played with a grid of squares or cells, and each cell contains a number. The objective is to eliminate numbers by filling in the squares such that remaining cells do not contain numbers that appear more than once in either a given row or column. Filled-in cells cannot be horizontally or vertically adjacent, although they can be diagonally adjacent. The remaining unfilled cells must form a single component connected horizontally and vertically.

HITORI RULES:

DEFINITION OF TERMS

- To whiten/eliminate: To eliminate an element means to exclude it from the shading process.
- To blacken: To blacken an element is to shade it. When an element is shaded, it means that it is repeating within a certain row or column it is in.
- Is safe: An element is considered safe if it is not vertically or horizontally adjacent with other shaded element(s).
- Three adjacent numbers: Three numbers appearing consecutively in a row or column.
- Number between a pair: A number that exists between an equal numbers.
- 000 – Marks the numbers as permanently white, therefore it should not be shaded.
- 111 – Marks the numbers as permanently black, therefore it is shaded.
- '#' - This represents the shaded numbers in the final output.

INPUT AND OUTPUT TYPES

- This code reads a given input file that consists a 2D array representing the Hitori Puzzle.
- The Output is directly displayed in the terminal/ console.

1) Look for three adjacent numbers (could be vertically or horizontally). Eliminate the middle number and blacken the other two.

2) Whiten the numbers around the shaded elements. If a number has been circled to show that it must be white, any same number in that same row and column must also be black (considering that it is safe).

3) Look for a number between two equal numbers (number between a pair). Whiten that number. If a number has been circled to show that it must be white, any same number in that same row and column must also be black (considering that it is safe). Repeat step 2.

4) Go back to the first element of the puzzle. If it is neither shaded or unshaded, check if it is nonrepeating in either or column, and if it is, whiten that element. If a number has been circled to show that it must be white, any same number in that same row and column must also be black (considering that it is safe).

5) Traverse vertically and repeat step 4 until solved.

ALGORITHM

1) Run threeAdjacentNumEliminate(rowLen, colLen, grid).
2) Run betWeenAPairEliminate(rowLen, colLen, grid).
3) Run solveHitori(rowLen, colLen, grid).

ALGORITHM

FUNCTIONS

- void printGrid() = prints the elements of a 2D array
- int isSafe () = determines whether a certain element is safe.
- int colCountPaubos() = returns how many times a certain element is repeated from its index(location) and downward.
- int colCountPataas() = returns how many times a certain element is repeated from its index and upward.

- int colCount() = returns how many times a certain element is repeated in a column.
- int rowCountRight() = returns how many times a certain element is repeated from its index to its right.
- int rowCountLeft() = returns how many times a certain element is repeated from its index to its left.

FUNCTIONS

FUNCTIONS

- int rowCount() = returns how many times a certain element is repeated in its row.
- bool isNeitherShadedOrUnshaded() = returns true if an element is shaded and false otherwise.
- blackenSaRow() = blackens all the numbers that is the same with the given value in that row
- blackenSaColumn() = blackens all the numbers that is the same with the given value in that COLUMN
- void whiten() = non-repeating elements are “whitened” and excluded in the shading process.
- int isCorner() = determines whether an element is located in either of the corners of the hitori puzzle.

