LP1

Assignment B4

Artificial Intelligence and Robotics

Title:- AIR

Date of Completion:- 4/11/2

Problem Statement:- Use Heuristic Search Technique to Implement Hill Climb Algorithm.

OR

Constraint Satisfaction Problem.

Implementing Crypt-arithmatic problem or n-queen or graph colouring problem (Branch n Bound and Backtracking)

Learning Objective :-

To understand constraint Satisfaction Problems

To implement n queen problem using Backtracking & branch & bound

Learning Outcomes:- Students will be able to

1) implement n queens problem
2) understand backtraking, branch & Boend, & constraint satisfaction problem

Theory: Software / Hardware Requirements :- OS (linux), Python, java, java IDE.

Theory:-

1) N queens problem:-

i) It is problem to arrange n queens on a chess board in such a way that no queen can attack another.

ii) Constraints:-
queens can attack in any horizontal, vertical & diagonal way.

iii) Algorithm :-

iii) Backtracking:-

1) It is a recursive algorithm for solving problems.
2) Incremental solution building & removes the solution that fails of satisfy the constraints.

Algorithm:-

a) Start from 1st position in the array
b) Place queen in the board & check
   i) After placing the queen, mark the position as a part of the solution and then recursively check if this will lead to a solution.
   ii) If placing the queen doesn't lead to a solution and track back and go to step(a) & place queens to other rows.
   iii) If all queens are placed return TRUE.
c) If all queens are placed return true.
d) If rows are tried and no solution is found return false.

iv) Branch and Bound

used to solve combinational optimization problems.
These problems are typically exponential in terms of time complexity.
Branch & bound can solve them relatively quick

Algorithm:-

i) Start by considering the root node & applying a lower-bounding and upper-bounding procedure to it.
ii) If the bounds match, then an optional solution has been

found and the algorithm is finished

iii) If they do not match, then algorithm runs on the child node

Testcase

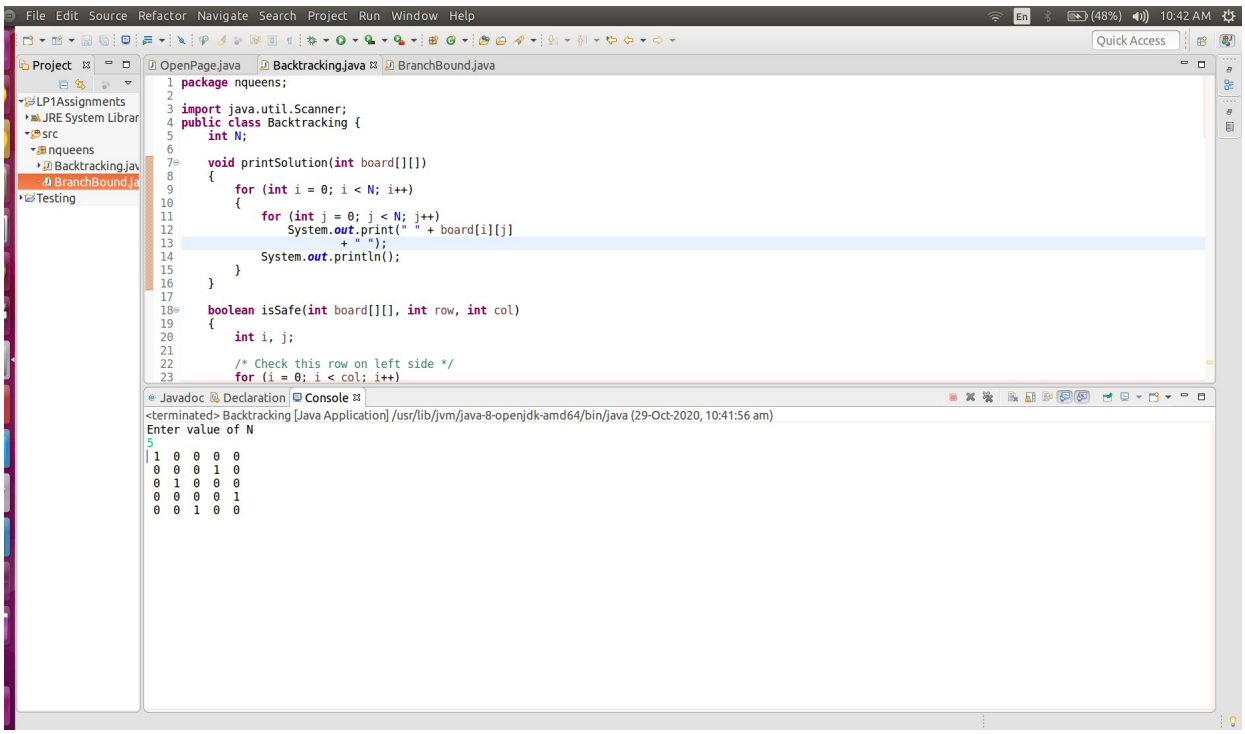
Input	Output	Remark
Backtracking n=5	1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0	Passed
Branch & Bound n=6	0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0	Passed

Conclusion:- Thus I understood constraint staisfaction problems, branch & bound, backtracking techniques & implemented the n queen problem.

