Assignment 1

Namn: Adnan Altukleh
Kurs: DV2619
Datum: 2022-09-26

Solve the 8 - queens problem using backtracking algorithm:

To solve the 8 queen problem I used the DFS algorithm which uses a backtracking method.

Depth-first search (DFS) is a searching algorithm for traversing a tree, tree structure, or graph. It starts at the root and explores as far as possible along **each branch before backtracking.**

BFS, Breadth-First Search,Finds a path between two nodes by taking
one step down all paths and then immediately backtracking,
 is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked, then its adjacent vertices are visited and stored in the queue.

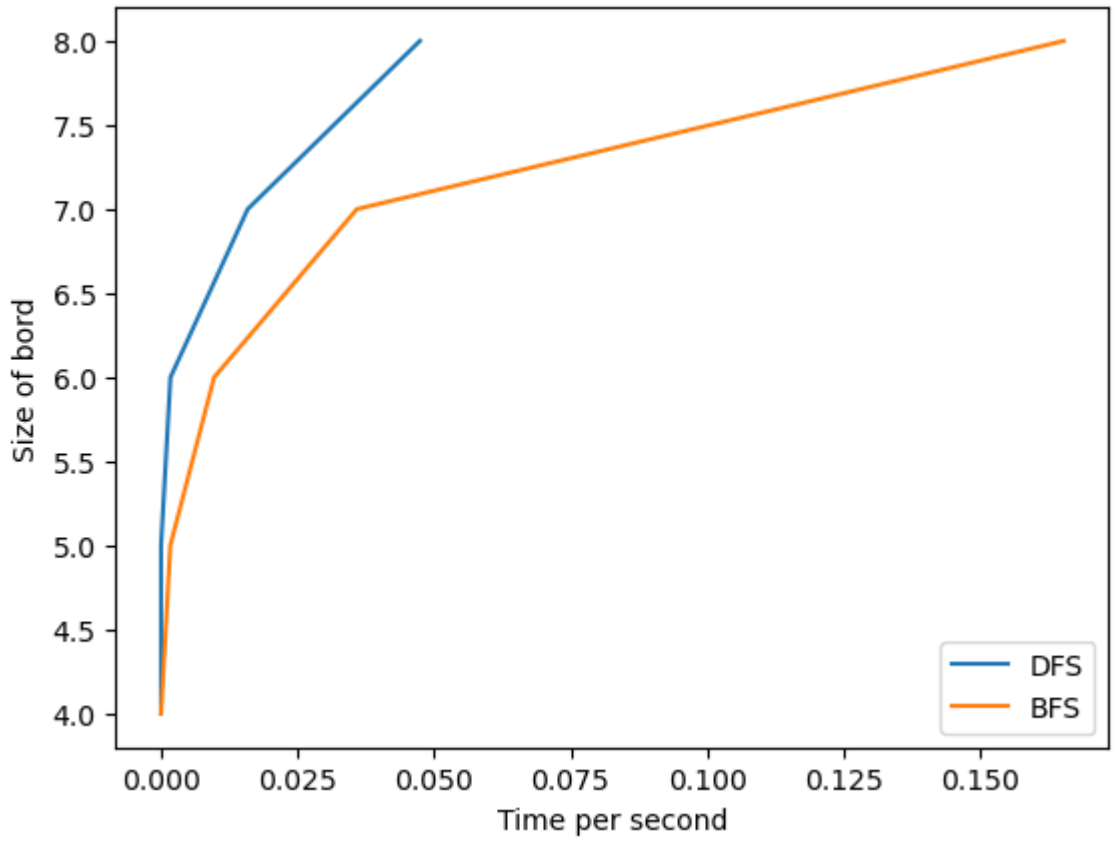
BFS VS DFS Solutions:

```
: N-Queens Problem :. size of board 8x8
Total DFS solutions: 92
 Total DFS time Average for size 8 0.052553796249887215
Total BFS solutions: 92
 Total BFS time Average for size 8 0.1489991208781367
```

We see that both methods have the same quantity of solutions, but DFS was faster than BFS.
NOTE: The Average time calculated as following the sum of the solutions and divided by quantity of the solutions.

