Phone Book

AVLNode Class

For this problem I decided to use an AVL tree ADT for no particular reason. This class has two constructors. One with arguments number, first name, and last name and another one with added left and right nodes to the arguments. The first one is what the "user" enters and it passes on the arguments to the more complicated constructor which initializes global variables of number, first name, last name, and also the left and right nodes (child nodes).

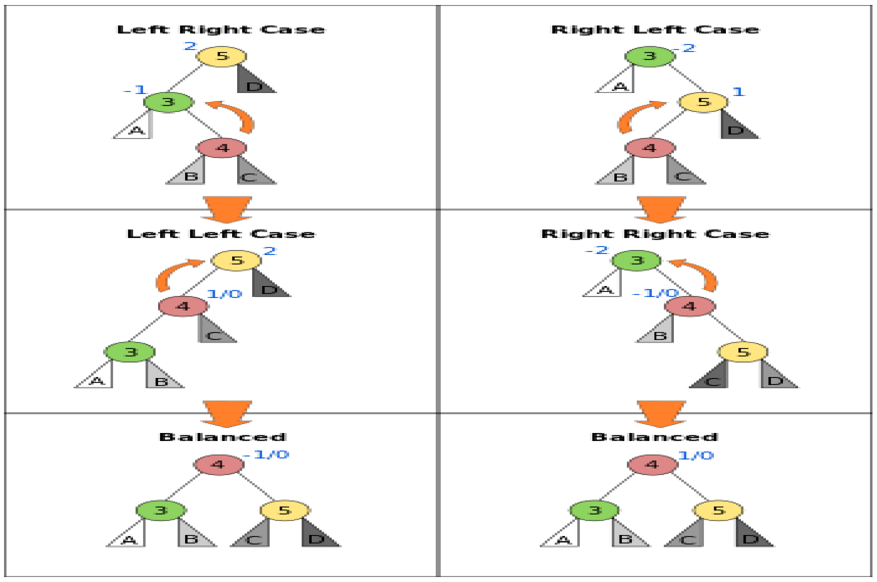
AVLTree Class

AVLTree is a class that has multiple functions that actually makes the tree and organizes all the data and does other algorithms like remove and search. This tree sorts only by phone numbers and each node carries the first name and the last name of the person with that number.

Insert

There are two insert methods, one is used to get the number, first, and last name and then sets the root node to insert of all the above arguments and goes through the actual insertion into the tree with a bunch of ifs and else statements to check exactly where it belongs.

At the start it is null so it just sets the root node to the first inserted node. After the first one is inserted it now checks to see if the next inserted node is less than or greater than the root nodes to see which side of the tree it goes to. Once that is found out it then recursively calls insert until it finds a null node that it can go into. When this is found out we then have to check to see the height of the tree, because all the nodes need to be within 1 node away from each other or less to be balanced. If it is 2 then it needs to rebalance it. There are 4 different cases of this as shown from this picture taken from Wikipedia.



When the appropriate case is found the program then rotates the tree the right way so then the height is equal to 1.

Remove

This remove method gets called in the main with a number and then calls the method with the added root. What this does is basically the same thing as insert but searches for the number. It first starts out at the root and checks if it is less than or greater than it. Once that is found out it then recursively goes through each node always checking greater than or less than until the node is found. After it is found it then checks to see if the left or right children have anything because if it does it has to balance it.

If one or both of the child nodes are not null then it checks to see which side has more height and then balances it accordingly with the cases.

isInsideuser (Search)

This is called by putting a number in its argument, and then it calls isInside method with an added node argument so we can keep track of the nodes it goes through. This uses the same concept as the previous methods as it keeps on checking less than or greater than until it doesn't find it or it does find it and it returns the node so we can get information like first name and even last name from it (Which is useful in the main).

MainTest Class

There is no need to go into too much detail with this class as it is just a GUI that uses buttons as the trigger to call functions such as insert, and it takes the text from the respective fields. There is also small error checks and if and else statements so you have to insert a phone number but you don't need a name, or you have to have a first name and a last name to remove by name.

As I had problems inserting a long into the AVLTree, I opted for making to files a 416 area code, and also a 905 area code. Pressing the appropriate buttons makes the tree for the respective files, and also sets a variable to 1 or -1 so that the save button knows which file to save it all to!

Inorder

Inorder is a method that just makes a list of the whole tree inorder. This makes it so that we can list it in the directory list, and a modified version goes through the same thing so that we can list all the names we searched for in order, and this function also gets modified a little bit deeper (I know bad coding) so that we can use it to search for a name and remove it from the list.