N Queens Backtracking

```java
package nqueens;
import java.util.Scanner;
public class Backtracking {
    int N;
    void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(" " + board[i][j]
                        + " ");
            System.out.println();
        }
    }
    boolean isSafe(int board[][], int row, int col)
    {
        int i, j;
        for (i = 0; i < col; i++)
            if (board[row][i] == 1)
                return false;
        for (i=row, j=col; i>=0 && j>=0; i--, j--)
            if (board[i][j] == 1)
                return false;
        for (i=row, j=col; j>=0 && i<N; i++, j--)
            if (board[i][j] == 1)
                return false;
        return true;
    }
    boolean solveNQUtil(int board[][], int col)
    {
        if (col >= N)
            return true;
        for (int i = 0; i < N; i++)
        {
            if (isSafe(board, i, col))
            {
                board[i][col] = 1;
                if (solveNQUtil(board, col + 1) == true)
                    return true;
                board[i][col] = 0; // BACKTRACK
            }
        }
        return false;
    }


    boolean solveNQ(int N)
    {
        int[][] board = new int[N][N];
        for(int i=0;i<N;i++)
        {
            for(int j=0;j<N;j++)
                board[i][j]=0;
        }
        if (solveNQUtil(board, 0) == false)
```

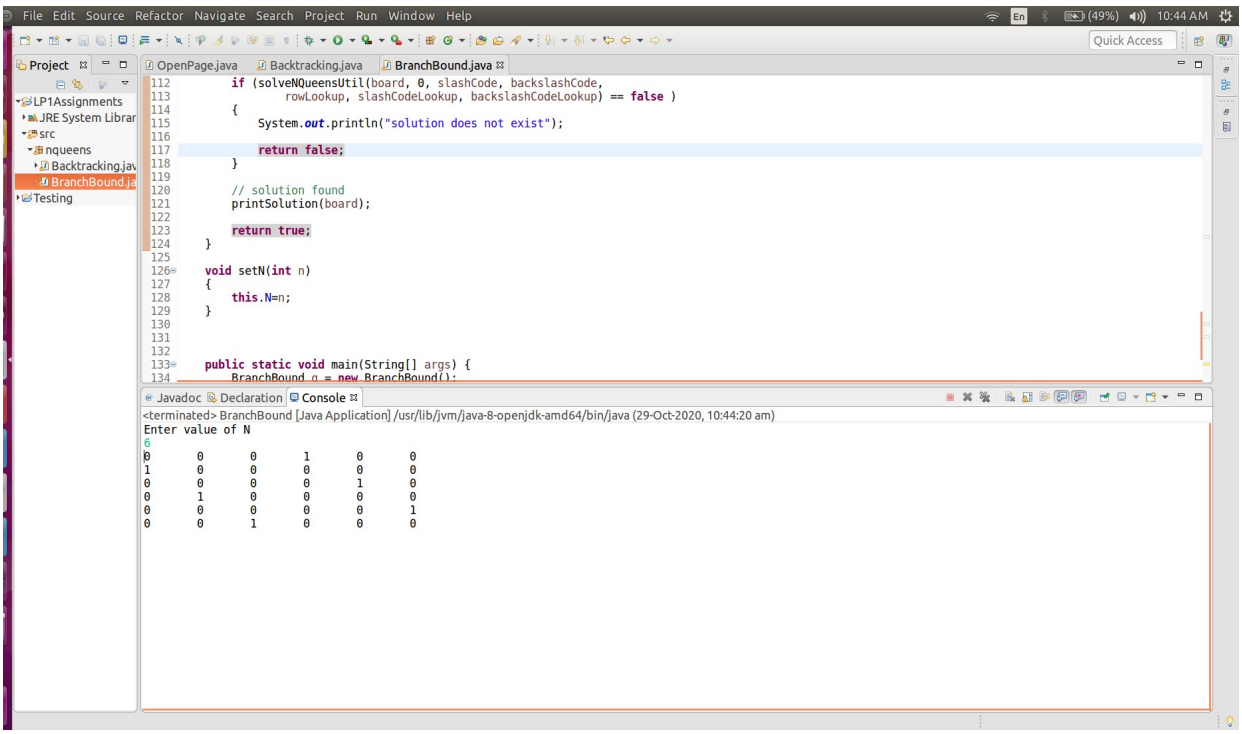
```
        {
            System.out.print("Solution does not exist");
            return false;
        }
        printSolution(board);
        return true;
    }

    void setN(int n)
    {
        this.N=n;
    }
    public static void main(String[] args) {
      Backtracking q = new Backtracking();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter value of N");
        int N = sc.nextInt();
        q.setN(N);
        q.solveNQ(N);
    }
}
```



