# DEVELOP PEAS DESCRIPTION FOR GIVEN AI TASKS

EXPT NO: 1

### AIM:

To develop PEAS description for given AI Tasks.

ALGORITHM:

PEAS stand for a performance measure, environment, Actuator, Sensor.

Performance Measure:

Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precepts.

Environment:

Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion. There are 5 major types of environments.

* Fully observable and partially observable.
* Episodic and sequential.
* Static and Dynamic.
* Discrete and Continuous.
* Deterministic and stochastic.

Actuator:

An actuator is a part of the agent that delivers the output of action to the environment.

Sensor:

Sensors are the receptive part of an agents that takes in the input for the agent.

PROGRAM:

import java.util.Scanner;

public class ChatBot {

String[][] chatBot = {

{"hi", "hello", "hola", "ola", "howdy"},

{"hi", "hello", "hey"},

{"how are you", "how r you", "how r u", "how are u"},

{"good", "doing well"},

{"yes"},

{"no", "NO", "NO!!!!!!!"},

{"good", "bye", "I am not supposed to say"}

};

ChatBot() {

Scanner in = new Scanner(System.in);

while (true) {

System.out.print("You: ");

String quote = in.nextLine();

if (quote.equals("exit") || quote.isEmpty()) {

System.out.println("Bye...");

break;

}

System.out.print("Bot: ");

while (

quote.charAt(quote.length() - 1) == '!' ||

quote.charAt(quote.length() - 1) == '.' ||

quote.charAt(quote.length() - 1) == '?'

) {

quote = quote.substring(0, quote.length() - 1);

}

byte response = 0;

int j = 0;

while (response == 0) {

if (inArray(quote.toLowerCase(), chatBot[j \* 2])) {

response = 2;

int r = (int) Math.floor(Math.random() \* chatBot[(j \* 2) + 1].length);

System.out.print(chatBot[(j \* 2) + 1][r]);

}

j++;

if (j \* 2 == chatBot.length - 1 && response == 0) {

response = 1;

}

}

if (response == 1) {

int r = (int) Math.floor(Math.random() \* chatBot[chatBot.length - 1].length);

System.out.print(chatBot[chatBot.length - 1][r]);

}

System.out.println();

}

}

public static void main(String[] args) {

new ChatBot();

}

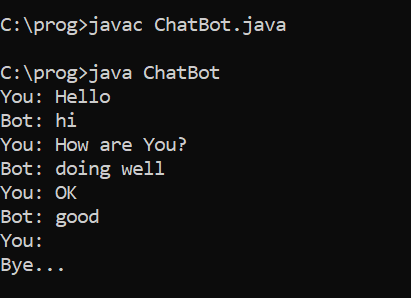
public boolean inArray(String in, String[] str) {

for (String s : str) if (s.equals(in)) return true;

return false;

}

OUTPUT:



RESULT:

Thus the program to develop PEAS description for given AI Tasks.

# DEPTH FIRST SEARCH

EXPT NO: 2a

### AIM:

To write a code to perform depth first search using Depth first search algorithm.

ALGORITHM:

Step 1: Create a class graph for representing directional graph.

Step 2: Call a function add edges to the graph.

Step 3: Mark all not visited vertices.

Step 4: Call helper function point DFS.

Step 5: End.

PROGRAM:

import java.util.\*;

class DFS {

private final int V;

private final LinkedList<Integer>[] adj;

@SuppressWarnings("unchecked")

DFS(int v) {

V = v;

adj = new LinkedList[v];

for (int i = 0; i < v; ++i)

adj[i] = new LinkedList();

}

public static void main(String[] args) {

DFS g = new DFS(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

System.out.println("Following is Depth First Traversal");

g.DFS();

}

void addEdge(int v, int w) {

adj[v].add(w);

}

void DFSUtil(int v, boolean[] visited) {

visited[v] = true;

System.out.print(v + " ");

for (int n : adj[v]) {

if (!visited[n])

DFSUtil(n, visited);

}

}

void DFS() {

boolean[] visited = new boolean[V];

for (int i = 0; i < V; ++i)

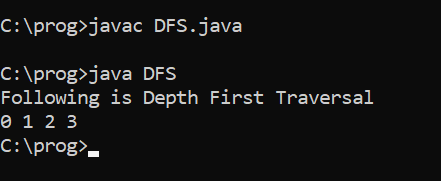
if (!visited[i])

DFSUtil(i, visited);

}

}

OUTPUT:



RESULT:

Thus the java program for implementing using DFS is successfully executed.