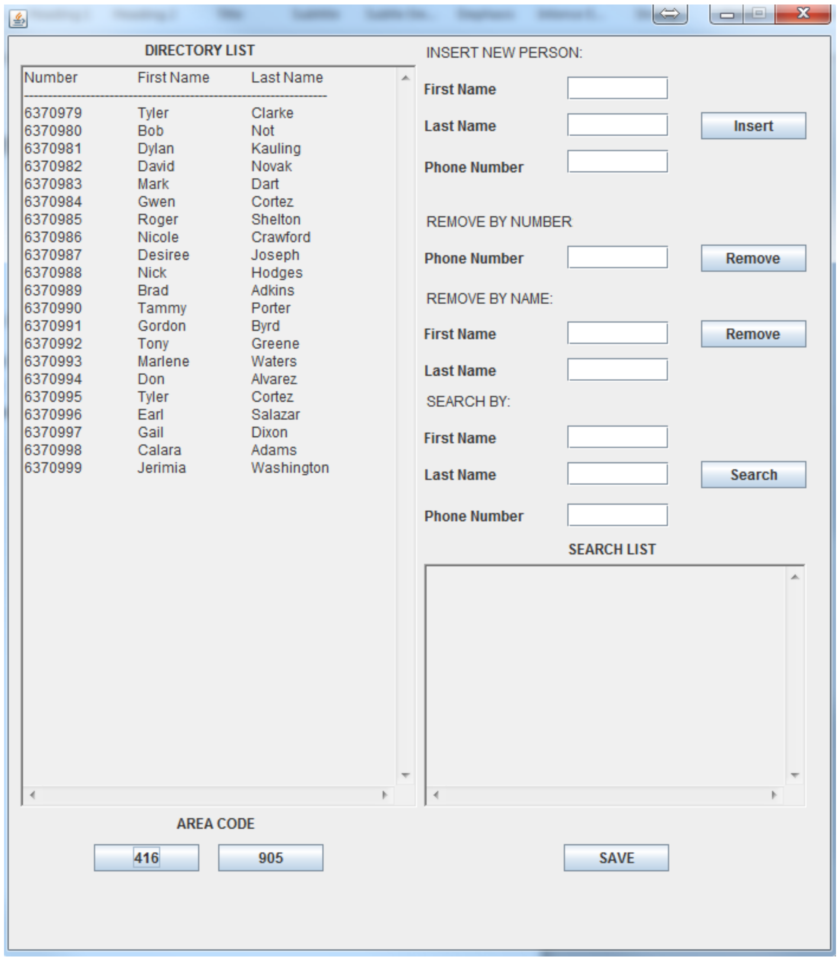
All of this can be saved by hitting the save button and gets saved to the same text file that was taken as the input.

Complexity Analysis

This is split into multiple sections for the different tree functions.

The search algorithm is O(Log n) at worst
The insert algorithm is O(Log n) at worst
The remove algorithm is O(Log n) at worst

Results



Number	First Name	Last Name
6370979	Tyler	Clarke
6370980	Bob	Not
6370981	Dylan	Kauling
6370982	David	Novak
6370983	Mark	Dart
6370984	Gwen	Cortez
6370985	Roger	Shelton
6370986	Nicole	Crawford
6370987	Desiree	Joseph
6370988	Nick	Hodges
6370989	Brad	Adkins
6370990	Tammy	Porter
6370991	Gordon	Byrd
6370992	Tony	Greene
6370993	Marlene	Waters
6370994	Don	Alvarez
6370995	Tyler	Cortez
6370996	Earl	Salazar
6370997	Gail	Dixon
6370998	Calara	Adams
6370999	Jerimia	Washington

All of this gives a nice GUI with all the functions available with textfield inputs and buttons.

Conclusion

In conclusion the AVL Tree is a good and fast way of making a phone book as it is quick to search for a number. The only problem I had which I thought I fixed is sometimes the remove function does not work properly when pressed. Even though it compares two of the exact same ints it cannot find it in the tree. This is fixed by typing in the number or name you want multiple times and pressing remove multiple times.

8 Queens

The only class in this is the EightQueens class which has a few methods that work together to complete the whole problem.

Place

Place is a class that checks if a queen can be placed in the row, column position. It checks if any other queen is on its horizontal, vertical, or diagonal and returns a Boolean.

Queen

The queen method places a queen on the board starting at coordinate 1x1 and goes through placing each queen one at a time checking every possible solution for the queen recursively calling out the queen method for the next queen. This method is slower than other methods and for high N queens values it takes a long time.

