Department of
Computer Science and Engineering

Title: Solve N-Queen Problem Using Backtracking Algorithm

Artificial Intelligence Lab

CSE 404

Green University of Bangladesh

1 Objective(s)

- To understand how backtrack algorithm performs to solve constraint satisfaction problem
- To understand how N-queen problem is solvable by using backtrack method.

2 Problem analysis

The N queens puzzle is the problem of placing eight chess queens on an N×N chessboard or grid so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. For example, following figure 1 is a solution for 6 Queen problem. 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 we will consider the 4 queens problem and then generate it to n - queens problem.

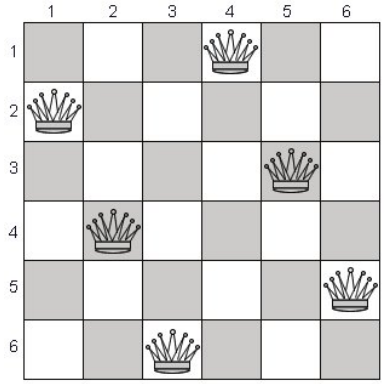

Figure 1: A sample solution of placing 6 queens in a grid

3 Algorithm

Step 1: Start in the leftmost column

Step 2: If all queens are placed
return true

Step 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.

Step 4: If all rows have been tried and nothing worked, return false to trigger backtracking.

3.1 Example

Let's understand an example by taking 4 queens to place which are 'q1', 'q2', 'q3' and 'q4'. Now, we place queen 'q1' in the very first acceptable position (1, 1). Next, we put queen 'q2' so that both these queens do not attack each other. We find that if we place 'q2' in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for 'q2' in column 3, i.e. (2, 3) but then no position is left for placing queen 'q3' safely. So we backtrack one step and place the queen 'q2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q4' can be placed safely. Then we have to backtrack till 'q1' and place it to (1, 2) and then all other queens are placed safely by moving 'q2' to (2, 4), 'q3' to (3, 1) and 'q4' to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the