# BREADTH FIRST SEARCH

EXPT NO: 2b

### AIM:

To write a java program to print the BFS traversal from the given source vertex.

ALGORITHM:

Step 1: Create a class graph for representing directed graph.

Step 2: Create a constructor.

Step 3: Call a function to add an edge into the graph.

Step 4: Make all the vertices as not visited.

Step 5: Create a queue for BFS.

Step 6: Make the current node as visited and enqueue it.

Step 7: Print statement.

Step 8: End.

PROGRAM:

import java.util.\*;

class BFS {

private final int V;

private final LinkedList<Integer>[] adj;

@SuppressWarnings("unchecked")

BFS(int v) {

V = v;

adj = new LinkedList[v];

for (int i = 0; i < v; ++i) adj[i] = new LinkedList();

}

public static void main(String[] args) {

BFS g = new BFS(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

System.out.println("Following is Breadth First Traversal (starting from vertex 2)");

g.BFS(2);

}

void addEdge(int v, int w) {

adj[v].add(w);

}

void BFS(int s) {

boolean[] visited = new boolean[V];

LinkedList<Integer> queue = new LinkedList<Integer>();

visited[s] = true;

queue.add(s);

while (queue.size() != 0) {

s = queue.poll();

System.out.print(s + " ");

for (int n : adj[s]) {

if (!visited[n]) {

visited[n] = true;

queue.add(n);

}

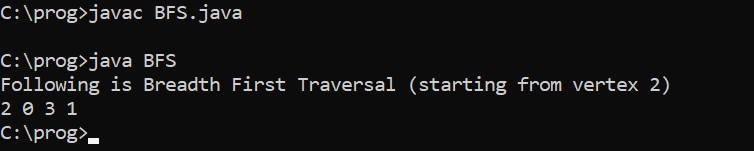
}

}

}

}

OUTPUT:



RESULT:

Thus the java program for implementing and executing BFS is executed successfully.

# IMPLEMENT A\* AND MEMORY BOUNDED A\* ALGORITHMS

EXPT NO: 3

### AIM:

To implement A\* and memory bounded A\*algorithms.

ALGORITHM:

We create two lists – Open list and closed list.

//A\* Search Algorithm.

1. Initialize the open list
2. Initialize the closed list

Put the starting mode on the open list (you can leave its f at zero)

1. While the open list is not empty
2. Find the node with the least F on the open list, call it “q”.
3. Pop q off the open list.
4. Generate q’s 8 successors and set their parents to q.
5. For each successor.

If successor is the goal, stop search.

else, compute both g and h for successor.

Successor. g = q. g + distance between successor and q.

Successor. h = distance from goal to successor.

Successor. f = successor. g + successor. h

If a node with same position as successor is in the OPEN list which has a lower f than successor, skip this successor.

If a node with same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise. Add the node to the open list end (for loop).

Push q on the closed list end (while loop).

PROGRAM:

import java.util.\*;

public class AstarSearchAlgo {

public static void main(String[] args) {

Node n1 = new Node("Arad", 366);

Node n2 = new Node("Zerind", 374);

Node n3 = new Node("Oradea", 380);

Node n4 = new Node("Sibiu", 253);

Node n5 = new Node("Fagaras", 178);

Node n6 = new Node("Rimnicu Vilcea", 193);

Node n7 = new Node("Pitesti", 98);

Node n8 = new Node("Timisoara", 329);

Node n9 = new Node("Lugoj", 244);

Node n10 = new Node("Mehadia", 241);

Node n11 = new Node("Drobeta", 242);

Node n12 = new Node("Craiova", 160);

Node n13 = new Node("Bucharest", 0);

Node n14 = new Node("Giurgiu", 77);

n1.adjacencies = new Edge[]{

new Edge(n2, 75),

new Edge(n4, 140),

new Edge(n8, 118)

};

n2.adjacencies = new Edge[]{

new Edge(n1, 75),

new Edge(n3, 71)

};

n3.adjacencies = new Edge[]{

new Edge(n2, 71),

new Edge(n4, 151)

