CSE365-ARTIFICIAL INTELLIGENCE

TOPIC: N QUEENS PROBLEM WITH BACKTRACKING.

Submitted by:
Mashrur Hossain Khan
2016-1-60-006

Sadia Tasneem
2016-1-60-074

Fahim Rubaiyat Zahir
2016-1-60-008

Submitted to:
Dr. Muhammad Ibrahim
Assistant Professor

ABSTRACT

This article details the formal specification and verification of the n-queen problem. The main purpose of this article is to specify and verify the abstract model of a C++ program which computes all the possible solutions of the n-queen problem. In this report, a solution is proposed for n-queen problem based on backtracking. Backtracking is a standard-method to find solution for particular kind of problems, known as “Constraint- Satisfaction” problems. The n-Queen problem is basically generalized form 8-Queen problem. In 8-queen problem, the goal is to place ‘n’ queens such that no queen can kill the other using standard chess queen moves. The solution can very easily be extended to the generalized form of the problem for large values of ‘n’.

Introduction:

The N-Queens puzzle is an extension of the 8-Queens puzzle proposed by Max Bezzel in 1848. In the 8-Queens puzzle, the question is how to place 8 queens on a standard 8x8 chess board such that no two queens are currently in a position to attack another queen, given the standard chess rules allowing horizontal, vertical, or diagonal movement by any number of squares. The N-Queens puzzle generalizes that question to that of placing N queens on an NxN chessboard such that none of them can attack another queen. For the 8-Queens problem, there are 92 distinct solution or, if symmetrical solutions are counted together, 12 unique solutions. The primary brute-force method of finding a single solution involves a backtracking search. This is essentially a search of a depth-first tree. We use the simple fact that only one queen can be place on a row to limit the domain to that of finding a solution that places 1 queen in any column in each of the N rows such that no queens are attacking. For every assignment at row n, we try every assignment at row n+1. We keep placing queens at row n+1 until either n=N and a solution is found – so we are done – or there is no possible assignment at that row n that does not conflict with a queen placed previously – so we backtrack and try the next assignment at row n. While this is exhaustive, it is also inefficient. This means that while N=28 may be quickly solvable, N=32 can take a while, and N=34 is severely impractical. One possible optimization that I looked at was checking for possible assignment consistency ahead of time instead of when placing the queen. This method is called backtracking with forward checking and it can be used limit the number of times an invalid position is looked at, as the algorithm recursively searches up and down the tree.

Implementation:

N Queen problem implementation generally consists of placing queens in a particular row and column and continuing to place the next queen in next row and column without violating the same row, same column and same diagonal constraint. The initial permutation consists of placing a queen in first row. There are n columns in the first row offering a possible of n combinations in first row. For every column in the first row there are n columns that are available for placing a queen in the second row raising the number of combinations in the second row to n*n combinations. At nth row there are nn combinations. This can be reduced to a minimum by using the following constraints No two queens can be placed in the same row No two queens can be placed in the same column No two queens can be placed in the same diagonal.

Algorithm:

```
1.Place the queen's column wise, start from the left most column
2.If all the queens are placed
    1.return true and print the solution matrix.
3.Else
    1. Try all the rows in the current column.
    2.Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
    3.if placing the queen in above step doesn't lead to the solution the backtrack mark the current cell in solution matrix as 0 and return false.
```

4*4 chessboard

The above picture shows an NxN chessboard and we have to place N queens on it. So, we will start by placing the first queen. Empty cell

q			

Now, the second step is to place the second queen in a safe position and then the third queen.

q			
		q	

q			
		q	
	q		

Now, you can see that there is no safe place where we can put the last queen. So, we will just change the position of the previous queen. And this is backtracking.

Also, there is no other position where we can place the third queen so we will go back one more step and change the position of the second queen.

q			
			q

And now we will place the third queen again in a safe position until we find a solution.

