Assignment 1

COS212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Deadline: 24/03/2023 at 23:59

Marks: 400

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you must implement them yourself.
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire assignment before you start coding.
- You will be afforded three upload opportunities.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.** Also, note that the OOP principle of code reuse does not mean you should copy and adapt code to suit your solution.

3 Outcomes

Upon completing this assignment, you will have implemented the following:

- A linked list to help with the rest of the assignment
- A 3D sparse table to create a Sudoku board
- Iterative methods to solve Sudoku. *Note: For complexity reasons, only three techniques are implemented, which means that most easy and medium Sudoku will be solved, but the algorithm will not always find the solution for more difficult puzzles, which require additional techniques.*

4 Background

4.1 Linked Lists

Linked lists were covered in detail in COS110 and, as such, won't be explained in detail in this assignment. This assignment will use a simple, singly linked link list. If a recap is needed, please refer to your COS110 lecture notes.

4.2 Sparse Table

Sparse tables are data structures used when data is stored in a table format but which might contain gaps. An array of linked lists are used to connect all the data points. A sparse table is used to eliminate gaps in regular 2D arrays. This assignment will use a 3D sparse table to store the data for a Sudoku grid. The three arrays are for rows, columns and blocks. This way, the linked lists can easily traverse every row, column and grid without needing to calculate which cells are part of each set. Figure 1 shows an example of how the cells are linked.

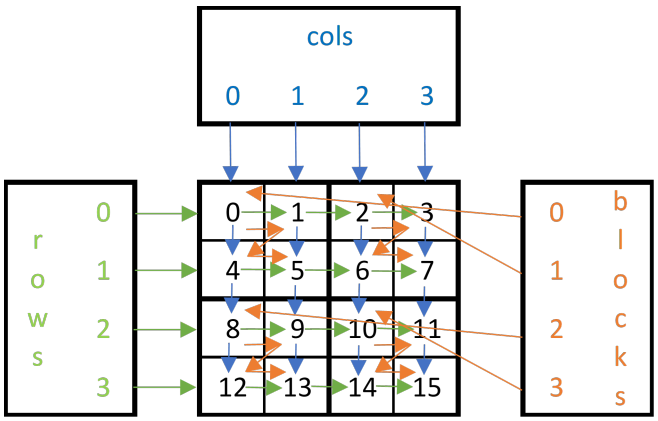


Figure 1: Sparse Table Linking

4.3 Sudoku

Sudoku is a puzzle game that first appeared in the 18th Century. The puzzle consists of a 3x3 grid of 3x3 cells. A cell is a space where a single number can be filled in. In traditional Sudoku, a cell can be any number between 1 and 9 inclusive. A 3x3 grid of cells is called a block; in this assignment, a 3x3 grid of blocks is called a board. The goal is to fill every cell on the board with a number without having any duplicate numbers in any of the rows, columns and blocks.

This assignment uses a more general version of Sudoku where an $m \times n$ grid is used. For clarity reasons, we use the variables:$numRows \times numCols$. In this format, every cell can contain the values [1,$numRows * numCols$], and the normal no duplicate rule applies. For example, in a 2x3 grid, the values [1,2*3] = [1,6] are allowed. Thus each row, column and grid must contain the values 1 through 6, shown below.

4	6	2	5	3	1
3	1	5	6	2	4
6	4	3	1	5	2
2	5	1	4	6	3
5	3	4	2	1	6
1	2	6	3	4	5

(a) 2x3 Board

4	3	6	2	5	1
6	1	4	5	3	2
2	5	3	1	4	6
5	6	1	4	2	3
3	2	5	6	1	4
1	4	2	3	6	5

(b) 3x2 Board

Figure 2: Solved Sudoku Examples

5 Tasks

Please note that the UML provided is the minimum required, and students may expand on it if necessary. **Note that the functions and variables shown in the UML may not be removed, changed or left unimplemented**, but you may add your own variables and functions to help implement the listed functions.

This assignment can be split into three tasks, making it easier to test since every task can be tested before moving on to the next. Task 1 is creating a Singly Linked Link List.