FUNCTIONS

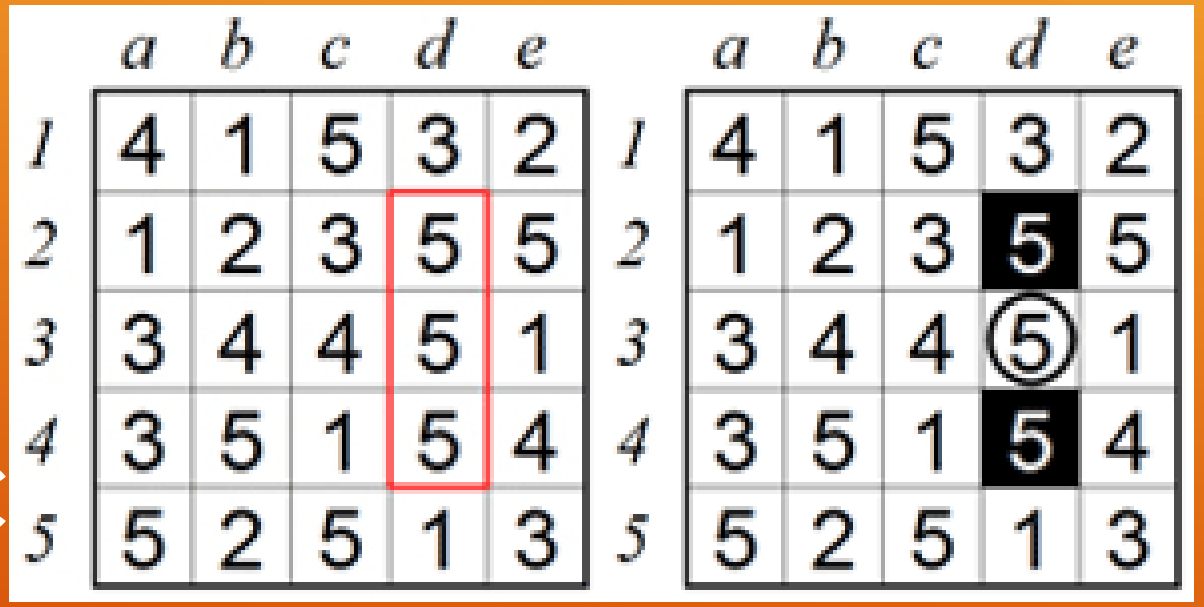
- int isEdge() = determines whether an element is located in the puzzle's edges.
- void eliminateAroundCorner() = eliminates the element located in the corner that is excluded in the shading process. It is eliminated through changing its value to 000.
- void eliminateAroundEdges() = eliminates the element located in the edge that is excluded in the shading process. It is eliminated through changing its value to 000.
- void eliminateAroundNonEdges() = eliminates the element located in the non-edges part of the puzzle that is excluded in the shading process. It is eliminated through changing its value to 000.

FUNCTIONS

- void eliminateAroundShadedParts() = eliminates the element located around a shaded part or cell that is excluded in the shading process. It is eliminated through changing its value to 000.
- void threeAdjacentNumEliminate() = eliminates the middle part of three adjacent numbers could be vertically or horizontally and therefore shading the two remaining.
- void betweenAPairEliminate() = eliminates elements that are located between two equal numbers.
- void eliminateNonRecurring() = eliminates elements that are non repeating in the puzzle.
- void solveHitori() = main function; performs all the functions mentioned and explained above. It prints the solved Hitori puzzle.

TEC

1. Searching for adjacent triplets:



	a	b	c	d	e
1	4	1	5	3	2
2	1	2	3	5	5
3	3	4	4	5	1
4	3	5	1	5	4
5	5	2	5	1	3

	a	b	c	d	e
1	4	1	5	3	2
2	1	2	3	5	5
3	3	4	4	(5)	1
4	3	5	1	5	4
5	5	2	5	1	3

TECH

2. Square between a pair:

	a	*b*	*c*	*d*	*e*
1	3	3	2	1	5
2	2	2	4	5	3
3	3	1	5	3	2
4	2	4	2	3	5
5	1	2	3	3	4

	a	*b*	*c*	*d*	*e*
1	3	3	2	1	5
2	2	2	4	5	3
3	3	1	5	3	2
4	2	④	2	3	5
5	1	2	3	3	4

RECOMMENDATION FOR FUTURE STUDIES

- Improve the code to make it applicable for all levels of Hitori Puzzle.
- Make the code interactive for the user.
- Use a different language wherein they can possibly add some graphics.
- They can also make a legit program (A game/ App) that is a game of Hitori where the program can count how many Hitori puzzle the user had correctly answer and how much time it took for that user to solve the puzzle.
- Counts how many possible solution there is for a given Hitori Puzzle.

OTHER INFORMATION

Hitori Generator

http://hitori-generator.soft112.com/download.html

Hitori Solver using Copris

http://bach.istc.kobe-u.ac.jp/copris/puzzles/hitori/index.html

Hitori Solver using Python

https://github.com/RonMidthun/hitori.py/blob/master/hitori.py

REFERENCES

- http://www.conceptispuzzles.com/index.aspx?uri=puzzle/hitori/techniques