AIML Lab 4

U18CO021: SAHIL BONDRE

Question:

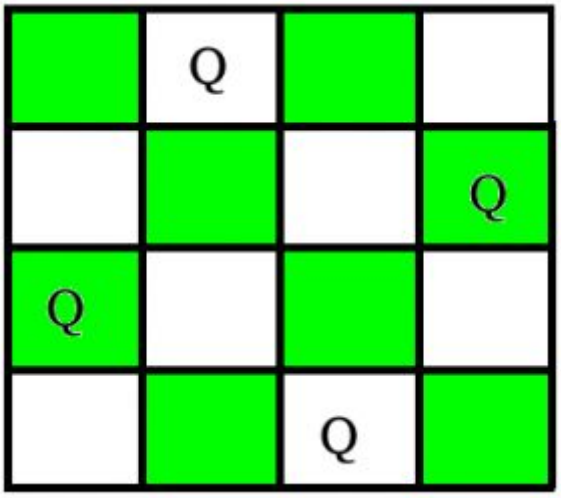
Implement N queens problem using below algorithms in prolog. Compare the complexity of both algorithms. Which algorithm is best suited for implementing N queens problem and why ?

1. Breadth First Search

2. Depth First Search

Solution:

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.



This problem can be solved using various techniques like BFS, DFS, Backtracking etc.

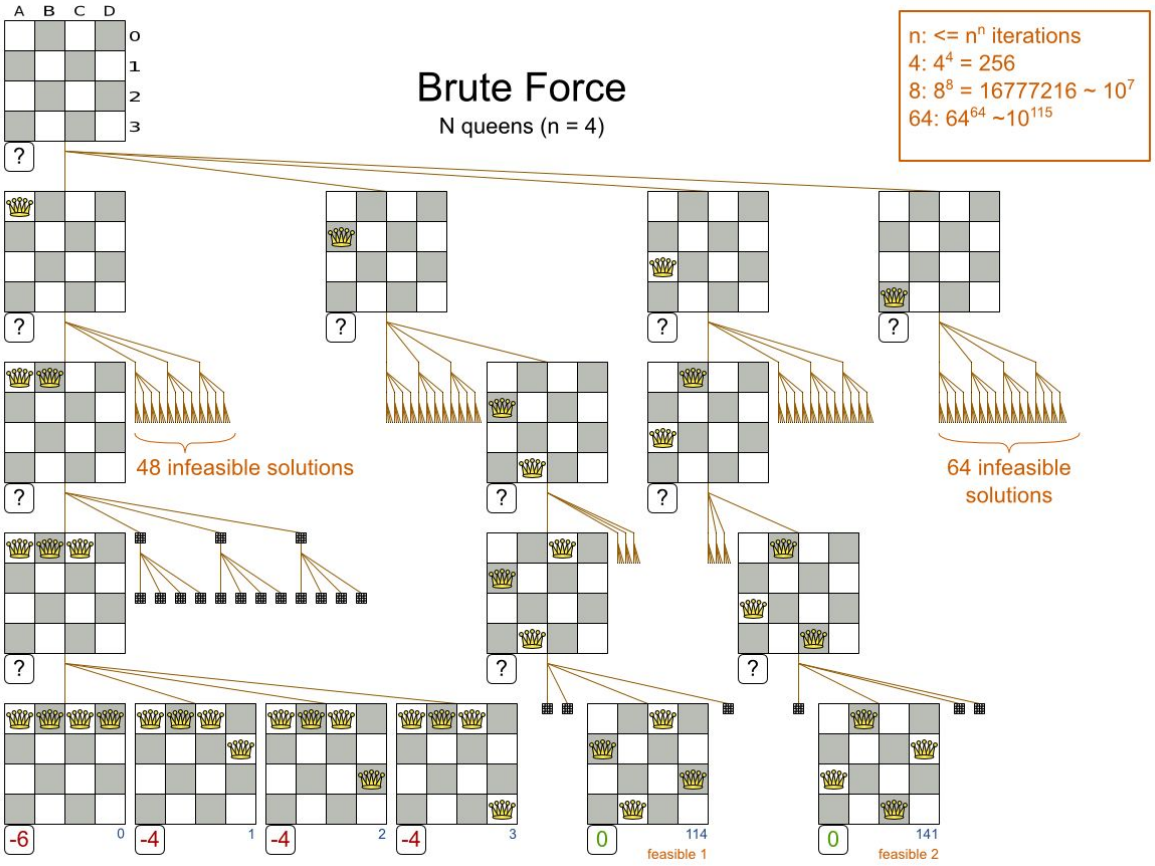
BFS takes O(n^n) time to solve as it tries every possible solution