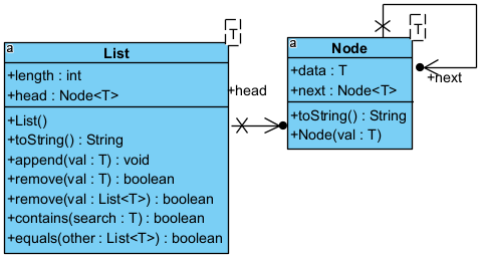


Figure 3: Task 1 UML

Task 2 creates the Cell and Board class and some basic functionality to test that the links are setup correctly. Task 3 is then used to try and solve the puzzle.

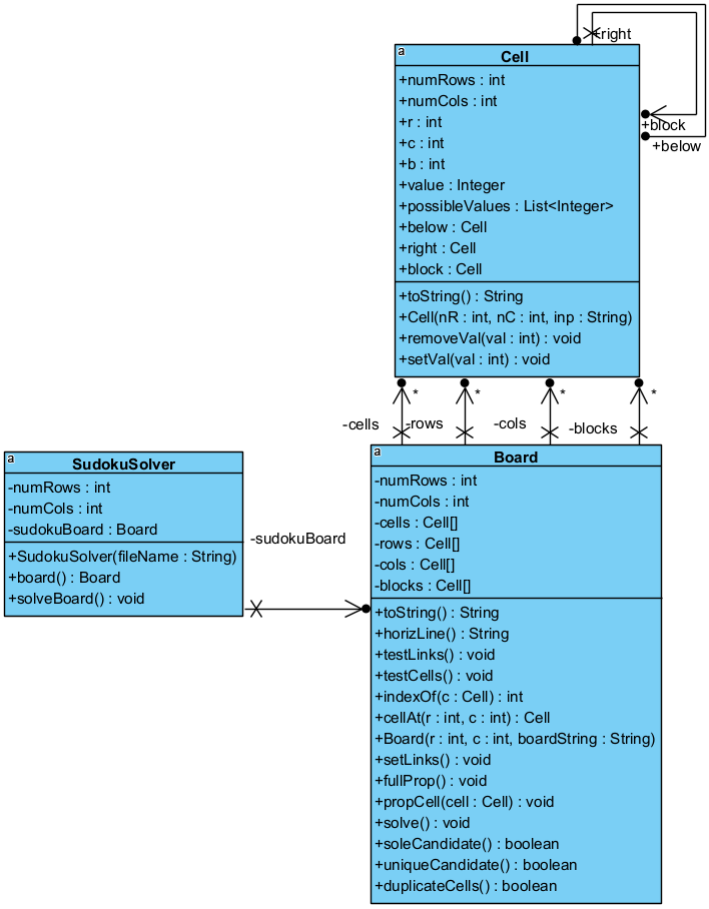


Figure 4: Task 2 & 3 UML

5.1 Node<T>

The node class is complete and has very little functionality on its own. There is no need to modify this class, but you may if you see fit.

5.2 List<T>

This is a generic linked list class which will be used in future tasks.
Note: There is a Java class called List, but you have to use and create your own version of it. Make sure you don't accidentally import the default Java one.
Tip: When using generics in Java, don't use == when comparing variables; rather, use .equals()

- Members
  - length:int
    - This is the number of nodes currently in the linked list and is updated whenever a function changes its size.
  - head:Node<T>
    - This is the head of the linked list.
    - If the list is empty, this variable will be null.
- Functions
  - List()
    - The constructor for the linked list, which initialises all members.
  - toString():String
    - Returns a string representation of the list.
    - The nodes should be comma-delimited and NOT end with a new line.
    - Ex: "1,2,3,4"
  - append(val:T):void
    - Create a new node with the given value and add it to the end of the list.
  - remove(val:T):boolean
    - If there is a node in the list with the given value remove the first occurrence of it.
    - If a node was found and removed, return true otherwise, return false.
  - remove(val:List<T>):boolean
    - Call the remove function with every node in the val List.
    - If zero nodes were deleted, return False, otherwise, return True.
    - Ex:
      Original List : "1,2,3,3,4,4,5"
      val List: "3,2,5"
      Resulting List: "1,3,4,4"

- contains(search:T):boolean
  - Returns true if the given variable is found inside the list; returns false otherwise.
- equals(other:List<T>):boolean
  - Returns true if the otherList equals this list returns false otherwise.
  - Lists are equal if the lengths are the same and the corresponding indices match.
  - Ex:
    "1,2,3,4" and "1,2,3,4" are equal
    "1,2,3,4" and "1,3,2,4" are not equal.

