Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Mini Project Report

On

"N-Queen Problem"

[Course Code: COMP 314]

Submitted by:

Nabin Ghimire (Roll No. 17)

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

10th April, 2022

Acknowledgement

I would like to express my sincere appreciation to my teacher Rajani Chulyadyo (PhD), for her appreciable guidance, encouragement, valuable inputs and productive criticism throughout the course of COMP 314 and this mini project would not have been possible without her.

I would like to thank Department of Computer Science and Engineering (DoCSE) for providing me a chance to work on the project.

Sincerely,

Nabin Ghimire (17)

Abstract

Among the eighteen mini projects assigned to us, I was assigned to implement N-Queen problem and finding the maximum size of chessboard that can be solved by the computer using backtracking. The implementation of project was done in Python3 using Visual Studio Code. The time complexity for each method was calculated and was found to be convincing. This project has helped me to develop a deep understanding in the backtracking by making me go through different implementations and developing the algorithm.

Keywords: *backtracking, N-Queen, Python, Visual Studio Code, Algorithms.*

Table of Contents

Acknowledgement i

Abstract ii

Chapter 1: Introduction 1

1.1. Backtracking 1

1.2. N-Queens Problem 2

1.3. Objectives 4

1.4. Motivation and Significance 4

Chapter 2: Procedure and Method 5

2.1 Procedure 5

2.1.1 Array 5

2.2.2 Stack 5

2.2 Tools Used 5

Chapter 3: Discussion 6

3.1. Array 6

3.2. Stack 6

Machine Performance 6

Time Complexity 7

Chapter 4: Conclusion 7

Appendix 1 8

Appendix 2 9

Chapter 1: Introduction

1.1. Backtracking

Backtracking can be defined as general algorithmic technique that consider searching every possible combination in order to solve a computational problem.

It uses the Brute force search to solve the problem, and the brute force search says that for the given problem, we try to make all the possible solutions and pick out the best solution from all the desired solutions. This rule is also followed in dynamic programming, but dynamic programming is used for solving optimization problems. In contrast, backtracking is not used in solving optimization problems. Backtracking is used when we have multiple solutions, and we require all those solutions.

Backtracking name itself suggests that we are going back and coming forward; if it satisfies the condition, then return success, else we go back again. It is used to solve a problem in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criteria.

The major advantage of the backtracking algorithm is the ability to find and count all the possible solutions rather than just one while offering decent speed. In fact this is the reason it is so widely used.

When we have multiple choices, then we make the decisions from the available choices. In the following cases, we need to use the backtracking algorithm:

- A piece of sufficient information is not available to make the best choice, so we use the backtracking strategy to try out all the possible solutions.

- Each decision leads to a new set of choices. Then again, we backtrack to make new decisions. In this case, we need to use the backtracking strategy.