q			
			q
	q		

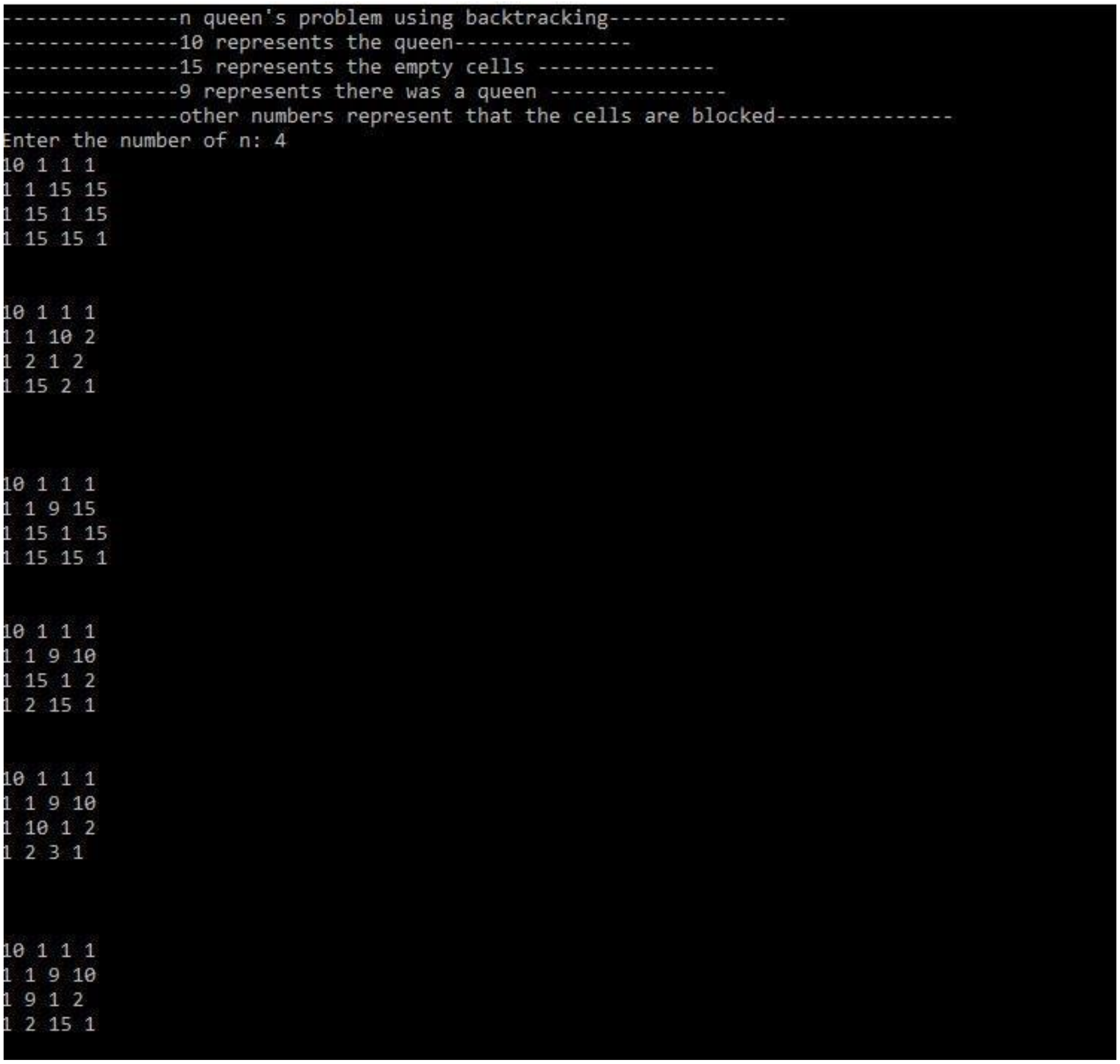
We will continue this process and finally, we will get the solution as shown below.

