KATHMANDU UNIVERSITY

DHULIKHEL, KAVRE

Subject: COMP 314: Algorithms and Complexity

Lab no: 4

Submitted By:

Name: Ayush Kumar Shah
Roll no: 44
Group: CE 3rd year 2nd sem
Level: UNG

Submitted To:

Dr. Bal Krishna Bal

Date of Submission:
02/07/2018

COMP 314 – Algorithms and Complexity

Lab 4 – Backtracking and the n-queens problem

Objective: Understand the backtracking problem taking the n-queens problem as an example.

In the lab report that you would submit, you would need to include the following:

1. Problem Formulation of the n-queens problem

2. Explicit and implicit constraints

3. Pseudocodes of the n-queens problem (both recursive and iterative version). Please refer to the scanned pages on Backtracking which have been uploaded in Piazza.

4. Results of executing the n-queens problem code. (Please refer to the link and tabular format below).

Please go through the link below which contains the Python implementation codes for the n-queens problem.

https://solarianprogrammer.com/2017/11/20/eight-queens-puzzle-python/

Run the code provided in the link for different sized n-queen problems like 4, 8, 12, 16, 20 etc. providing the arguments to the NQueens function. Please record the times for getting the solutions to the different sized n-queens problem. Record the problem size and no. of returned solutions and the time taken to return the solutions in the following table format: N No. of solutions returned Time for getting the solution.

1. **Problem Formulation of the n-queens problem**

N queens are to be placed on an n*n chessboard so that no two attack; that is no two queens are on the same row, column or diagonal. All solutions to the n-queen problem can be represented as n n-tuples (x_1, x_2, x_3,. . . , x_n) where x_i is the column on which queen i is to be placed on row i.

2. **Explicit and implicit constraints**

Explicit constraints- S_i = {1, 2, 3,. . . ,n} ;1 <= i <= n Therefore, the solution set consists of n^n–n tuples.

Implicit constraints- No two x_i can be the same (all queens cannot be on same column) and no two queens can be on the same diagonal.

3. **Pseudocodes of the n-queens problem (both recursive and iterative version)**

Recursive pseudocode:

```
Algorithm Place (k,i)
//returns true if a queen can be placed in kth row and ith column.
//Otherwise it returns false. x[] is a global array whose first (k-1) values have been set

for j <-1 to k-1 do
        If ((x[j]=i) or (Abs(x[j]-i) = Abs(j-k)))
                then return false;
        return true;

Algorithm Nqueens(k,n)

for i<-1 to n do
        if Place(k,i) then
                x[k]<-I;
                if (k=n) then write (x[1:n]);
                else NQueens(k+1,n);
```

Iterative pseudocode:

```
bool PlaceQueens()
{
	int start = 0, col = 1;
	int row2;
	Stack s; //an integer stack
	while(col <= 8)
	{
		bool placed = false;
		for(int row = start+1; row <= 8; row++)
		if can place queen in (row, col)
		{
			placed = true;
			s.push(row);
			col++;
			break;
		}
		if(!placed)
		{
			If s.isEmpty() return false;
			//backtrack to the previous queen
			//and try to place her in a new spot
			s.pop(row2);
			start = row2;
			col--;
		}
		else start = 0;
	}
	return true;
}
```

4. **Results of executing the n-queens problem code.**

Code:

```python
"""The n queens puzzle."""
from time import time
class NQueens:
    """Generate all valid solutions for the n queens puzzle"""
    def __init__(self, size):
        # Store the puzzle (problem) size and the number of valid solutions
        self.size = size
        self.solutions = 0
        self.solve()

    def solve(self):
        """Solve the n queens puzzle and print the number of solutions"""
        positions = [-1] * self.size
        self.put_queen(positions, 0)
        print("Found", self.solutions, "solutions.")

    def put_queen(self, positions, target_row):
        """
        Try to place a queen on target_row by checking all N possible cases.
        If a valid place is found the function calls itself trying to place a queen
        on the next row until all N queens are placed on the NxN board.
        """
        # Base (stop) case - all N rows are occupied
        if target_row == self.size:
            self.show_full_board(positions)
            # self.show_short_board(positions)
            self.solutions += 1
        else:
            # For all N columns positions try to place a queen
            for column in range(self.size):
                # Reject all invalid positions
                if self.check_place(positions, target_row, column):
                    positions[target_row] = column
                    self.put_queen(positions, target_row + 1)

    def check_place(self, positions, ocuppied_rows, column):
        """
        Check if a given position is under attack from any of
        the previously placed queens (check column and diagonal positions)
        """
        for i in range(ocuppied_rows):
            if positions[i] == column or \
                positions[i] - i == column - ocuppied_rows or \
                positions[i] + i == column + ocuppied_rows:

                return False
        return True
```

```
    def show_short_board(self, positions):
        """
        Show the queens positions on the board in compressed form,
        each number represent the occupied column position in the corresponding row.
        """
        line = ""
        for i in range(self.size):
            line += str(positions[i]) + " "
        print(line)

def main():
    """Initialize and solve the n queens puzzle"""
    print("Enter value of n- (4,8,12,16,..) for n queens problem")
    n=int(input())
    start=time()
    NQueens(n)
    end=time()
    result=end-start
    print ("The time to calculate "+str(n)+"-queens problem is "+str(result)+"seconds")

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

Result:

N	No. of solution returned	Time for getting the solution (seconds)
4	2	0.00025653839111328125
8	92	0.019404172897338867
12	14200	11.493143558502197

Output:

Enter value of n- (4,8,12,16,..) for n queens problem

4

. Q . .

. . . Q

Q . . .

. . Q .

. . Q .

Q . . .

. . . Q

. Q . .

Found 2 solutions.

The time to calculate 4-queens problem is 0.00025653839111328125 seconds

Enter value of n- (4,8,12,16,..) for n queens problem

8

Q . . . .

. . . . Q . . .

. . . . Q

. . . . . Q . .

. . Q . . . . .

. . . . . . Q .

. Q . . . . . .

. . . Q . . . .

. . . . Q

. Q . . . . . .
. . . . Q . . .
. . Q . . . . .
Q . . . .
. . . . . . Q .
. . . Q . . . .
. . . . . Q . .

.

.

.

. . . . Q
. . Q . . . . .
Q . . . .
. . . . . Q . .
. Q . . . . . .
. . . . Q . . .
. . . . . . Q .
. . . Q . . . .

. . . . Q
. . . Q . . . .
Q . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .

. . . . . . Q .

. . . . Q . . .

Found 92 solutions.

The time to calculate 8-queens problem is 0.019404172897338867 seconds

Enter value of n- (4,8,12,16,..) for n queens problem

12

Q . . . .

. . Q . . . .

. . . . Q . . . .

. . . . Q . . . .

. . . . Q . .

. . . . Q

. . . . . Q . . . . . .

. . . . Q .

. Q . . . .

. . . . . . Q . . . . .

. . . . Q . . .

. . . Q . . . .

. . . . Q

. . . . Q . .

. . . . Q . . . .

. . Q . . . .

. . . . Q . . . .

. Q . . . .

. . . . Q .

Q . . . .

. . . . . . Q . . . . .

. . . Q . . . .

. . . . . Q . . . . . .

. . . . Q . . .

.

.

.

. . . . Q

. . . . Q . .

. . . . Q . . . .

. . . . Q . . . .

. . Q . . . .

Q . . . .

. . . . . . Q . . . . .

. Q . . . .

. . . . Q .

. . . . . Q . . . . . .

. . . Q . . . .

. . . . Q . . .

Found 14200 solutions.

The time to calculate 12-queens problem is 11.493143558502197 seconds