N Queens Branch and Bound

```
package nqueens;
import java.util.*;
public class BranchBound {
    int N;

    void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(board[i][j]+"\t");
            System.out.println();
        }
    }


    boolean isSafe(int row, int col, int slashCode[][],
                int backslashCode[][], boolean rowLookup[],
                boolean slashCodeLookup[], boolean backslashCodeLookup[] )
    {
        if (slashCodeLookup[slashCode[row][col]] ||
                backslashCodeLookup[backslashCode[row][col]] ||
                rowLookup[row])
            return false;

        return true;
    }


    boolean solveNQueensUtil(int board[][], int col,
                             int slashCode[][], int backslashCode[][], boolean
rowLookup[],
                             boolean slashCodeLookup[], boolean
backslashCodeLookup[] )
    {
        if (col >= N)
            return true;

        for (int i = 0; i < N; i++)
        {
            if ( isSafe(i, col, slashCode, backslashCode, rowLookup,
                    slashCodeLookup, backslashCodeLookup) )
            {
                board[i][col] = 1;
                rowLookup[i] = true;
                slashCodeLookup[slashCode[i][col]] = true;
                backslashCodeLookup[backslashCode[i][col]] = true;

                if ( solveNQueensUtil(board, col + 1, slashCode, backslashCode,
                        rowLookup, slashCodeLookup, backslashCodeLookup) )
                    return true;
                board[i][col] = 0;
                rowLookup[i] = false;
                slashCodeLookup[slashCode[i][col]] = false;
```

```
                backslashCodeLookup[backslashCode[i][col]] = false;
            }
        }
        return false;
    }


    boolean solveNQueens()
    {
        int board[][] = new int[N][N];
        for(int i=0;i<N;i++)
        {
            for(int j=0;j<N;j++)
                board[i][j]=0;
        }
        int slashCode[][] = new int[N][N];
        int backslashCode[][] = new int[N][N];
        boolean rowLookup[] = new boolean[N];
        for(int i=0;i<N;i++)
            rowLookup[i]=false;
        boolean slashCodeLookup[] = new boolean[2*N - 1];
        for(int i=0;i<2*N-1;i++)
            slashCodeLookup[i]=false;
        boolean backslashCodeLookup[] = new boolean[2*N - 1];
        for(int i=0;i<2*N-1;i++)
            backslashCodeLookup[i]=false;
        for (int r = 0; r < N; r++) {
            for (int c = 0; c < N; c++) {
                slashCode[r][c] = r + c;
                backslashCode[r][c] = r - c + N-1;
            }
        }
        if (solveNQueensUtil(board, 0, slashCode, backslashCode,
                rowLookup, slashCodeLookup, backslashCodeLookup) == false )
        {
            System.out.println("solution does not exist");
            return false;
        }
        printSolution(board);

        return true;
    }
    void setN(int n)
    {
        this.N=n;
    }
    public static void main(String[] args) {
      BranchBound q = new BranchBound();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter value of N");
        int N = sc.nextInt();
        q.setN(N);
        q.solveNQueens();

    }
}
```


File Edit Source Refactor Navigate Search Project Run Window Help
10:44 AM
Quick Access
Project
OpenPage.java
Backtracking.java
BranchBound.java
LP1Assignments
JRE System Librar
src
nqueens
Backtracking.jav
BranchBound.ja
Testing
if (solveNQueensUtil(board, 0, slashCode, backslashCode,
rowLookup, slashCodeLookup, backslashCodeLookup) == false)
{
System.out.println("solution does not exist");
return false;
}
// solution found
printSolution(board);
return true;
}
void setN(int n)
{
this.N=n;
}
public static void main(String[] args) {
BranchBound q = new BranchBound();
Javadoc
Declaration
Console
<terminated> BranchBound [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (29-Oct-2020, 10:44:20 am)
Enter value of N
6
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0