Create_Array

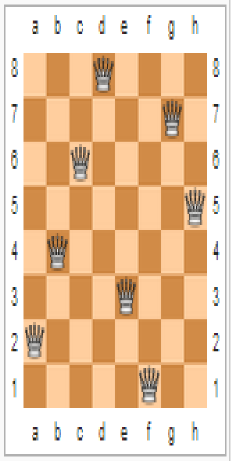
This method takes all the solutions and stores them in an array, so it basically makes a 3D array as a solution is a 2D Array the solution number would be the 3rd dimension. This makes it easy to implement the GUI which is a splitpane gui GUI with selectable solutions.

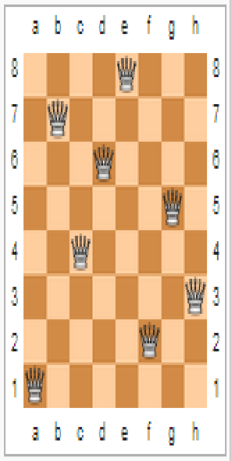
Complexity Analysis

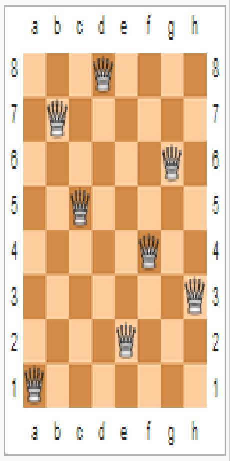
The complexity for this whole problem is quite slow at O(M*N^N) and could even go to O(M*N!) if this gets modified for N queens.

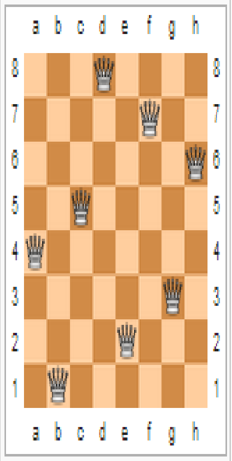
Results

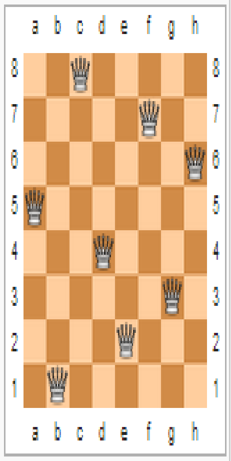
There are 12 explicit solutions to this problem as shown below. The rest of the solutions (up to 92 as seen in our java program) are 12 of these solutions rotated 90 degree positions and their reflections and also 180 degree rotations and their reflections.

