

LAPORAN TUGAS
DESIGN AND ANALYSIS OF ALGORITHM
TUGAS 6



Kelas : D

05111840000058

Rafif Ridho

05111840000153

Vieri Fath Ayuba

Dosen :

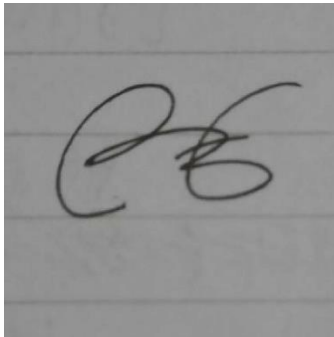
Misbakhul Munir Irfan Subakti, S.Kom,M.Sc

Departement of Informatics
Faculty Of Intelligent Electrical And Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya
2020

Pledge and Declaration

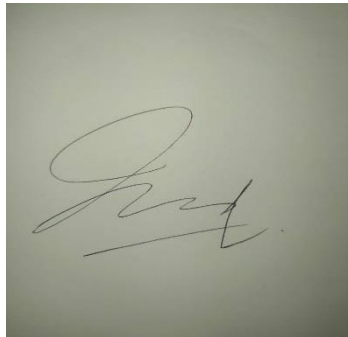
"By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved quiz 2 by myself, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have been done any cheating and/or plagiarism."

Surabaya , 25 March 2020

A handwritten signature in black ink on a light-colored background. The signature is stylized, starting with a large 'R' and ending with a horizontal stroke.

Rafif Ridho

05111840000058

A handwritten signature in black ink on a light-colored background. The signature is stylized, starting with a large 'V' and ending with a horizontal stroke.

Vieri Fath Ayuba

05111840000153

8 Queen Problem

INTRODUCTION

The N-Queens problem is a classic problem that is often used in discussions of various search strategies. The problem is often defined in terms of a standard 8-by-8 chess board, although it can be defined for any N-by-N board and is solvable for N ≥ 4 .

In general, the problem is to place the queens on the board so that no two queens can attack each other. The method for solving the problem depends on the knowledge of the rules for moving a queen in the game of chess. Specifically, a queen can move in any straight line, either along a row, along a column, or a diagonal. In an N-by-N board, each queen will be located on exactly one row, one column, and two diagonals.

The requirement that no two queens be placed in the same row restricts the number of queens that can be placed on an N-by-N board to N. Thus, a standard 8-by-8 chess board can take at most eight queens. The problem is to find an arrangement that allows placement of N queens.

DFS or Depth First Search is a search algorithm that search from tree root to sub tree in search space, recursion from sub tree when reach to non promise state or stop search when find goal.

CODE IMPLEMENTATION

A. Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

Step by Step :

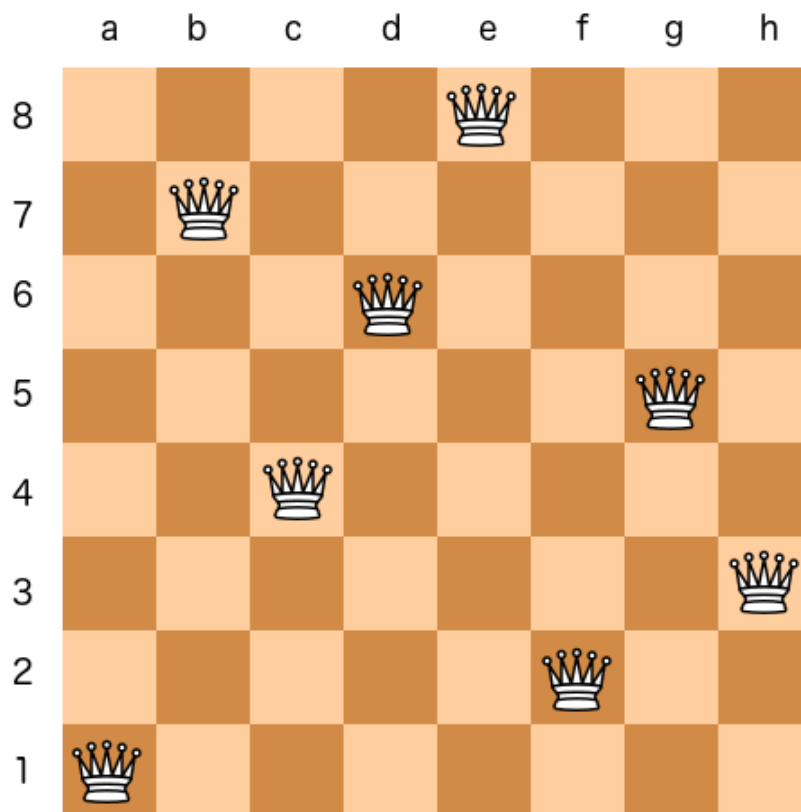
- 1) Start in the leftmost column
- 2) If all queens are placed
return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row
then mark this [row, column] as part of the
solution and recursively check if placing
queen here leads to a solution.
 - b) If placing 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.
- 3) If all rows have been tried and nothing worked,
return false to trigger backtracking.