};

n4.adjacencies = new Edge[]{

new Edge(n1, 140),

new Edge(n5, 99),

new Edge(n3, 151),

new Edge(n6, 80),

};

n5.adjacencies = new Edge[]{

new Edge(n4, 99),

new Edge(n13, 211)

};

n6.adjacencies = new Edge[]{

new Edge(n4, 80),

new Edge(n7, 97),

new Edge(n12, 146)

};

n7.adjacencies = new Edge[]{

new Edge(n6, 97),

new Edge(n13, 101),

new Edge(n12, 138)

};

n8.adjacencies = new Edge[]{

new Edge(n1, 118),

new Edge(n9, 111)

};

n9.adjacencies = new Edge[]{

new Edge(n8, 111),

new Edge(n10, 70)

};

n10.adjacencies = new Edge[]{

new Edge(n9, 70),

new Edge(n11, 75)

};

n11.adjacencies = new Edge[]{

new Edge(n10, 75),

new Edge(n12, 120)

};

n12.adjacencies = new Edge[]{

new Edge(n11, 120),

new Edge(n6, 146),

new Edge(n7, 138)

};

n13.adjacencies = new Edge[]{

new Edge(n7, 101),

new Edge(n14, 90),

new Edge(n5, 211)

};

n14.adjacencies = new Edge[]{

new Edge(n13, 90)

};

AstarSearch(n1, n13);

List<Node> path = printPath(n13);

System.out.println("Path: " + path);

}

public static List<Node> printPath(Node target) {

List<Node> path = new ArrayList<>();

for (Node node = target; node != null; node = node.parent) {

path.add(node);

}

Collections.reverse(path);

return path;

}

public static void AstarSearch(Node source, Node goal) {

Set<Node> explored = new HashSet<>();

PriorityQueue<Node> queue = new PriorityQueue<>(20,

Comparator.comparingDouble(i -> i.f\_scores)

);

source.g\_scores = 0;

queue.add(source);

boolean found = false;

while ((!queue.isEmpty()) && (!found)) {

Node current = queue.poll();

explored.add(current);

if (current.value.equals(goal.value)) {

found = true;

}

for (Edge e : current.adjacencies) {

Node child = e.target;

double cost = e.cost;

double temp\_g\_scores = current.g\_scores + cost;

double temp\_f\_scores = temp\_g\_scores + child.h\_scores;

if ((explored.contains(child)) &&

(temp\_f\_scores >= child.f\_scores)) {

} else if ((!queue.contains(child)) ||

(temp\_f\_scores < child.f\_scores)) {

child.parent = current;

child.g\_scores = temp\_g\_scores;

child.f\_scores = temp\_f\_scores;

queue.remove(child);

queue.add(child);

}

}

}

}

}

class Node {

public final String value;

public final double h\_scores;

public double g\_scores;

public double f\_scores = 0;

public Edge[] adjacencies;

public Node parent;

public Node(String val, double hVal) {

value = val;

h\_scores = hVal;

}

public String toString() {

return value;

}

}

class Edge {

public final double cost;

public final Node target;

public Edge(Node targetNode, double costVal) {

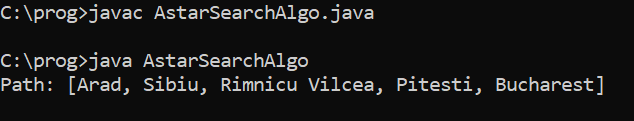
target = targetNode;

cost = costVal;

}

}

OUTPUT:



RESULT:

Thus the java program for implementation A\* and memory bounded A\* Algorithms is executed and verified successfully.

# IMPLEMENT GENETIC ALGORITHMS FOR AI TASKS

EXPT NO: 4

### AIM:

To implement Genetic Algorithm for AI Tasks.

ALGORITHM:

Step 1: Start

Step 2: The first step is to create a population of random bit strings. We could use Boolean values True and False, string values ‘0’ and ‘1’, or integer values 0 and 1. In this case, we will use integer values

Step 3: To use a function named objective () as a generic objective function and call it to get a fitness score.

Step 4: We can then call this function one time for each position in the population to create a list of parents.

Step 5: End.

