**ASSIGNMENT 3**

**Title:** Write a recursive program to find the solution of placing n queens on the chessboard so that no two queens attack each other using Backtracking.

**Software Requirement:** Ubuntu, C++ Compiler

**Theory:**

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time. Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

**1.Backtracking**

Backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output. The Brute force approach tries out all the possible solutions and chooses the desired/best solutions. The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.This approach is used to solve problems that have multiple solutions.

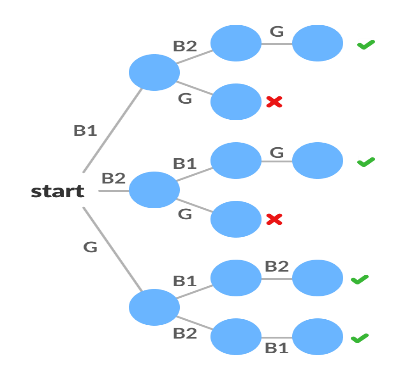
There are three types of problems in backtracking :

* Decision Problem – In this, we search for a feasible solution.
* Optimization Problem – In this, we search for the best solution.
* Enumeration Problem – In this, we find all feasible solutions.

**Example Backtracking Approach**

* Problem: You want to find all the possible ways of arranging 2 boys and 1 girl on 3 benches. Constraint: Girl should not be on the middle bench.
* Solution: There are a total of 3! = 6 possibilities. We will try all the possibilities and get the possible solutions. We recursively try all the possibilities.

The following state space tree shows the possible solutions.



**Steps to follow in Backtracking:**

We can consider backtracking process as finding a particular leaf in the tree.

* If node N is goal node, then return success and exit.
* If node N is leaf node but node is not a goal node then returns failure.
* Otherwise, for each child C of node N
  + Explore child node C
  + If C is goal node, return success
  + Else return failure

Draw an abstract tree. The tree is typically search depth first and the nodes are implicit meaning they are generated as need. A node is said to be promising if the partial solution is still feasible. Any time the partial node becomes infeasible the node, called non-promising the branch will no longer be pursued. So, the algorithm backtracks to the first promising node and explores the other branches of the state-space tree.

**Algorithm: Backtracking**

Algorithm Backtrack(k)

//X[1….k-1] is the solution vector

//T(x[1],x[2],….x[k-1] is the state space tree

//Bk( ) is the bounding function

for each x[k] ∈ T[x[1],x[2],…..x[k-1]] do

if (Bk(x[1],x[2],…..,x[k-1])) == TRUE then

if(x[1], x[2],….,x[k]) is path to answer node then

print(x[1],x[2],…x[k])

end

if k< n then

Backtrack(k+1)

end

end

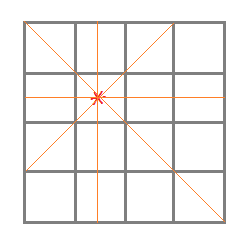
end

**2.N Queen’s Problem:**

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. In chess, a queen can attack horizontally, vertically, and diagonally. No two queens are on the same row, column, or diagonal.

Mathematically, It is given as, If 2 queens are placed at position (i, j) and (k, l) where I and k are the row indices and j and i are the column indices then,

* i != k (No same Row)
* j != l (No same Column)
* |i-k| != |j – l| (No same Diagonal)



**N Queens Problem: General Procedure**

* Identification of Data Structures
* Identification of explicit Constraints
* Identification of implicit Constraints
* Building a solution using backtracking
* Building the state space tree

**N Queens Algorithm**

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row

then mark this [row, column] as part of the

solution and recursively check if placing

queen here leads to a solution.

b) If placing the queen in [row, column] leads to

a solution then return true.

c) If placing queen doesn't lead to a solution then

unmark this [row, column] (Backtrack) and go to

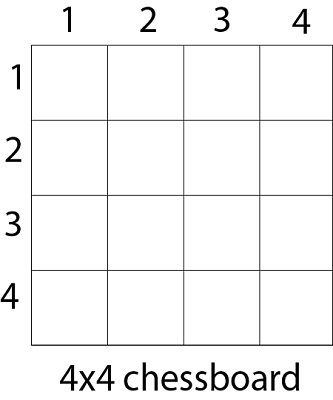
step (a) to try other rows.

4) If all rows have been tried and nothing worked,

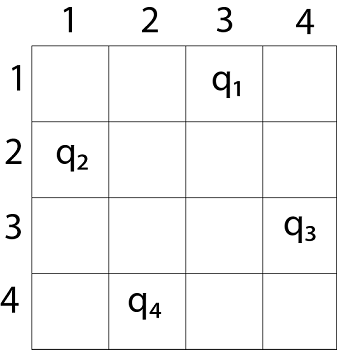
return false to trigger backtracking.

**N Queen’s Problem: Example**

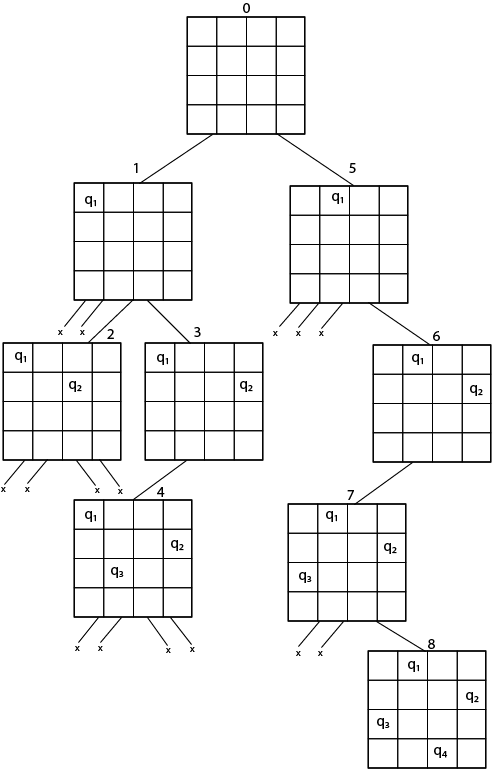
N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal. It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first consider the 4 queens problem and then generate it to n - queens problem. Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



Since, we have to place 4 queens such as q1 q2 q3 and q4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."



The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:



We can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.

**3.Complexity Analysis**

* For finding a single solution where the first queen Q has been assigned the first column and can be put on N positions, the second queen has been assigned the second column and would choose from N-1 possible positions and so on; the time complexity is O ( N ) \* ( N - 1 ) \* ( N - 2 ) \* … 1 ). i.e The worst-case time complexity is O ( N! ).
* Thus, for finding all the solutions to the N Queens problem the time complexity runs in polynomial time.

**Conclusion:**

N queen’s problem using Backtracking is implemented successfully.