B. Solution

The eight queens puzzle has 92 distinct solutions. If solutions that differ only by the symmetry operations of rotation and reflection of the board are counted as one, the puzzle has 12 solutions. These are called fundamental solutions; representatives of each are shown below.

A fundamental solution usually has eight variants (including its original form) obtained by rotating 90, 180, or 270° and then reflecting each of the four rotational variants in a mirror in a fixed position. However, should a solution be equivalent to its own 90° rotation (as happens to one solution with five queens on a 5×5 board), that fundamental solution will have only two variants (itself and its reflection). Should a solution be equivalent to its own 180° rotation (but not to its 90° rotation), it will have four variants (itself and its reflection, its 90° rotation and the reflection of that). If $n > 1$, it is not possible for a solution to be equivalent to its own reflection because that would require two queens to be facing each other. Of the 12 fundamental solutions to the problem with eight queens on an 8×8 board, exactly one (solution 12 below) is equal to its own 180° rotation, and none is equal to its 90° rotation; thus, the number of distinct solutions is $11 \times 8 + 1 \times 4 = 92$.

All fundamental solutions are presented below:



SOURCE CODE

```
#include<bits/stdc++.h>
#define N 8

void printSolution(int board[N][N])
{
    static int k = 1;
    printf("%d-\n",k++);
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}
```

```

bool solveNQUtil(int board[N][N], int col)
{
    if (col == N)
    {
        printSolution(board);
        return true;
    }
    bool res = false;
    for (int i = 0; i < N; i++)
    {

        if ( isSafe(board, i, col) )
        {

            board[i][col] = 1;

            res = solveNQUtil(board, col + 1) || res;

            board[i][col] = 0;
        }
    }
    return res;
}

void solveNQ()
{
    int board[N][N];
    memset(board, 0, sizeof(board));

    if (solveNQUtil(board, 0) == false)
    {
        printf("Solution does not exist");
        return ;
    }

    return ;
}

int main()
{
    solveNQ();
}

```

Analysis

- Void printSolution(int board[N][N])

This function is used to print all possible solution in a shape of a board

- Bool isSafe(int board[N][N], int row, int col)

This function is created to check whether a queen can be placed on board[row][col], the function is called when “col” queens are already placed in columns from 0 to col -1, so we only need to check left side for attacking queens

- Bool solveNQUtil(int board[N][N], int col)

This function is used to solve 8 Queen problem recursively , the base case of this recursive function is if all queen are placed then return true, then we start from column 0 and try placing queen in all rows one by one, in each row the function will call isSafe to check whether the queen can be placed there then recur to place the rest of the queens else if the queen is inside the attack range of another queen then remove the queen from the board then backtrack, if the queen cant be place in any row in this column then it will return false

- Bool solveNQ()

This function will make a board 8x8 with each value of the array is 0 then it will solve 8 queen problem using backtracking, it mostly uses solveNQUtil() function to solve the problem, it will return false if there is no possible solution otherwise it will return true and print all possible solution

- Int main()

In main we only call solveNQ() function which will run all the other function

Output

```
D:\File Kuliah\PAA\8queen.exe
1 0 0 0 0 0 0 0

Solution 90
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
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 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0

Solution 91
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0

Solution 92
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
```

The program found in total 92 possible solution, in the picture above we only cropped the latest solution which is proof of how many solutions are there for 8 queens.

Link to github :

https://github.com/vierifath/IF184401_DAA_D_Q2_Group10