4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

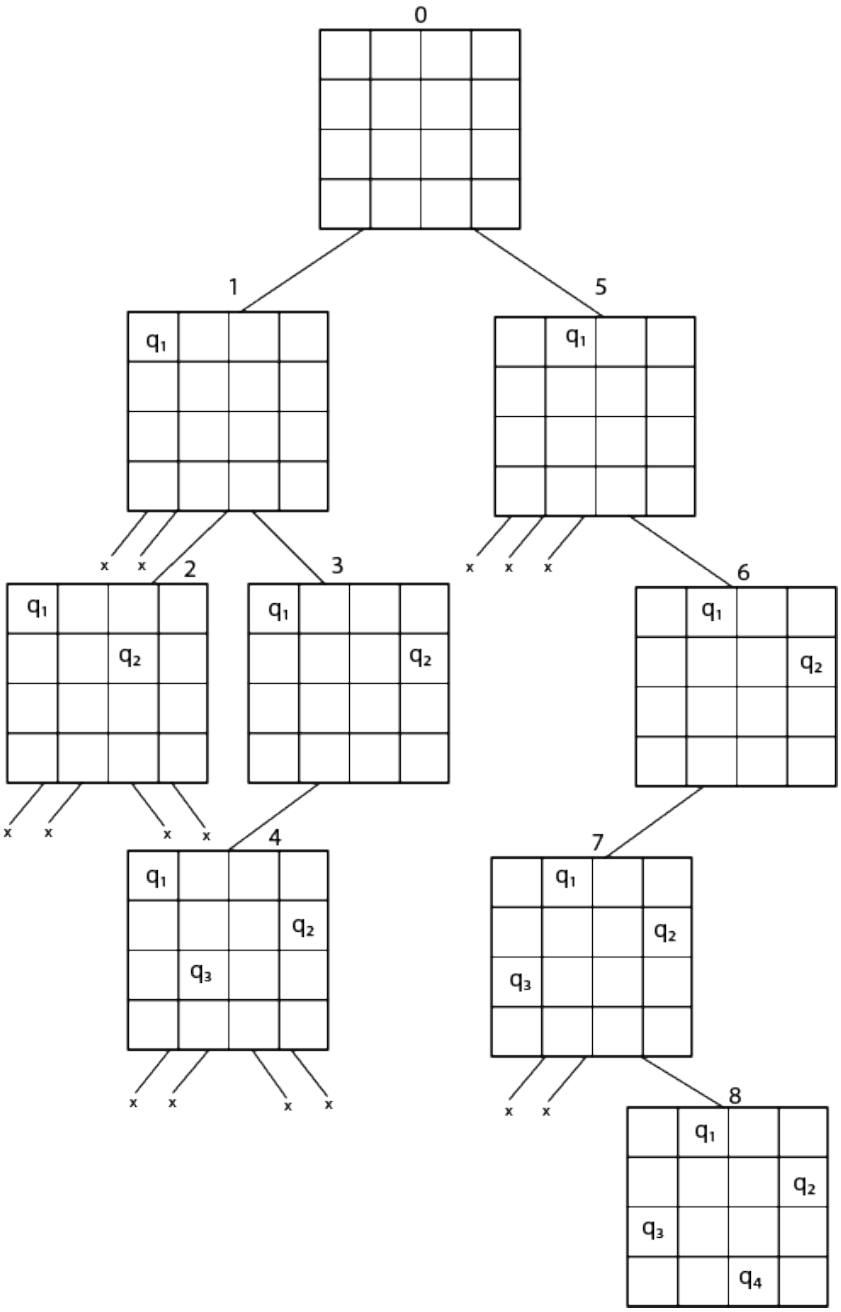

Figure 2: Backtracking steps for 4 queen placement

4 Implementation in Java

```
1  package n_queen;
2
3  import java.util.Scanner;
4
5  /* Java program to solve N Queen Problem using backtracking */
6  public class N_queen {
7
```

```
8       int N;
9       N_queen(int a)
10      {
11          N = a;
12      }
13      /* A utility function to print solution */
14      void printSolution(int board[][]) {
15          for (int i = 0; i < N; i++) {
16              for (int j = 0; j < N; j++) {
17                  System.out.print(" " + board[i][j]+ " ");
18              }
19              System.out.println();
20          }
21      }
22      /* A utility function to check if a queen can
23    be placed on board[row][col]. Note that this
24    function is called when "col" queens are already
25    placeed in columns from 0 to col -1. So we need
26    to check only left side for attacking queens */
27      boolean isSafe(int grid[][], int row, int col) {
28          int i, j;
29
30          /* Check this row on left side */
31          for (i = 0; i < col; i++) {
32              if (grid[row][i] == 1) {
33                  return false;
34              }
35          }
36
37          /* Check upper diagonal on left side */
38          for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
39              if (grid[i][j] == 1) {
40                  return false;
41              }
42          }
43          /* Check lower diagonal on left side */
44          for (i = row, j = col; j >= 0 && i < N; i++, j--) {
45              if (grid[i][j] == 1) {
46                  return false;
47              }
48          }
49          return true;
50      }
51
52      /* A recursive utility function to solve N Queen problem */
53      boolean solveNQUtil(int grid[][], int col) {
54          /* base case: If all queens are placed then return true */
55          if (col >= N) {
56              return true;
57          }
58
59          /* Consider this column and try placing this queen in all rows one by
                one */
60          for (int i = 0; i < N; i++) {
61              /* Check if the queen can be placed on board[i][col] */
62              if (isSafe(grid, i, col)) {
63                  /* Place this queen in board[i][col] */
64                  grid[i][col] = 1;
```

```
                /* recur to place rest of the queens */
                if (solveNQUtil(grid, col + 1) == true) {
                    return true;
                }
                /* If placing queen in board[i][col] doesn't lead to a solution
                   then remove queen from board[i][col] */
                grid[i][col] = 0; // BACKTRACK
            }
        }
        /* If the queen can not be placed in any row in this colum col, then
           return false */
        return false;
    }

    boolean solveNQ() {
        int grid[][] = new int[N][N]; // create N*N grid and initialized to 0
        if (solveNQUtil(grid, 0) == false) {
            System.out.print("Solution does not exist for "+N+" queens");
            return false;
        }
        System.out.println("Solution found for "+N+" queens");
        printSolution(grid);
        return true;
    }

    public static void main(String args[]) {
        int n;
        Scanner sc = new Scanner(System.in);
        System.out.println("Number of queen to place - ");
        n = sc.nextInt();
        N_queen Queen = new N_queen(n);
        Queen.solveNQ();
    }
}
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Input:
Number of queen to place -
7
Output:
Solution found for 7 queens
1 0 0 0 0 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 0
0 0 0 0 0 1 0
0 0 1 0 0 0 0
0 0 0 0 0 0 1
0 0 0 1 0 0 0

6 Implementation in Python

```
class NQueen:
    def __init__(self, a):
        self.N = a
```

```
4
5       def print_solution(self, board):
6           for i in range(self.N):
7               for j in range(self.N):
8                   print(" " + str(board[i][j]) + " ", end="")
9               print()
10
11      def is_safe(self, grid, row, col):
12          for i in range(col):
13              if grid[row][i] == 1:
14                  return False
15
16          for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
17              if grid[i][j] == 1:
18                  return False
19
20          for i, j in zip(range(row, self.N), range(col, -1, -1)):
21              if grid[i][j] == 1:
22                  return False
23
24          return True
25
26      def solve_nq_util(self, grid, col):
27          if col >= self.N:
28              return True
29
30          for i in range(self.N):
31              if self.is_safe(grid, i, col):
32                  grid[i][col] = 1
33
34                  if self.solve_nq_util(grid, col + 1):
35                      return True
36
37                  grid[i][col] = 0
38
39          return False
40
41      def solve_nq(self):
42          grid = [[0] * self.N for _ in range(self.N)]
43
44          if not self.solve_nq_util(grid, 0):
45              print("Solution does not exist for", self.N, "queens")
46              return False
47
48          print("Solution found for", self.N, "queens")
49          self.print_solution(grid)
50          return True
51
52
53  def main():
54      n = int(input("Number of queens to place: "))
55      queen = NQueen(n)
56      queen.solve_nq()
57
58
59  if __name__ == "__main__":
60      main()
```

7 Sample Input/Output (Compilation, Debugging & Testing)

Input:
Number of queen to place -
7
Output:
Solution found for 7 queens
1 0 0 0 0 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 0
0 0 0 0 0 1 0
0 0 1 0 0 0 0
0 0 0 0 0 0 1
0 0 0 1 0 0 0

8 Discussion & Conclusion

Based on the focused objective(s) to understand the constraint satisfaction problem N-queen using backtracking method, the additional lab exercise will increase confidence towards the fulfilment of the objectives(s).

9 Lab Exercise (Submit as a report)

- The N-queens puzzle is the problem of placing N queens on a (N×N) chessboard such that no two queens can attack each other. Find all distinct solution to the N-queen problem.
  - Hint: For N = 4 there are two possible solutions -

0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0

Solution 1

0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

Solution 2

10 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected for lab exercise.