5.3 SudokuSolver

This class is complete, and there is no need to modify it. It is recommended that you read through the code given since the constructor shows how to read from a text file and some basic String manipulation in Java which will not always be given in future Practicals and Assignments. The constructor assumes a certain format for the textiles ,which is shown below:

```
{numRows}x{numCols}
1 2 3 4 5 6 - - -
1 2 3 4 5 6 - - -
1 2 3 4 5 6 - - -
...
```

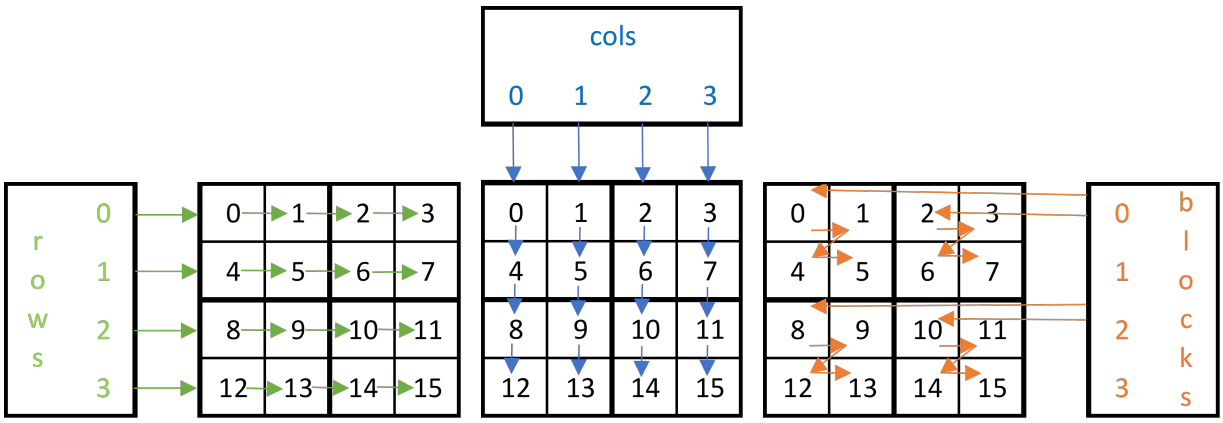
- numRows is the number of rows per block
- numCols is the number of columns per block
- - is used for empty cells

5.4 Cell

- Members
  - numRows:int
    - The number of Rows in the Sudoku grid.
  - numCols:int
    - The number of Columns in the Sudoku grid.
  - r:int
    - The row this cell is part of. This corresponds to the index inside the rows array from the Board class. This will be in the range $[0, numRows \times numCols)$.
  - c:int
    - The column this cell is part of. This corresponds to the index inside the cols array from the Board class. This will be in the range $[0, numRows \times numCols)$.

  - b:int
    - The block number this cell is part of. This corresponds to the index inside the blocks array from the Board class. This will be in the range [0,$numRows \times numCols$).
  - value:Integer
    - If the cell is filled in, this Integer will be the value of the cell otherwise, it is null. *Note in Java there is a difference between 0 and null.*
  - possibleValues:List<Integer>
    - This is a linked list of all possible values this cell can still be. This variable is null if the cell already has a value filled in.
  - below:Cell
    - This is the next cell used to link the columns.
  - right:Cell
    - This is the next cell used to link the rows.
  - block:Cell
    - This is the next cell used to link the blocks.
- Functions
  - toString():String
    - This function is given to you and there is no need to change it.
    - This returns a string representation of the cell padded to work with the Board's toString function.
  - Cell(nR:int,nC:int,inp:String)
    - nR is used to set numRows.
    - nC is used to set numColumns.
    - inp is a String which will be either "-" or the String version of the number. If the String is a number set value to the number and possibleValues to null. If the String is "-" set value to null and possibleValues such that it is a list with all values which are possible.
  - removeVal(val:int):void
    - If possibleValues is not null, call the remove function on possibleValues with the passed-in input. Otherwise, do nothing.
    - This function will be used later if a certain value is not valid for this cell.
  - setVal(val:int):void
    - Set the value variable to the passed-in parameter and set possibleValues to null.