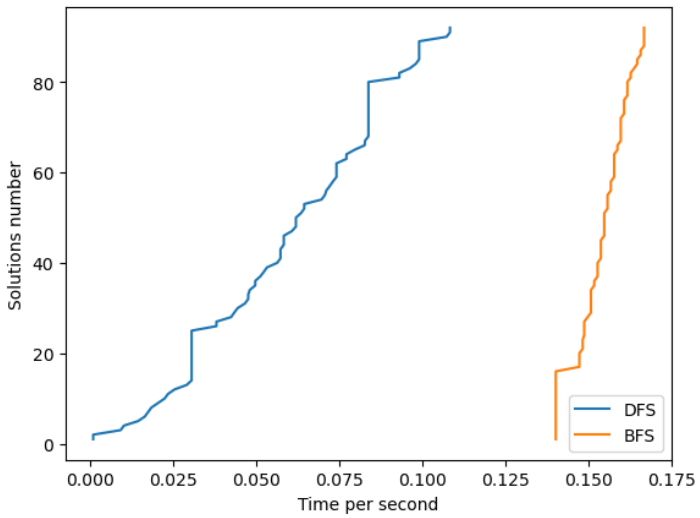
If we compare the both methods with different sizes of boards, **which is faster**?

```
.: N-Queens Problem :.
Total DFS solutions: 2
Total DFS time Average for size 4 0.0
Total BFS solutions: 2
Total BFS time Average for size 4 0.0
Total DFS solutions: 10
Total DFS time Average for size 5 0.0
Total BFS solutions: 10
Total BFS time Average for size 5 0.0016987323760986328
Total DFS solutions: 4
Total DFS time Average for size 6 0.0017293095588684082
Total BFS solutions: 4
Total BFS time Average for size 6 0.009725749492645264
Total DFS solutions: 40
Total DFS time Average for size 7 0.015871876478195192
Total BFS solutions: 40
Total BFS time Average for size 7 0.03586403727531433
Total DFS solutions: 92
Total DFS time Average for size 8 0.04741881982139919
Total BFS solutions: 92
Total BFS time Average for size 8 0.165327497150587
```



`Figur nr 1:` compare the both methods with different sizes of boards

Still the DFS in the lead, DFS is faster than BFS in all cases but **why**?
for that we have to look a little closer to the solutions



`Figure nr 2:` showing the time that 92 solutions took for board size 8x8

We can see that DFS is still running faster, but we can also see that DFS takes more time in the last solutions than it tooks in the beginning. Which makes the DFS growing with linjer function here while the BFS is growing almost constant.
This happen because DFS get the first solution from leaf node and then backtracking

If we look at solutions for size 7x7 board we will draw the same conclusions as board size 8x8 see figure 3

