Pyramids

bvuongle

Jan 21, 2023

MODULE LIST:

CHAPTER

ONE

PYRAMIDS

1.1 board module

class board.**Board**(*dim=0*, *board=[]*)

Bases: `object`

Board class is used to store the data of a square board with a specified dimension.

checkDimNBoardRelation(*dim: int*, *board: list*)

Check if in initial tuple the value of dimension is equal to the actual dimension in the table.

Parameters

- **dim** (*int*) – dimension is passed to the function
- **board** (*list*) – board is passed to the function

Raises

WrongDimension – exception

property dim: int

Dimension of the board.

Returns

dimension of this object

Return type

int

property board: list

The data contained in the board.

Returns

data of this object

Return type

list

fillBoardWithValue(*value*)

Used to fill every cells in the "board" property with the value "value". This method will update the data of the "board" property and return None

Parameters

value (*int*) – Values to be placed in board cells

1.2 hints module

class hints.**HintsData**(*dim=0*, *topHint=[]*, *botHint=[]*, *rightHint=[]*, *leftHint=[]*)

Bases: object

HintsData class contains input data, which are indicators of how many pyramids can be seen from a given position.

There are four types of hints: topHint, botHint, rightHint, leftHint.

Along with that is the dim parameter that will determine the size of the problem.

static checkDim(*lists: list*) → int

Check if the list of hint types has the same data length.

Parameters
lists (*list*) – list of hint types

Raises
LengthFileIncorrect – exception - inconsitent data length

Returns
valid dimension value

Return type
int

checkHint(*lst: list*) → list

Check if the value in the list of hints is valid.

Parameters
lst (*list*) – list of hints

Raises

- **NonStandardChars** – exception - data contains non-standard characters - not digits
- **OutsideRange** – exception - data are numbers outside the range 1 to N, resulting in an unsolved problem

Returns
list of hints data has been normalized

Return type
list

property dim: int

Dimension of the problem

Returns
dimension

Return type
int

property topHint: list

Hint at the top of the board applies to the columns viewed from the top.

Returns
Hints of type 1

Return type
list

property botHint: list

Hint at the bottom of the board applies to the columns viewed from the bottom.

Returns
Hints of type 2

Return type
list

property rightHint: list

Hint on the right side of the board applies to rows viewed from the right.

Returns
Hints of type 3

Return type
list

property leftHint: list

Hint on the left side of the board applies to rows viewed from the left.

Returns
Hints of type 4

Return type
list

getData(*dir*)

Get data from file File is only allowed to contain 4 lines, representing the hint for the top, bottom, right and left

Parameters
dir (*str*) – path to input file

Raises

- **FileNotFoundError** – exception - File not found in "dir" path
- **LengthFileIncorrect** – exception - File contains more or less than 4 lines of data

1.3 condition_analysis module

class condition_analysis.**CondBoard**(*dim: int*, *board=[]*)

Bases: *Board*

This is a class that inherits from the Board class, where each cell of the board is a list instead of an integer. This class will analyze the suggested dataset and generate a list of values that can be put in each board cell, improving time and memory when creating configurations.

static noSolutionCheck(*hint: list*) → bool

Check if the list of hints has the ability to give a solution The problem has no solution if there is more than one value 1 or N in the list.

Parameters
hint (*list*) – List of hints

Returns
This problem have solution or not?

Return type
bool

checkMissingData(*lst: list*) → bool

Check if the data length of the list of hints is enough to generate the condition board or not?

Parameters
lst (*list*) – List of hints

Returns
Insufficient data or not?

Return type
bool

setNdelValueCol(*row: int*, *col: int*, *value: list*) → None

Set this value for cell at (row, column) and remove it from cell's list in the same column.

Parameters

- **row** (*int*) – index of the row of the cell that needs to be changed
- **col** (*int*) – index of the col of the cell that needs to be changed
- **value** (*list*) – the list of values to be assigned to the cell, possibly a list with one element

setNdelValueRow(*row: int*, *col: int*, *value: list*) → None

Set this value for cell at (row, column) and remove it from cell's list in the same row.

Parameters

- **row** (*int*) – index of the row of the cell that needs to be changed
- **col** (*int*) – index of the col of the cell that needs to be changed
- **value** (*list*) – the list of values to be assigned to the cell, possibly a list with one element

topCond(*topHint: list*) → None

Analyze and eliminate impossibility based on type one hints.

Parameters
topHint (*list*) – List of type one hints

Raises

- **InsufficientData** – exception - raise when missing data occurs
- **NoSolutionError** – exception - raise when there is no satisfactory solution to this hint

botCond(*botHint: list*) → None

Analyze and eliminate impossibility based on type two hints.