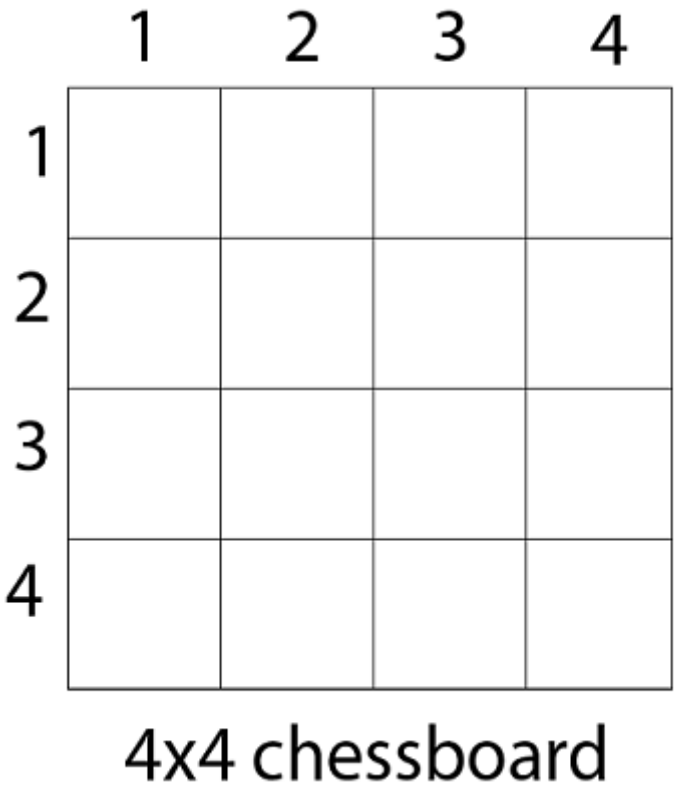
1.2. N-Queens Problem

The n-queens problem is about finding how many different ways queens can be placed on a chessboard so that none attack each other.

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.

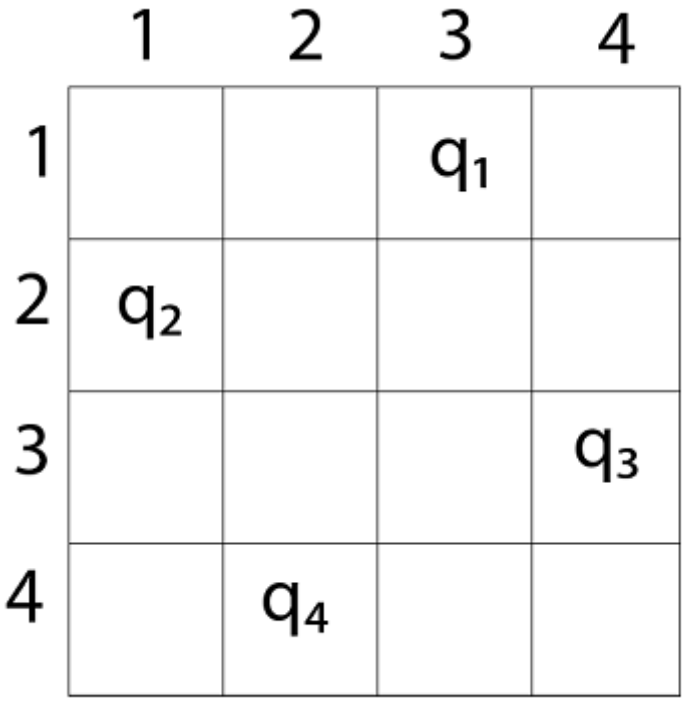
It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



4x4 chessboard

Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 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.



1.3. Objectives

Being a computer engineer, we must be able to code every possible concepts as far as possible to understand the concept more precisely. This mini-project helps apply the knowledge of algorithm course and use those concept solve the problem in python. As for this project, the objectives are:

1. To understand the concept of backtracking and its application.
2. To better understand one of the application i.e (N-Queens problem) of Backtracking,
3. To find out all the possible solution how many different ways queens can be placed on a chessboard so that none attack each other.
4. To determine how much my machine can evaluate the result of various board size.

1.4. Motivation and Significance

As we were assigned to choose a mini project I was puzzled regarding what to choose. I was told to select three project topic preferences. And I was happy to be doing the project no 6 as my mini project.

As the world is getting digitalized, we as developers need to enhance our skills in the field of computer and technology and this has become one of our motivations. Completion of project for the partial fulfillment of IIIrd year IInd semester mini project is also one of the motivations.

Chapter 2: Procedure and Method

2.1 Procedure

I implemented the program to calculate all the optimal Solution of the n-queens using the Python language making the classes. The concept of the Stack and the array is used in the program.

2.1.1 Array

I have maintained the array inside of the queue in order to store the all the possible optimal solution for the board of size N*N where we can keep the N queens.

2.2.2 Stack

We will store an array inside a stack and the solution will be popped out of the stack and all the optimal solution will be printed.

2.2 Tools Used

Text Editor: VS Code

Compiler: Python

Version control: git and GitHub

Chapter 3: Discussion

3.1. Array

Array is a data structure consisting of collection of elements each identified by an index. The memory address of the first element of an array is called first address or foundation address. Arrays are among the oldest and most important data structures, and are used by almost every program. They are also used to implement many other data structures, such as lists, strings and Queue.