	q		
			q
q			
		q	

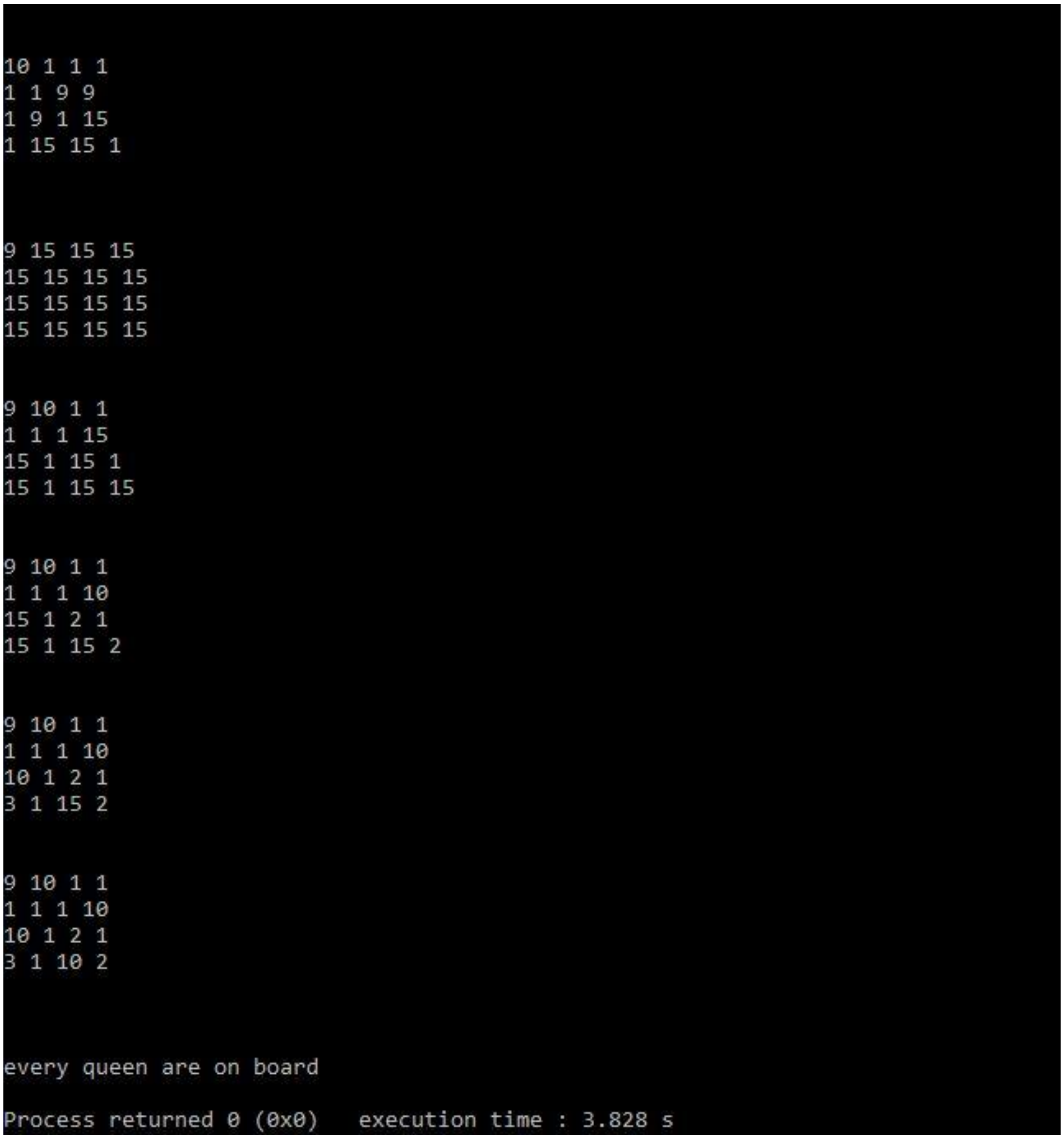
In this code we assign, the empty cell by 15.The 4*4 matrix. Here 10 represents the queen and no queen cannot kill the other chessboard queen through same row, column and diagonal. 15 represents there are empty cell. If any queen already has placed into the chessboard we consider 9.

Start from the leftmost column and the queen placed assign by 10. Any queen cannot attack in same column, row and diagonally each cell assign 1(count=1 already assign in code). That indicates, no other queen attacking in these cell. All the non-attacking cell are empty. Then try all the rows in current column and count++. If placing the queen do not give any solution that time need to backtracking.

Result:



```
---------------n queen's problem using backtracking---------------
---------------10 represents the queen---------------
---------------15 represents the empty cells ---------------
---------------9 represents there was a queen ---------------
---------------other numbers represent that the cells are blocked---------------
Enter the number of n: 4
10 1 1 1
1 1 15 15
1 15 1 15
1 15 15 1


10 1 1 1
1 1 10 2
1 2 1 2
1 15 2 1



10 1 1 1
1 1 9 15
1 15 1 15
1 15 15 1


10 1 1 1
1 1 9 10
1 15 1 2
1 2 15 1


10 1 1 1
1 1 9 10
1 10 1 2
1 2 3 1



10 1 1 1
1 1 9 10
1 9 1 2
1 2 15 1
```



```
10 1 1 1
1 1 9 9
1 9 1 15
1 15 15 1



9 15 15 15
15 15 15 15
15 15 15 15
15 15 15 15


9 10 1 1
1 1 1 15
15 1 15 1
15 1 15 15


9 10 1 1
1 1 1 10
15 1 2 1
15 1 15 2


9 10 1 1
1 1 1 10
10 1 2 1
3 1 15 2


9 10 1 1
1 1 1 10
10 1 2 1
3 1 10 2



every queen are on board

Process returned 0 (0x0)   execution time : 3.828 s
```

Conclusion:

Finding all solutions to the 4queens puzzle is a good example of simple but nontrivial problem. For this reason, it is often used as an example problem for various programming techniques, including nontraditional approaches. Most often it is used as an example of a problem that can be solved with a recursive algorithm.