PROGRAM:

import java.util.Random;

public class SimpleDemoGA {

Population population = new Population();

Individual fittest;

Individual secondFittest;

int generationCount = 0;

public static void main(String[] args) {

Random rn = new Random();

SimpleDemoGA demo = new SimpleDemoGA();

demo.population.initializePopulation();

demo.population.calculateFitness();

System.out.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

while (demo.population.fittest < 5) {

++demo.generationCount;

demo.selection();

demo.crossover();

if (rn.nextInt() % 7 < 5) {

demo.mutation();

}

demo.addFittestOffspring();

demo.population.calculateFitness();

System.out.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

}

System.out.println("\nSolution found in generation " + demo.generationCount);

System.out.println("Fitness: " + demo.population.getFittest().fitness);

System.out.print("Genes: ");

for (int i = 0; i < 5; i++) {

System.out.print(demo.population.getFittest().genes[i]);

}

System.out.println();

}

void selection() {

fittest = population.getFittest();

secondFittest = population.getSecondFittest();

}

//Crossover

void crossover() {

Random rn = new Random();

int crossOverPoint = rn.nextInt(population.individuals[0].geneLength);

for (int i = 0; i < crossOverPoint; i++) {

int temp = fittest.genes[i];

fittest.genes[i] = secondFittest.genes[i];

secondFittest.genes[i] = temp;

}

}

void mutation() {

Random rn = new Random();

int mutationPoint = rn.nextInt(population.individuals[0].geneLength);

if (fittest.genes[mutationPoint] == 0) {

fittest.genes[mutationPoint] = 1;

} else {

fittest.genes[mutationPoint] = 0;

}

mutationPoint = rn.nextInt(population.individuals[0].geneLength);

if (secondFittest.genes[mutationPoint] == 0) {

secondFittest.genes[mutationPoint] = 1;

} else {

secondFittest.genes[mutationPoint] = 0;

}

}

Individual getFittestOffspring() {

if (fittest.fitness > secondFittest.fitness) {

return fittest;

}

return secondFittest;

}

void addFittestOffspring() {

fittest.calcFitness();

secondFittest.calcFitness();

int leastFittestIndex = population.getLeastFittestIndex();

population.individuals[leastFittestIndex] = getFittestOffspring();

}

}

class Individual {

int fitness;

int[] genes = new int[5];

int geneLength = 5;

public Individual() {

Random rn = new Random();

for (int i = 0; i < genes.length; i++) {

genes[i] = Math.abs(rn.nextInt() % 2);

}

fitness = 0;

}

public void calcFitness() {

fitness = 0;

for (int i = 0; i < 5; i++) {

if (genes[i] == 1) {

++fitness;

}

}

}

}

class Population {

Individual[] individuals = new Individual[10];

int fittest = 0;

public void initializePopulation() {

for (int i = 0; i < individuals.length; i++) {

individuals[i] = new Individual();

}

}

public Individual getFittest() {

int maxFit = Integer.MIN\_VALUE;

int maxFitIndex = 0;

for (int i = 0; i < individuals.length; i++) {

if (maxFit <= individuals[i].fitness) {

maxFit = individuals[i].fitness;

maxFitIndex = i;

}

}

fittest = individuals[maxFitIndex].fitness;

return individuals[maxFitIndex];

}

public Individual getSecondFittest() {

int maxFit1 = 0;

int maxFit2 = 0;

for (int i = 0; i < individuals.length; i++) {

if (individuals[i].fitness > individuals[maxFit1].fitness) {

maxFit2 = maxFit1;

maxFit1 = i;

} else if (individuals[i].fitness > individuals[maxFit2].fitness) {

maxFit2 = i;

}

}

return individuals[maxFit2];

}

public int getLeastFittestIndex() {

int minFitVal = Integer.MAX\_VALUE;

int minFitIndex = 0;

for (int i = 0; i < individuals.length; i++) {

if (minFitVal >= individuals[i].fitness) {

minFitVal = individuals[i].fitness;

minFitIndex = i;

}

}

return minFitIndex;

}

public void calculateFitness() {

for (Individual individual : individuals) {

individual.calcFitness();

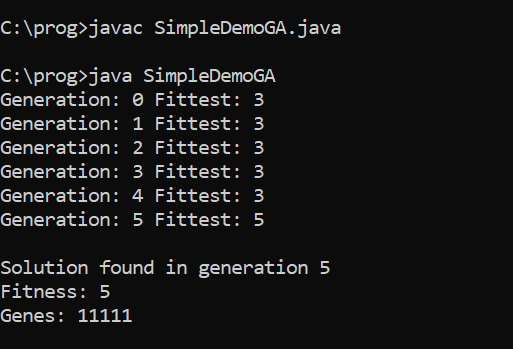
}

getFittest();

}

}

OUTPUT:



RESULT:

Thus the program to implement Genetic Algorithms for AI Tasks has been executed and verified successfully.

