Laboratory practice No. 2

Luis Bernardo Zuluaga
Universidad Eafit
Medellín, Colombia
lbzuluagag@eafit.edu.co

Manuela Zapata
Universidad Eafit
Medellín, Colombia
mzapatag1@eafit.edu.co

Bosse Bandowski
Universidad Eafit
Medellín, Colombia
bbandowsk@eafit.edu.co

3) Practice for final project defense presentation

1. There are many more methods to solve the n-queens problem, brute force is what we used in this case, but there are even better ways to solve it. Depth first search, for example, is faster and much more efficient. Moreover, there are more methods such as depth first with heuristics, beam search, and branch and bound.
2.

1	5730
2	10450
3	16032
4	22500
5	33430
6	42150
7	51757
8	146686
9	184474
10	387455
11	1718745
12	3790732
13	11491510
14	73583982
15	1501128367
16	DNF
17	DNF
18	DNF

19	DNF
20	DNF
21	DNF
22	DNF
23	DNF
24	DNF
25	DNF
26	DNF
27	DNF
28	DNF
29	DNF
30	DNF
31	DNF
32	DNF

3. To solve given task, we created an auxiliary class "Board" which would be used to store the input data. Each board object will have a dynamic array storing all bad fields and an integer field indicating the size of the board. The input is read and saved in a dynamic array of boards (ArrayList<Board>) to allow different numbers of test cases and keep track of the order. The algorithm iterates over all specified boards and computes the number of solutions for each board by recursively searching all viable queen configurations with backtracking.

4. The algorithm for one particular board starts by placing a queen in the top left corner of the board. It then proceeds to the next column to the right and tries to place a queen in a valid position. If a valid position is found, it continues by trying to place a queen in the next column to the right. If there is no valid position, the algorithm tracks back to the most recently placed queen and tries to look for other valid positions in that column (row by row, top to bottom). When all columns have been filled, a feasible solution has been found and the solution count is increment by 1. The backtracking in this case works the same way as when there are no more valid positions in a column.
The positions of the queens are stored as integers in an array of the same size as the board. The indices of the array indicate the column and the numbers specify the row.

5. Time complexity:
$O(n,m) = n * m * O(n,m-1) * m$
$= n * m! * m$

6. In the time complexity analysis above, *n* indicates the number of test cases and *m* stands for the size (i.e. the side length) of the biggest board. This algorithm is rather slow because it has an additional factor *m* due to the nature of the function *checkPos (x, y, b)* which has linear time complexity. Implemented differently (i.e. with a matrix marking all attacked fields) this could be decreased to constant time.

4) Practice for midterms

1) A. actual>maximo
B. O(n^2)
2) A. return (arr,k+1)
B. O(n!)
3) A. return i;

B. return n;
C. O(n*m)

4) A. temp%10
B. O((n*m)*log(m))

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co