Parameters
botHint (*list*) – List of type two hints

Raises

- **InsufficientData** – exception - raise when missing data occurs
- **NoSolutionError** – exception - raise when there is no satisfactory solution to this hint

rightCond(*rightHint: list*) → None

Analyze and eliminate impossibility based on type three hints

Parameters
rightHint (*list*) – List of type three hints

Raises

- **InsufficientData** – exception - raise when missing data occurs
- **NoSolutionError** – exception - raise when there is no satisfactory solution to this hint

leftCond(*leftHint: list*) → None

Analyze and eliminate impossibility based on type four hints

Parameters
leftHint (*list*) – List of type four hints

Raises

- **InsufficientData** – exception - raise when missing data occurs
- **NoSolutionError** – exception - raise when there is no satisfactory solution to this hint

remRedundantCond() → None

Since the above four methods are not executed concurrently, it is possible to result in some impossibility of configuration that has not been eliminated. This method will "double check" and delete it.

analyzeBasicCond(*hints:* HintsData) → None

Method used to separate hint types and call the respective methods declared above, for convenience :)

Parameters
hints (HintsData) – Data of hints

1.4 board_resolver module

class board_resolver.**BoardResolver**(*hints=<hints.HintsData object>*)

Bases: object

BoardResolver class stores the data of a problem, including the input data: hints, the condition board and the problem results (if it's exist). It also includes methods that are used as tools to solve the problem from input data.

property flag

This flag marks if the current object is storing the configuration as the solution of the problem or not. If flag is 0, then no solution has been found yet, and vice versa if flag is 1

Returns
value of flag

Return type
int

property curBrd

Current configuration of the board for the problem. By default, all cells in the board have a value of 0. If flag = 1, then this property is the answer to the problem.

Returns
Current configuration of the board

Return type
Board

property condBrd

Condition board is created for the problem

Returns
condition board

Return type
CondBoard

property hints

Input data of the problem - hints

Returns
list of lists of hints :)

Return type
HintsData

static getRow(*matrix*, *row*) → list

Returns a list of the current value of the specified row in a matrix

Parameters

- **matrix** (*list*) – Matrix for which we need to take the value
- **row** (*int*) – Row in the matrix that we need to get the value

Returns
List of values in current row of the matrix

Return type
list

static getCol(*matrix*, *col*) → list

Returns a list of the current value of the specified column in a matrix

Parameters

- **matrix** (*list*) – Matrix for which we need to take the value
- **col** (*int*) – Column in the matrix that we need to get the value

Returns
List of values in current column of the matrix

Return type
list

static numVisiblePyramids(*arr*) → int

Calculates the length of the longest incremented sequence that starts with the first element of the array.

Used to calculate how many pyramids can be seen at the specified location.

Returns
the length of the longest incremented sequence

Return type
int

saveData(*dir*)

Store the result of the problem in a file if flag = 1 (found the answer)

Parameters
dir (*str*) – the path to the file where answer will be saved

Raises
FileNotFoundError – exception - directory is empty

checkResultWithCond() → bool

Check if the current configuration of the object is satisfied with the input condition.

At each row/column, we will get the list of values of that row/column, calculate how many pyramids we can see with this configuration and compare it with the corresponding hint (if the hint is not 0).

Returns
Is the answer right or wrong?

Return type
bool

backtracking(*row: int*, *col: int*)

Backtracking algorithm, where the problem is solved. This algorithm will generate all possible configurations of the board (based on the condition board and the values that have already been placed in the board) until it finds a match.

Parameters

- **row** (*int*) – current row that the algorithm considers
- **col** (*int*) – current column that the algorithm considers

resolver()

Method prepares the condition table and calls the method to solve the problem.

Raises
NoSolutionError – exception - can't find the solution

1.5 main module

class main.**ErrorDialog**(*parent=None*)

Bases: QDialog

Error message interface

closeDialog()

class main.**PyramidsWindow**(*parent=None*)

Bases: QMainWindow

The main interface of the GUI

static setTableValue(*table*, *data*)

showHome()

Display home screen interface

showSolve()

Display solve screen interface

showHelp()

Display help screen interface

showAbout()

Display about screen interface

showError(*error*)

Display error screen interface

Parameters
error (*Exception*) – type of error

exitProg()

clearAnsTable()

renewBoard()

Change the size of the board display interface according to the size selected by the user

resetBoard()

Reset all solve screen interface to default

getInputData() → bool

Read input data entered from the user through the GUI and run the method to solve it.

getDataFrFile() → bool

Read input data from file and display it on GUI

pasteInputData(*hints:* HintsData)

Display input datasets to GUI

saveDataToFile() → bool

Save the result (if exists) to file

solve(*hints:* HintsData) → None

Method solves the problem and prints it to the GUI. This method receives input data read from the GUI

Parameters
hints (HintsData) – Input data read from GUI

main.**rapidSolver**(*dir: str*) → *BoardResolver*

Method used to solve the problem without the need for a GUI. Required to get the path to the input data file

Parameters
dir (*str*) – path to input data file

Returns
object carries the solution of the problem (if exists)

Return type
BoardResolver

main.**guiMain**(*args*)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

P

R

S

T