IMPLEMENT SIMULATED ANNEALING ALGORITHMS FOR AI TASKS

EXPT NO: 5

AIM:

To implement Simulated Annealing algorithms for AI Tasks.

ALGORITHM:

Step 1: Start

Step 2: initial\_temp = 90

final\_temp = .1

alpha = 0.01

current\_temp = initial\_temp

Step 3: Set initial state and set it as the solution. You can set it up as a particular state or generate it randomly.

current\_state = initial\_state

solution = current\_state

Step 4: Repeat this process until the current temperature is less than the final temperature.

while current\_temp > final\_temp

Step 5: For each iteration, we will get a random neighbor of the current state ( the following state that we of from the current state)

neighbor = random.choice(self.get\_neighbors())

cost\_diff = self.get\_cost(self.current\_state) = self.get\_cost(neighbour)

Step 6: End

PROGRAM:

import java.util.Arrays;

public class SimulatedAnnealing {

static final double Tmin = .0001;

static final double alpha = 0.9;

static final int numIterations = 100;

static final int M = 5, N = 5;

public static double T = 1;

public static void main(String[] args) {

String[][] sourceArray = new String[M][N];

Solution min = new Solution(Double.MAX\_VALUE, null);

Solution currentSol = genRandSol();

while (T > Tmin) {

for (int i = 0; i < numIterations; i++) {

if (currentSol.CVRMSE < min.CVRMSE) {

min = currentSol;

break;

}

Solution newSol = neighbor(currentSol);

double ap = Math.pow(Math.E,

(currentSol.CVRMSE - newSol.CVRMSE) / T);

if (ap > Math.random())

currentSol = newSol;

}

T \*= alpha;

}

System.out.println(min.CVRMSE + "\n\n");

for (String[] row : sourceArray) Arrays.fill(row, "X");

for (int object : min.config) {

int[] coord = indexToPoints(object);

sourceArray[coord[0]][coord[1]] = "-";

}

for (String[] row : sourceArray)

System.out.println(Arrays.toString(row));

}

public static Solution neighbor(Solution currentSol) {

return currentSol;

}

public static Solution genRandSol() {

int[] a = {1, 2, 3, 4, 5};

return new Solution(-1, a);

}

public static int[] indexToPoints(int index) {

return new int[]{index % M, index / M};

}

static class Solution {

public double CVRMSE;

public int[] config;

public Solution(double CVRMSE, int[] configuration) {

this.CVRMSE = CVRMSE;

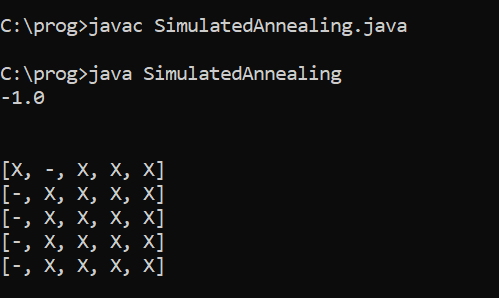
config = configuration;

}

}

}

OUTPUT:



RESULT:

Thus the program to implement Simulated Annealing algorithm for AI Tasks is executed and verified successfully.

IMPLEMENT ALPHA-BETA TREE SEARCH

EXPT NO: 6

AIM:

To implement Alpha-Beta Tree Search with MINIMAX

ALGORITHM:

Step 1: Start

Step 2: Start with the initial move and define the worst-class alpha and beta values, I.e, α = -∞ and β = +∞. Since the initial value of alpha is less than beta so we didn’t prune it. Now it’s my turn for MAX. So at node D, the value of alpha will be calculated.