3.2. Stack

Stack is a linear Data Structure which follow particular order in which operations are performed. The Order may be LIFO (Last In Last Out) or FILO (First In Last Out).

Machine Performance

As, I am finding all the optimal solution for the given Board of N*N where we are putting N queens such that no queens attack each other.

Doing the code, I came to find that my machine can find the solutions up to N=13. After that my Computer got hang and cannot find the further solution.

```
C:\Users\dell\Desktop\N-Queens-Problem>python main.py
N-Queens Problem :
Please enter the size of board: 13
Do you want the solutions to be printed (Y/N): n
Total BFS solutions: 73712
```

Time Complexity

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**.

Chapter 4: Conclusion

In this way, I completed my mini project and learnt about the backtracking in detail. Also, the concept of the n-queens became clear to me. Also, the conclusion about the machine performance of given code was evaluated according to my computer. Also, I studied about its Time complexity going through the various sites and concluded the mini project.

Appendix 1

Table 1: Gantt chart for our project

Tasks	Weeks					
	1	2	3	4	5	6
Planning						
Research Work						
Work thinking						
Coding						
Implementations and Merging						
Debugging						
Testing						
Documentation						

Fig 2: Gantt chart

Appendix 2

Queen.py

```python
from queue import Queue


class NQueens:
    def __init__(self, size):
        self.size = size

    def solve_dfs(self):
        if self.size < 1:
            return []
        solutions = []
        stack = [[]]
        while stack:
            solution = stack.pop()
            if self.conflict(solution):
                continue
            row = len(solution)
            if row == self.size:
                solutions.append(solution)
                continue
            for col in range(self.size):
                queen = (row, col)
                queens = solution.copy()
                queens.append(queen)
                stack.append(queens)
        return solutions
    # function to check if the position is available or there is conflict

    def conflict(self, queens):
        for i in range(1, len(queens)):
            for j in range(0, i):
                a, b = queens[i]
                c, d = queens[j]
                if a == c or b == d or abs(a - c) == abs(b - d):
                    return True
        return False

    def print(self, queens):
        for i in range(self.size):
            print(' ---' * self.size)
            for j in range(self.size):
                p = 'Q' if (i, j) in queens else ' '
                print('| %s ' % p, end='')
            print('|')
        print(' ---' * self.size)
```

Main.py

```python
from queen import NQueens


def main():
    print('N-Queens Problem :')
    size = int(input('Please enter the size of board: '))
    print_solutions = input(
        'Do you want the solutions to be printed (Y/N): ').lower() == 'y'
    n_queens = NQueens(size)
    dfs_solutions = n_queens.solve_dfs()
    if print_solutions:
        for i, solution in enumerate(dfs_solutions):
            print('DFS Solution %d:' % (i + 1))
            n_queens.print(solution)
    print('Total DFS solutions: %d' % len(dfs_solutions))


if __name__ == '__main__':
    main()
```

Output

```
C:\Users\dell\Desktop\N-Queens-Problem>python main.py
N-Queens Problem :
Please enter the size of board: 4
Do you want the solutions to be printed (Y/N): y
DFS Solution 1:
 --- --- --- ---
|   |   | Q |   |
 --- --- --- ---
| Q |   |   |   |
 --- --- --- ---
|   |   |   | Q |
 --- --- --- ---
|   | Q |   |   |
 --- --- --- ---
DFS Solution 2:
 --- --- --- ---
|   | Q |   |   |
 --- --- --- ---
|   |   |   | Q |
 --- --- --- ---
| Q |   |   |   |
 --- --- --- ---
|   |   | Q |   |
 --- --- --- ---
Total DFS solutions: 2
```

```
C:\Users\dell\Desktop\N-Queens-Problem>python main.py
N-Queens Problem :
Please enter the size of board: 5
Do you want the solutions to be printed (Y/N): n
Total DFS solutions: 10
```