OBJECTIVES:

This problem can be solved by searching for a solution. The initial state is given by the empty chess board. Placing a queen on the board represents an action in the search problem. A goal state is a configuration where none of the queens attacks any of the others. Note that every goal state is reached after exactly 8 action

BACKGROUND STUDY

- Should have prior knowledge on any programming language to implement the algorithm.
- Need knowledge of tree searching algorithm.
- Input an initial state of the board.
- Solving problem by searching algorithm that would be covered in theory class and that knowledge will help you here to get the implementation idea.

RECOMMENDED READING

- http://www.aiai.ed.ac.uk/~gwickler/eightqueens.html
- BOOK
- http://en.wikipedia.org/wiki/Eight_queens_puzzle

8 Queen Problem

The 8-queens problem can be defined as follows: Place 8 queens on an (8 by 8) chess board such that none of the queens attacks any of the others. A configuration of 8 queens on the board is shown in figure 1, but this does not represent a solution as the queen in the first column is on the same diagonal as the queen in the last column.

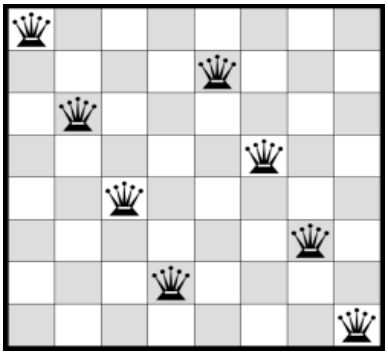

Figure: A solution of 8 queen problem

This formulation as a search problem can be improved when we realize that, in any solution, there must be exactly one queen in each of the columns. Thus, the possible actions can be restricted to placing a

queen in the next column that does not yet contain a queen. This reduces the branching factor from (initially) 64 to 8.

Furthermore, we need only consider those rows in the next column that are not already attacked by a queen that was previously on the board. This is because the placing of further queens on the board can never remove the mutual attack and turn the configuration into a solution. To search for a solution, first select a search strategy. Next, there are some configuration options for the search process. If the search space is to be searched as a graph, multiple paths leading to the same node will usually only be explored once. In a tree, search states that can be reached by multiple paths will also be explored multiple times. However, given our formulation of the search problem, there can only be one path to every state. The number of states to be generated can be limited to the given value, resulting in the search being abandoned at that point. For a depth-first search it is also possible to set a depth limit, meaning no states at a greater depth will be explored.

Finally, a trace of the search can be written to the window and/or a text file. The operation performed by a search engine consists of selecting a current search node and generating its successors, and the trace reflects this process. For example, the line: current state: 11: <State (2qs): 7 3 0 0 0 0 0 0>>

indicates that the selected node contains the given state. In this state 2 queens are on the board. The following numbers then indicate the positions of all queens on the board, starting with the first column. In the example, the first queen is in row 7 (numbering of rows starts from 0 here), and the second queen is in row 3. The remaining zeros can be ignored as there are only two queens on the board.

The lines following this one in the trace describe the successor states that have been generated, for example:

successor state: 59: <State (3qs): 7 3 6 0 0 0 0 0>

successor state: 60: <State (3qs): 7 3 1 0 0 0 0 0>

successor state: 61: <State (3qs): 7 3 0 0 0 0 0 0>

So, expanding the given search node (11) resulted in three new search nodes (59, 60 and 61).

Even with tracing off the window will display some information about the search (which is not part of the trace). It will show how many states have been explored (the goal test has been performed and successors have been generated) and how many states have been generated (explored states plus fringe nodes). If a solution is found this will also be printed. Finally, the elapsed time taken to perform the search is printed. Note that writing the trace usually takes more time than searching itself.

Algorithm:

- Start with one queen at the first column first row

- Continue with second queen from the second column first row

- Go up until find a permissible situation

- Continue with next queen

To backtrack:

Remember that we used backtrack when we cannot find admissible position for a queen in a column. Otherwise we go further with the next column until we place a queen on the last column. Therefore your code must have fragment:

```
int PlaceQueen(int board[8], int row)

If (Can place queen on ith column)

 PlaceQueen(newboard, 0)

Else

 PlaceQueen(oldboard,oldplace+1)

End .
```

LABROTORY EXERCISES

1. Implement n-queen algorithm using the pseudocode described above in your preferred language.
   Input: a 8x8 matrix
   Output: possible states of the 8 queen positions in the matrix without conflicting each other.