Step 3: Now the next move will be on node B and its turn for MIN now, Som at node B, the value of alpha- beta will be MIN

Step 4: In the next step the algorithm again comes to node A from node B. At node A Alpha will be changed to maximum value as MAX

Step 5: At node F the value of Alpha will be compared to the left branch which is 0

Step 6: Now node F will return the node value 1 to C and will compare the beta value at C. Now its turn for MIN

Step 7: Return Output

Step 8: Stop

PROGRAM:

class CFG{

static int MAX = 1000;

static int MIN = -1000;

static int minmax(int depth, int nodeIndex, boolean maximizingPlayer,

int[] values, int alpha, int beta){

if (depth==3) return values[nodeIndex];

if (maximizingPlayer){

int best = MIN;

for (int i =0; i<2; i++){

int val = minmax(depth+1, nodeIndex\*2+i, false,

values, alpha, beta);

best = Math.max(best, val);

alpha = Math.max(alpha, best);

if (beta<=alpha) break;

}

return best;

}

else

{

int best = MAX;

for (int i = 0; i<2; i++){

int val = minmax(depth+1, nodeIndex\*2+i, true,

values, alpha, beta);

best = Math.min(best, val);

beta = Math.min(beta, best);

if (beta<=alpha) break;

}

return best;

}

}

public static void main(String[] args){

int[] values = {3, 5, 6, 9, 1, 2, 0, -1};

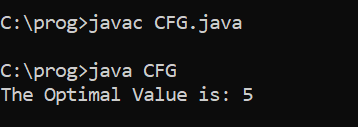
System.out.println("The Optimal Value is: "+

minmax(0, 0, true, values, MIN, MAX));

}

}

OUTPUT:



RESULT:

Thus the program to implement alpha-beta search with MINIMAX is executed and verified successfully.

Implement backtracking algorithms for CSP

Backtracking Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row

then mark this [row, column] as part of the

solution and recursively check if placing

queen here leads to a solution.

b) If placing the queen in [row, column] leads to

a solution then return true.

c) If placing queen doesn't lead to a solution then

unmark this [row, column] (Backtrack) and go to

step (a) to try other rows.

4) If all rows have been tried and nothing worked,

return false to trigger backtracking.

Implementation of Backtracking solution

Program:

/\* Java program to solve N Queen Problem using

backtracking \*/

public class NQueenProblem {

final int N = 4;

/\* A utility function to print solution \*/

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();

}

}

/\* A utility function to check if a queen can

be placed on board[row][col]. Note that this

function is called when "col" queens are already

placeed in columns from 0 to col -1. So we need

to check only left side for attacking queens \*/

boolean isSafe(int board[][], int row, int col)

{

int i, j;

/\* Check this row on left side \*/

for (i = 0; i < col; i++)

if (board[row][i] == 1)

return false;

/\* Check upper diagonal on left side \*/

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j] == 1)

return false;

/\* Check lower diagonal on left side \*/

for (i = row, j = col; j >= 0 && i < N; i++, j--)

if (board[i][j] == 1)

return false;

return true;

}

/\* A recursive utility function to solve N

Queen problem \*/

boolean solveNQUtil(int board[][], int col)

{

/\* base case: If all queens are placed

then return true \*/

if (col >= N)

return true;

/\* Consider this column and try placing

this queen in all rows one by one \*/

for (int i = 0; i < N; i++) {

/\* Check if the queen can be placed on

board[i][col] \*/

if (isSafe(board, i, col)) {

/\* Place this queen in board[i][col] \*/

board[i][col] = 1;

/\* recur to place rest of the queens \*/

if (solveNQUtil(board, col + 1) == true)

return true;

/\* If placing queen in board[i][col]

doesn't lead to a solution then

remove queen from board[i][col] \*/

board[i][col] = 0; // BACKTRACK

}

}

/\* If the queen can not be placed in any row in

this column col, then return false \*/

return false;

}

/\* This function solves the N Queen problem using

Backtracking. It mainly uses solveNQUtil () to

solve the problem. It returns false if queens

cannot be placed, otherwise, return true and

prints placement of queens in the form of 1s.

Please note that there may be more than one

solutions, this function prints one of the

feasible solutions.\*/

boolean solveNQ()

{

int board[][] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

System.out.print("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

// driver program to test above function

public static void main(String args[])

{

NQueenProblem Queen = new NQueenProblem();

Queen.solveNQ();

}

}

Output:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

Implement local search algorithms for CSP