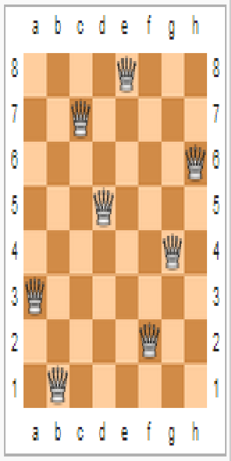

Solution 1

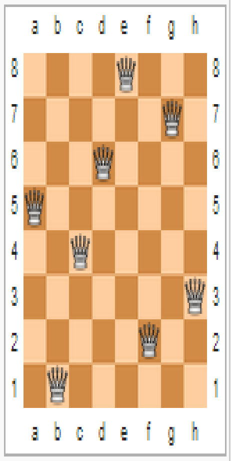

Solution 2

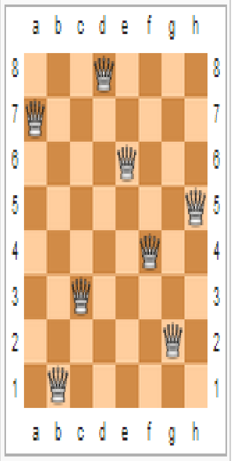

Solution 3

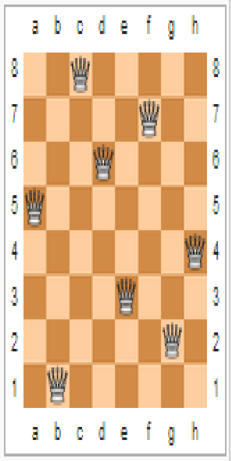

Solution 4

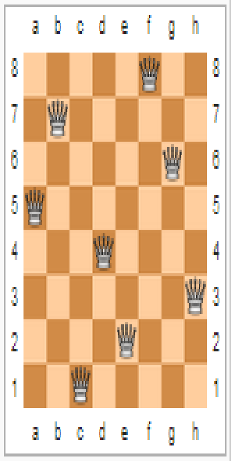

Solution 5

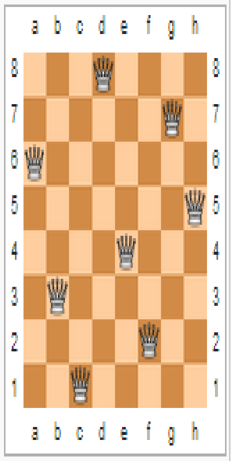

Solution 6

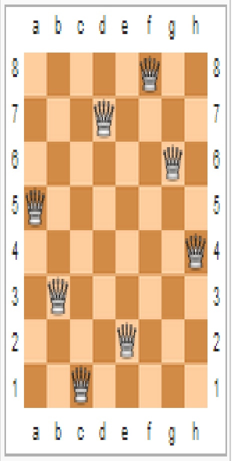

Solution 7


Solution 8


Solution 9


Solution 10


Solution 11


Solution 12

Conclusion

In conclusion, this way of doing the 8 queens problem is probably one of the slowest but does require less code in general because of the recursion. There are better and faster ways the can do N queens = 28 in 14 seconds but are more complicated.

K-Map Minimizer

The only class in this is the KMapSolver class which has several methods that work together to complete the whole problem.

AddLabel

The addLabel method is a simple method to add the labels to the GUI when required for both the truth table and k-map. Some were added with this method and others directly as appropriate.

AddButton

The addButton method is a similar method to the one previous in that it is used to add buttons to the GUI when it is invoked. Again some were added using this and others directly.

ActionPerformed

The actionPerformed method is called when an action occurs on the GUI. The method then determines what kind of action it was and what to do with it. If it determines that the solve button was clicked it triggers the solve method. If it determines a button related to a truth table or k-map value was pressed then it changes the value on both the GUI and in memory.

Solve

Solve is responsible for calling the series of methods used to carry out the full solution of the k-map. It executes the methods in order from createFirstList down to a final generateBoolean. It also prints out the contents of various arrays and integers to the console to allow for a better view of the program's operation behind the scenes and how it follows the Quine-McCluskey algorithm/Tabular method of Minimization through each step of the process.

CreateFirstList

CreateFirstList is a simple method which forms a list consisting of the function's minterms.

CreateSecondList

CreateSecondList is a complex method which seeks to combine the values of the firstList to create new terms which are only one variable different from each other as well as one index level.

CreateThirdList

CreateThirdList does the same as createSecondList but instead using the terms in the secondList rather than firstList to make its combinations. The result being an even more simplified list of terms.

FindTerms

This method creates the chart that seeks to remove redundant prime implicants. When the lists were combined to create the future lists the terms were tagged to indicate if they had been used in a combination at all. If they had not, they are a prime implicant, and thus are added to the chart in this method.

SolveChart

The solveChart method uses the chart two-dimensional array created in the findTerms method to try and find a minimized solution for the truth table/k-map. It iterates through the chart checking for prime implicants that are solely responsible for covering certain minterms. These are essential prime implicants. Other prime implicants must then be found to cover the remaining minterms not covered by the essentials if such minterms exist. The result is a list of terms that together make up all the minterms in the function and will be translated to a Boolean expression.