5.5 Board



(a) rows linking (b) cols linking (c) blocks linking

Figure 5: Board linking

- Members
  - numRows:int
    - The number of rows in each block.
  - numCols:int
    - The number of columns in each block.
  - cells:Cell[]
    - An array of Cells. This is used to store all of the cells in the board. The cells are ordered from left to right and then from top to bottom. In Figure 5, the black numbers show the indices of the array.
    - This is a flattened 1D representation of the grid.
  - rows:Cell[]
    - The rows array is used to store the heads of the rows. In Figure 5, the array stores the Cells at positions 0,4,8 and 12. From there the right links can be used to traverse every row.
  - cols:Cell[]
    - The cols array is used to store the heads of the columns. In Figure 5, the array stores the Cells at positions 0,1,2 and 3. From there the below links can be used to traverse every column.
  - blocks:Cell[]
    - The blocks array is used to store the heads of the blocks. In Figure 5, the array stores the Cells at positions 0,2,8 and 10. From there the block links can be used to traverse every block.

- Functions
  - toString():String, horizLine():String
    - These functions are given. Don't change them since they will be used to mark your output when submitting.
    - These functions are used to print the Sudoku Board. These functions will correctly display the Sudoku board and add padding if necessary.
  - indexOf(c:Cell):int, cellAt(r:int,c:int):Cell
    - These two functions are given to you. They are used for functions mentioned in the previous bullet. You may use them if you see a need for them in the other functions.
  - testLinks():void, testCells():void
    - These functions are given. Don't change them since they will be used to mark your output when submitting.
    - These functions are used for marking and testing. You may use them to check that your cells are linked correctly, but they are also used for marking. If you want to expand on their functionality make copies of them so that you don't change the original functions.
  - setLinks():void
    - Create the rows, cols and blocks arrays.*Hint: They should all be the same size.*
    - The arrays should store the head of each linked list used for traversing the grid. Look at Figure 5 to see how the Cells are linked. Use the cells array to link the cells below, right and block links.**Don't create deep copies of the cells**.
    - Use the given Main to see if your links are set up correctly for the given example. For examples with other size grids, use the included spreadsheet to see how other sizes should be ordered and linked.
    - *Hint: Start by trying to figure out which nodes are the first ones in their blocks. From this you can use the below and right pointers to get the rest of the block.*
  - Board(r:int,c:int,boardString:String)
    - Start by setting the numRows and numCols variables to the passed-in parameters.
    - Create the cells array to be the correct size. The array should be the same size as the number of cells in the board.*Hint: Look at the provided excel spreadsheet to see how numRows and numCols relate to the number of cells.*
    - The passed-in boardString is used to initialise the cells. Every value is space-separated and there are no newlines. The cells should be separated and used to call the Cell constructor, which automatically deals with the casting and if the cell is empty. Make sure your function can handle any length of numbers and are not hard coded to use single-digit numbers.
    - After the cells array is populated, call the setLinks() function to create the other arrays.