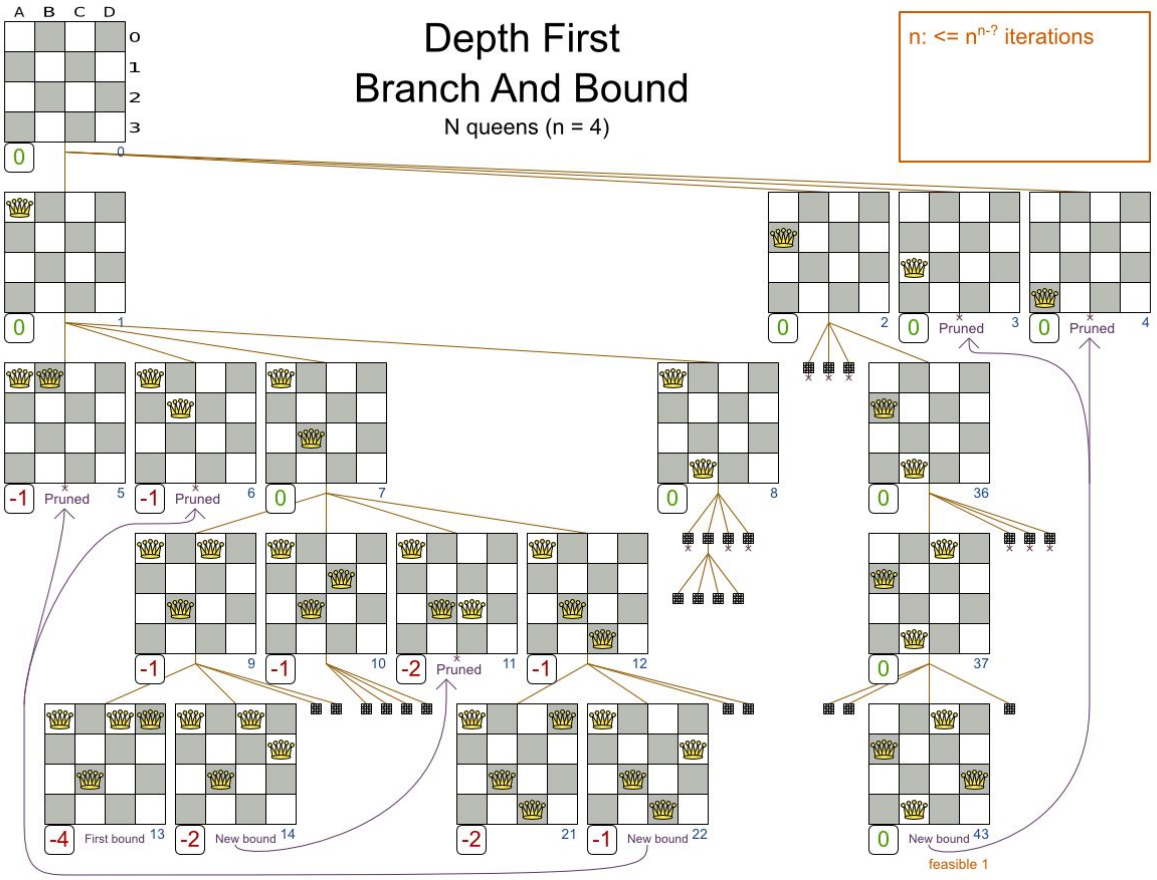
DFS takes O(n!) time as it discards the invalid solutions and their following recursive calls as and when they are found