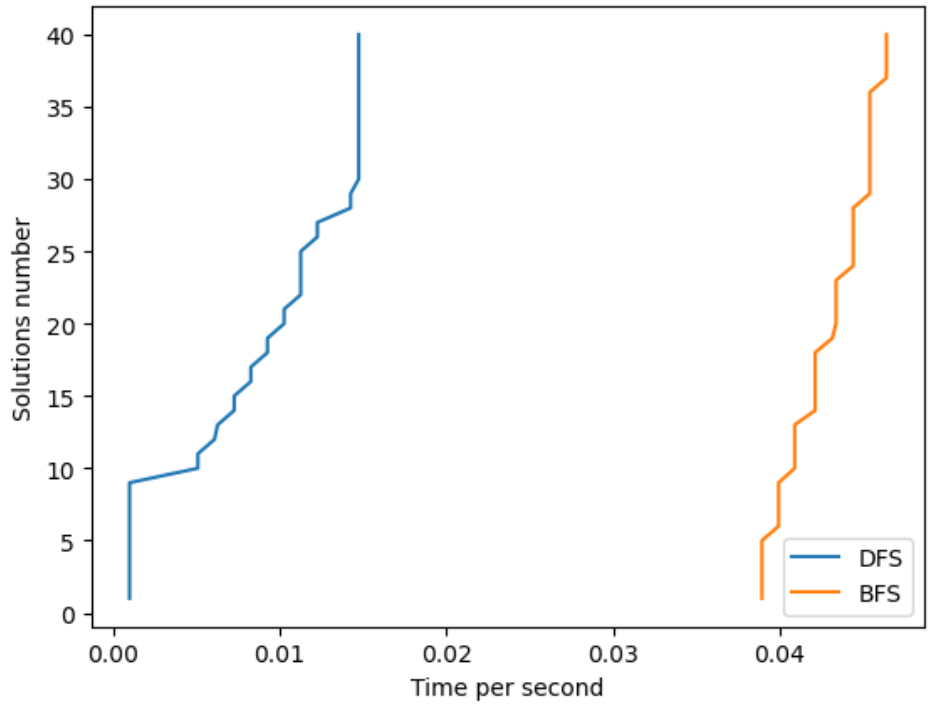


Figure nr 3: showing the time that 40 solutions took for board size 7x7

We discovered an interesting thing when we look at solutions for board size 5x5

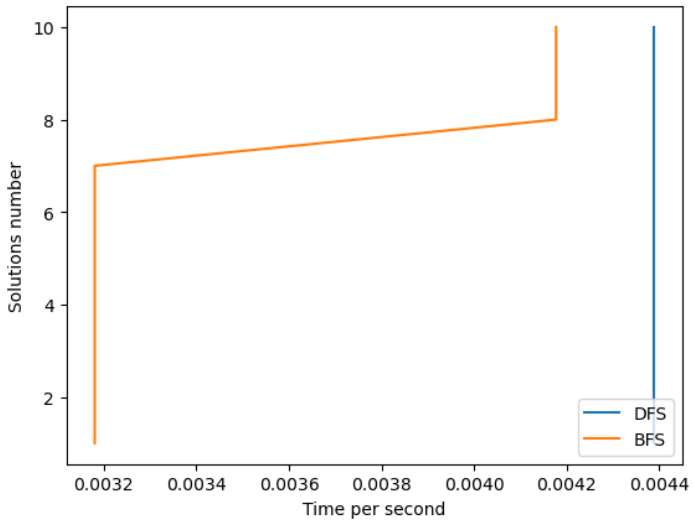


Figure nr 4: showing the time that 10 solutions took for board size 5x5

Here we can see that BFS runs faster than DFS but **note** the graphs will vary with each rerun to algorithms but it will not make a big difference to the final result DFS will always be faster than BFS with board size 8x8.

This happens because the BFS performs better when the target is close to the source.While DFS performs better when the target is far from the source.

We can conclude that DFS is better than BFS in running time. Because DFS use backtracking method in a way that finds a path between two vertices by exploring each possible path as far as possible before backtracking. when the

DFS runs faster but they both give the same quantity of solutions, and BFS performs better when the target is close to the source.

Calculating the time complexity:

Time complexity formel $T(n) = C \times O(f(n))$
We have $T(n)$ which is the running time that we maused. $f(n)$ is $n!$ we will calculating C then test it to see if $f(n)$ is $n!$ and the time complexity that we measure is correct.

Why is O(N!)
The first queen has N placements, the second queen must not be in the same column as the first as well as at an oblique angle, so the second queen has N-1 possibilities, and so on, with a time complexity of O(N!).