- fullProp():void
  - Iterate through the cells array and call the propCell function with every cell in the board.
- propCell(cell:Cell):void
  - If the passed in Cell has not been filled-in (The value variable is null), return immediately and do nothing.
  - This function propagates the passed-in Cell's value and removes that value from the possibleValues from the other Cells in the row, column and block.
  - Use the r,c and b variables of the passed-in Cell to know which cells to update. Loop through the correct row, column and block and call the removeVal function on every cell, removing the value of the passed-in Cell.
- solve():void
  - This function consists of a single while loop and a counter.
  - The while loop condition is:

```
soleCandidate() || uniqueCandidate() || duplicateCells()
```

  - Inside the body of the loop increment a counter.
  - After the loop print the counter with the following with a new line:

```
Number of moves: {counter}
```

- soleCandidate():boolean
  - This function is the first iterative technique. Loop through all of the cells and if there is a Cell with only one possible value then fill in that value and propagate the change.
  - Pseudocode is shown in Section 8.1
- uniqueCandidate():boolean
  - This is the second iterative technique. Loop through the rows, columns and blocks and if there is only one cell that a certain value can be filled in do that.
  - Pseudocode is shown in Section 8.2
- duplicateCells():boolean
  - This is the third and final iterative technique for the assignment. This is a simpler version of the "Naked Subset" technique from https://www.kristanix.com/sudokuepic/sudoku-solving-techniques.php.
  - Loop through every row, column and block. If there are any two cells that have only two values and the two cells are equal this technique applies. Since those two cells can only be those values those values can be removed from the rest of the group.
  - Pseudocode is shown in Section 8.3

6 Submission instructions

Do not submit any custom function classes.
Your code should be able to be compiled and run with the following commands:
javac *.java
java Main

Once you are satisfied that everything is working, you must create an archive of all your Java code, into one archive called uXXXXXXXX.{tar.gz/tar/zip} where X is your student number. Submit your code for marking under the appropriate link before the deadline. **Make sure your archive consists of only java files and no folders or subfolders.**

Please ensure that you thoroughly test your code before submitting it. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

7 Submission checklist

- Board.java
- Cell.java
- List.java
- Node.java
- SudokuSolver.java

8 Psuedocode

8.1 soleCandidate psuedocode

```
for(Loop through cells wit a loop variable called c){
    if(c has possibleValues and the length of possibleValues is 1){
        Save the data of the head of c.possibleValues into a variable called value
        Call setVal on c using value
        Call propCell with c
        return true
    }
}
return false
```

8.2 uniqueCandidate psuedocode

```
for(Loop through rows with a loop variable called row){
    Create a 1D int array of size {numRows * numCols} and call it counts
    Create a Cell variable called rowPtr and set it equal to row
    while(rowPtr is not null){
        if(rowPtr has possibleValues){
            Create a Node<Integer> variable called nodePtr and set it equal to the first node in the rowPtr
               possibleValues
            while(nodePtr is not null){
                increment counts[nodePtr.data - 1]
                update nodePtr to the next value
            }
        }
        update rowPtr to the next value
    }
    for(i in range [0,numRows * numCols]){
        if(counts[i] == 1){
            rowPtr = row
            while(rowPtr is not null){
                if(rowPtr has possibleValues and contains {i + 1}){
                    Call setVal on rowPtr with {i+1}
                    Call propCell with rowPtr
                }
                update rowPtr to the next val
            }
            return true
        }
    }
}

Repeat the above for cols
Repeat the above for blocks
return false
```

8.3 duplicateCells psuedocode

```
for(Loop through rows with a loop variable called row){
    Create Cell variable called rowPtr and set it equal to row
    while(rowPtr is not null){
        if(rowPtr has possibleValues and the length is 2){
            Create a Cell variable called secondPtr and set it equal to rowPtr.right
            while(secondPtr is not null){
                if(rowPtr.possibleValues equals secondPtr.possibleValues){
                    Create a Cell variable called thirdPtr and set it equal to row
                    Create a boolean variable change and set it equal to false
                    while(thirdPtr is not null){
                        if(thirdPtr is not secondPtr and thirdPtr is not rowPtr and thirdPtr has possibleValues){
                            change = change || thirdPtr.possibleValues.remove(rowPtr.possibleValues)
                        }
                        update thirdPtr to the next val
                    }
                    if(change){
                        return true
                    }
                }
                update secondPtr to the next val
            }
        }
        update rowPtr to the next val
    }
}

Repeat the above for cols
Repeat the above for blocks
return false
```