Backtracking solves this problem in O(n!) time

Illustrations of BFS and DFS approach:

(https://stackoverflow.com/questions/52089745/can-we-solve-4-queen-problem-using-best-first-search)





board.py:

```
class Board:
    SIZE = 8

    def __init__(self, state = None):
        if state is None:
            state = [0] * self.SIZE
        # 0 implies not set
        assert len(state) == self.SIZE
        self.state = state

    def __eq__(self, other):
        return self.state == other.state

    def is_full(self):
        for i in self.state:
            if i == 0:
                return False
        return True
```

```
    def is_legal(self):
        for i in range(self.SIZE):
            for j in range(i + 1, self.SIZE):
                if self.state[i] != 0 and self.state[j] != 0 and
self.state[i] == self.state[j]:
                    return False
                if self.state[i] != 0 and self.state[j] != 0 and j - i
== abs(self.state[i] - self.state[j]):
                    return False

        return True

    def is_solved(self):
        if not self.is_full():
            return False

        # no zeros


        # no same columns
        if len(self.state) != len(set(self.state)):
            return False

        # no same diagonals
        for i in range(self.SIZE):
            for j in range(i + 1, self.SIZE):
                if j - i == abs(self.state[i] - self.state[j]):
                    return False

        return True

    def print(self):
        for i in range(self.SIZE):
            row = ""
            for j in range(self.SIZE):
                if self.state[i] - 1 == j:
                    row += "Q "
                else:
                    row += "* "
            print(row)
        print()

    def add_queen(self, num):
        new_state = list.copy(self.state)
        for i in range(self.SIZE):
```

```
            if self.state[i] == 0:
                new_state[i] = num
                break
        return Board(new_state)
```

main.py:

```
from board import Board
from typing import Deque
import time
from collections import deque

board = Board()


def dfs():

    stack: Deque[Board] = deque()
    stack.append(board)
    max_size = len(stack)

    while stack:
        node = stack.pop()

        if node.is_solved():
            print("Found!")
            node.print()
            return max_size

        for i in range(Board.SIZE):
            new = node.add_queen(i + 1)
            if new.is_legal():
                max_size = max(max_size, len(stack))
                stack.append(new)

    print("Not Found!")
    return max_size


def bfs():

    queue: Deque[Board] = deque()
    queue.append(board)
    max_size = len(queue)
```

```
    while queue:
        node = queue.popleft()

        if node.is_solved():
            print("Found!")
            node.print()
            break

        for i in range(Board.SIZE):
            new = node.add_queen(i + 1)
            if new.is_legal():
                max_size = max(max_size, len(queue))
                queue.append(new)

    print("Not Found!")
    return max_size


print("DFS:")
start = time.time()
size = dfs()
print(f"Time taken: {time.time() - start}")
print(f"Memory taken: {size}")
print()
print("BFS:")
start = time.time()
size = bfs()
print(f"Time taken: {time.time() - start}")
print(f"Memory taken: {size}")
```

```
➜  lab-4 git:(master) ✗ python main.py
DFS:
Found!
* * * * * * * Q
* * * Q * * * *
Q * * * * * * *
* * Q * * * * *
* * * * * Q * *
* Q * * * * * *
* * * * * * Q *
* * * * Q * * *

Time taken: 0.010920286178588867
Memory taken: 18

BFS:
Found!
Q * * * * * * *
* * * * Q * * *
* * * * * * * Q
* * * * * Q * *
* * Q * * * * *
* * * * * * Q *
* Q * * * * * *
* * * Q * * * *

Not Found!
Time taken: 0.15296626091003418
Memory taken: 572
➜  lab-4 git:(master) ✗
```