GenerateBoolean

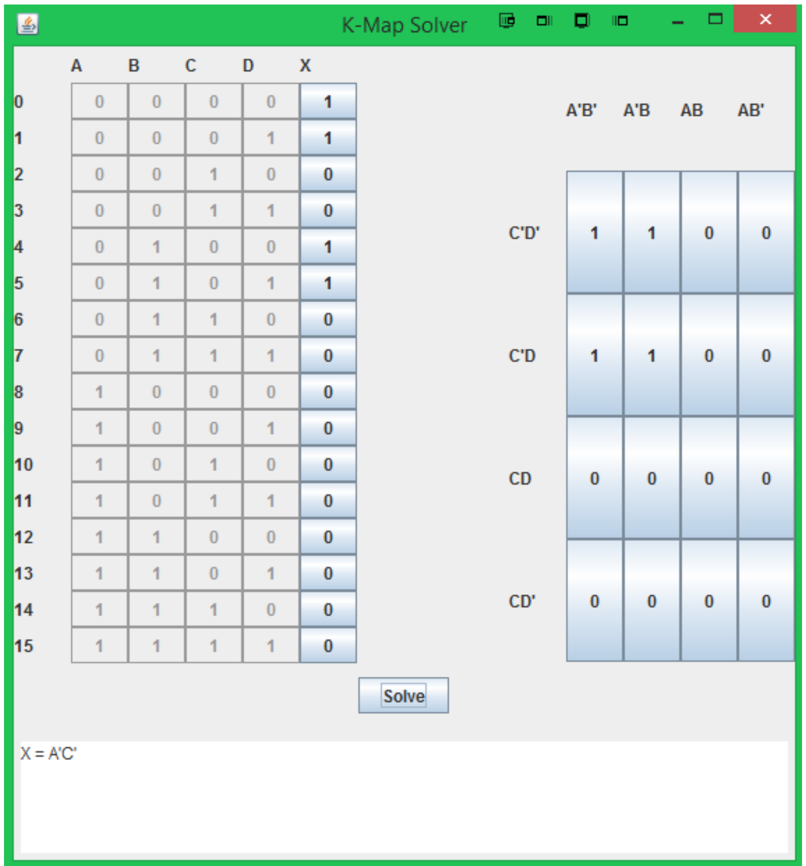
This method generates the final, minimized Boolean expression to be displayed in the GUI text area and sets it for display. It generates the Boolean expression by iterating through the finalTerms array generated by the solveChart method and first counts how many minterms there are. Based on the number of minterms in the finalTerm it determines which list it came from and in extension which actual term it was by matching the minterms to the values of the terms from the original lists generated by their respective functions.

Complexity Analysis

The Quine-McCluskey algorithm is the heart of this program and the algorithm has a complexity that is rated as NP-hard, or Non-deterministic Polynomial time hard. This means that it falls within a class of problems which are at least as hard as the hardest problems of NP, making it hard to classify otherwise in terms of time with respect to the number of variables.

Results

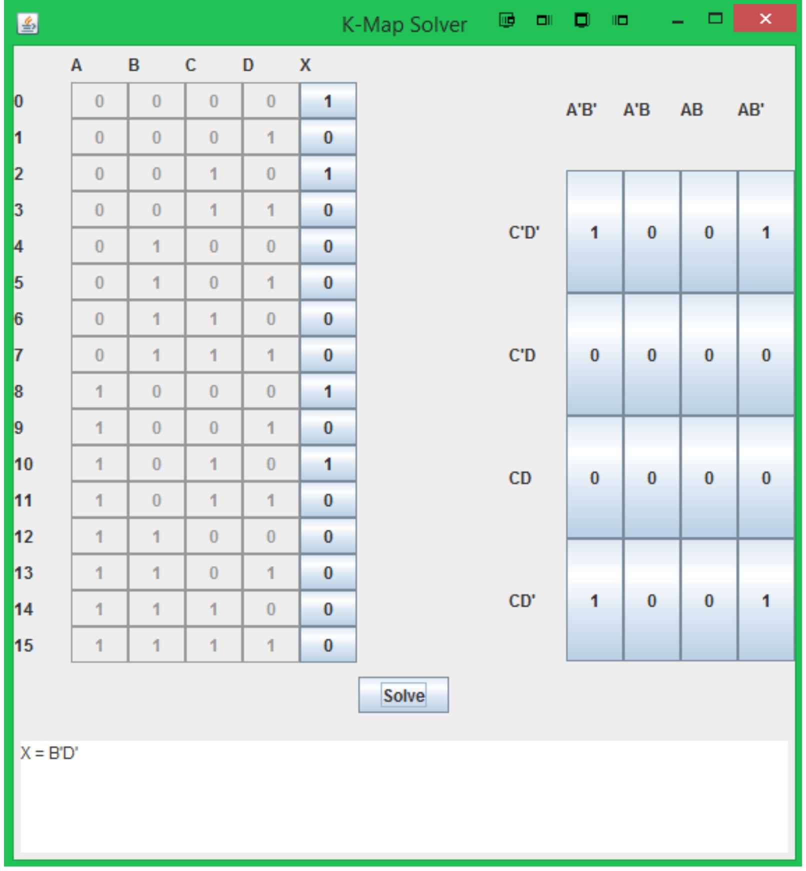
The code isn’t entirely complete for the k-map minimizer; however it still works for several different combinations, such as the ones shown below.



	A	B	C	D	X
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

	A'B'	A'B	AB	AB'
C'D'	1	1	0	0
C'D	1	1	0	0
CD	0	0	0	0
CD'	0	0	0	0

```
X = A'C'
```



	A	B	C	D	X
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

	A'B'	A'B	AB	AB'
C'D'	1	0	0	1
C'D	0	0	0	0
CD	0	0	0	0
CD'	1	0	0	1

```
X = B'D'
```