Source code:

```
//15 means empty cell
//10 means there is a queen
//check backtrack
#include <iostream>
using namespace std;
int flag1=0;
void draw(int board[50][50],int n){
      for(int u=0; u<n; u++){
            for(int v=0; v<n; v++){
                  cout<<board[u][v]<<" ";
            }

            cout<<endl;
      }
      cout<<endl<<endl;
}

void blocked(int i,int j,int n,int board[50][50],int count){
      for(int y=0; y<n; y++){//row
            if( board[i][y]==15)
                  board[i][y]=count;
      }
      for(int z=0; z<n; z++){//column
            if( board[z][j]==15)
                  board[z][j]=count;
      }
      int sum=i+j;
      int sub=i-j;
```

```
	//diagonal
	for(int dx=0; dx<n; dx++){
		for(int dy=0; dy<n; dy++){
			if(dx+dy==sum){
				if(board[dx][dy]==15)
					board[dx][dy]=count;
				}


			if(dx-dy==sub){
				if(board[dx][dy]==15)
					board[dx][dy]=count;
				}
		}

	}

	}



int Closeblocked(int i,int j,int n,int board[50][50],int count){

	for(int dx=0; dx<n; dx++){
		for(int dy=0; dy<n; dy++){

			if(board[dx][dy]==count){
				flag1=1;
				board[dx][dy]=15;
			}
			if(board[0][0]==15){
				//cout<<"abc"<<endl;
				for(int ii=0; ii<n; ii++){
					for(int jj=0; jj<n; jj++){
						board[ii][jj]=15;
					}
				}
				board[0][0]=9;
				count=1;
			}


		}
	}
	count-=1;

	return count;
}
int QueenPosition(int n,int board[50][50],int count){
	int i,j;

	for(i=0; i<n;i++){
```

```
            for(j=0; j<n; j++){
                  if(board[i][j]==10)break;
                  if(board[i][j]==15){
                        if(i==count){
                        count++;
                        board[i][j]=10;//10 means there is a queen
                        blocked(i,j,n,board,count);
                        draw(board,n);
                        break;
                  }
                  }
            }
      }
      cout<<endl;
      return count;
}

void MakeNineFifteen(int board[50][50],int n,int i,int j){
      for(int ii=i+1;ii<n ; ii++){
            for(int jj=0; jj<n; jj++){
                  if(ii==0 && jj==0)continue;
                  else if(ii==i && jj==j)continue;
                  else if(board[ii][jj]==9)
                        board[ii][jj]=15;
            }
      }
}

int main(){
      int
n,board[50][50],i,j,count=0,CountOfTotalQueenOnBoard=0,flag=0,x[10],k=
0;

      cout<<"---------------n queen's problem using backtracking-------
--------\n";
      cout<<"---------------â™› represents the queen---------------\n";
      cout<<"---------------â□Œ represents the emoty sells ------------
---\n";

      cout<<"Enter the number of n: ";
      cin>>n;

      for(i=0;i<n; i++){
            for(j=0; j<n; j++){
                        board[i][j]=15;//every room is empty
            }
      }
      //CountOfTotalQueenOnBoard=QueenPosition(n,board,count);
      //draw(board,n);
      while(1){

            if(count==n){
                    cout<<"every queen are on board"<<endl;
                    break;
                }else{
                    count=QueenPosition(n, board, count);
                    for(i=n-1; i>=0; i--){
                        if(flag==1){
                            flag=0;
                            break;
                        }
                        for(j=n-1; j>=0; j--){
                            if(board[i][j]==10){
                                if(count==n){
                                    cout<<"every queen are on
board"<<endl;

                                    return 0;
                                }
                                board[i][j]=9;
                                x[k]=count-1;
                                count--;
                                //cout<<x[k]<<endl;
                                if(x[k]!=x[k-1] && k!=0){
                                            cout<<"sadia trt
dibe"<<endl;

                                    MakeNineFifteen(board,n,i,j);
                                    k=-1;
                                }
                                //if(board[0][0]==9 &&
count==1)MakeNineFifteen(board,n);
                                count=Closeblocked( i, j, n, board,
count+1);

                                draw(board,n);
                                flag=1;
                                k++;
                                break;
                            }
                        }

                    }
                }

            }
            return 0;
        }
```