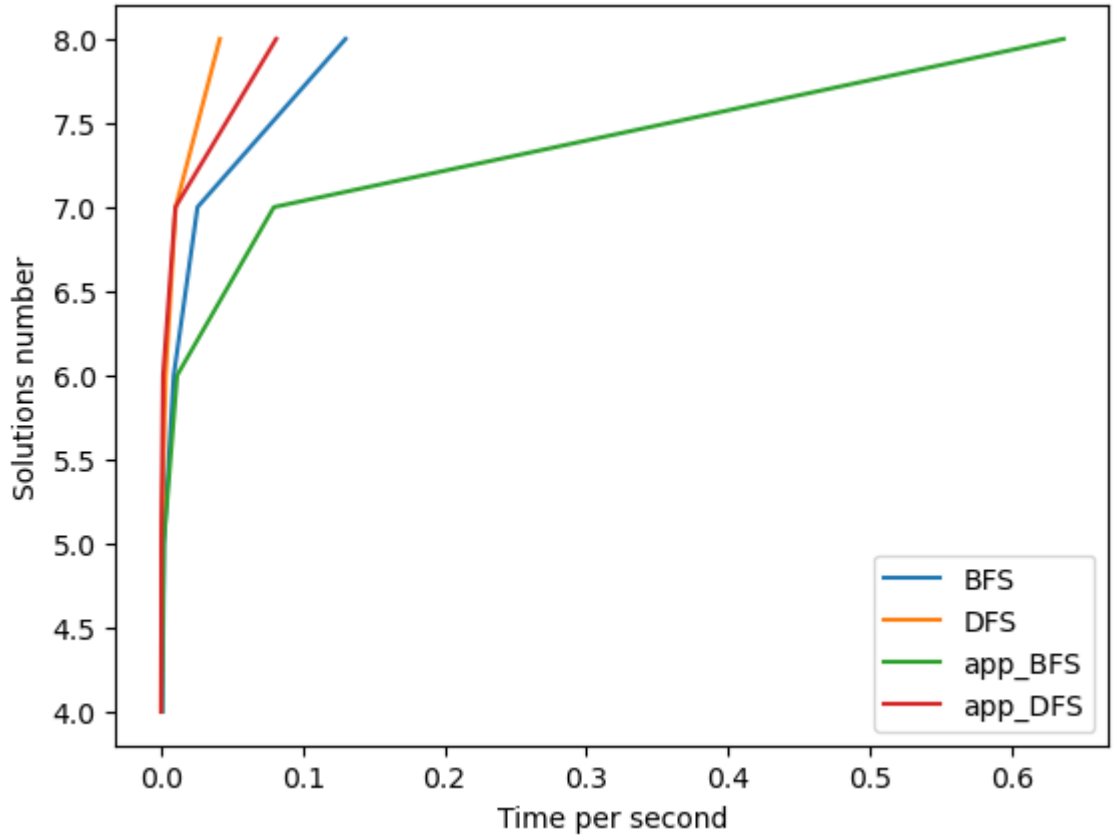
$$C_{DFS} = \frac{T(n)}{O(f(n)))} = 2.0150516498409 69e-06$$

$$C_{BFS} = \frac{T(n)}{O(f(n)))} = 1.5796705828484706e-05$$

Now we have a constant c so we calculate the time complexity for each algorithm and compare it with the running time that we had figure nr 1.

$$T(n) = C \times O(f(n))$$

```
Time complexity for DFS for board size 4-8:  [4.83612396e-05,
2.41806198e-04, 1.45083719e-03, 1.01558603e-02,
 8.12468825e-02]
Time complexity for BFS for board size 4-8:  [3.79120940e-04,
1.89560470e-03, 1.13736282e-02, 7.96153974e-02,
 6.36923179e-01]
```



`Figure nr 5:` compare the both methods with different sizes of boards and the approximations graf for each algorithm

We see that $O(n!)$ is the closest function that can perform the same time complexity for those algorithms.

Backtracking algorithm VS Force Algorithm: The Brute-Force Algorithm involves finding all possible permutations of the queen positions and then evaluating each to determine if it is a valid solution. That means the algorithm starts by finding all the combinations for a set of n queens in n x n places without delay and without regard for order then iterating through all combinations to find all possible solutions.
This is the easier search algorithm to implement and it will always find a solution if it exists. The computational cost is proportional to the number of candidates and increases to be impractical very quickly. This is known as “combinatorial explosion” and limits the use of this algorithm as the computational cost of finding a solution grows exponentially as the search space increases. In this “brute force” algorithm we find the solution by looking at every position on an NxN board, N times, for N queens. Assuming the reader understands Big O Notation, this means $O(n^n)$

So now we know that $O(n^n)$. That means the backtracking algorithm is faster than the force algorithm. Because the n^n grow up faster than $n!$.