Assignment No: 7

**Title:** Use Heuristic search technique to implement Hill Climbing Algorithm.

**Theory:**

Hill climbing search algorithm is one of the simplest algorithms which falls under local search and optimization techniques. Hill climbing evaluates the possible next moves and picks the one which has the least distance. It also checks if the new state after the move was already observed. If true, then it skips the move and picks the next best move. As the vacant tile can only be filled by its neighbors, Hill climbing sometimes gets locked and couldn’t find any solution. It’s one of the major drawbacks of this algorithm.

Another drawback which is highly documented is local optima. The algorithm decides the next move(state) based on immediate distance(cost), assuming that the small improvement now is the best way to reach the final state. However, the path chosen may lead to higher cost(more steps) later.

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution.

So, given a large set of inputs and a good heuristic function, the algorithm tries to find the best possible solution to the problem in the most reasonable time period.

Solution is not necessary, a optimal solution (Global Optimal Maxima) but it is consider to be good solution according to time period.

Mathematical optimization problems: Implies that hill-climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs.

For example, hill climbing can be applied to the travelling salesman problem.where we need to minimize the distance traveled by the salesman.

**Program:**

HillClimbingAlgo.java

import java.util.ArrayList;

import java.util.Scanner;

public class HillClimbingAlgo {

State gstate, cstate, sstate;

Scanner sc = new Scanner(System.in);

ArrayList<State> ngb = new ArrayList<State>();

HillClimbingAlgo() {

gstate = new State();

cstate = new State();

sstate = new State();

}

void display(State s) {

int k = 0;

for (int j = 0; j < 3; j++) {

for (int i = 0; i < 3; i++) {

System.out.print(s.arr[k] + " ");

k++;

}

System.out.println();

}

}

void input() {

System.out.println("Enter the start state");

for (int i = 0; i < 9; i++) {

sstate.arr[i] = sc.nextInt();

}

System.out.println("Enter the goal state");

for (int i = 0; i < 9; i++) {

gstate.arr[i] = sc.nextInt();

}

}

int h(State s) {

int hvalue = 0;

for (int i = 0; i < 9; i++) {

if (s.arr[i] != gstate.arr[i]) {

hvalue++;

}

}

return hvalue;

}

int blpos(State s) {

for (int j = 0; j < 9; j++) {

if (s.arr[j] == 0) {

return j;

}

}

return 0;

}

void Movegen(State s) {

int p = blpos(s);

ngb.clear();

if (p % 3 != 0) {

State n1 = new State(s);

n1.arr[p] = n1.arr[p - 1];

n1.arr[p - 1] = 0;

n1.h = h(n1);

ngb.add(n1);

}

if (p < 6) {

State n1 = new State(s);

n1.arr[p] = n1.arr[p + 3];

n1.arr[p + 3] = 0;

n1.h = h(n1);

ngb.add(n1);

}

if (p > 2 && p < 9) {

State n1 = new State(s);

n1.arr[p] = n1.arr[p - 3];

n1.arr[p - 3] = 0;

n1.h = h(n1);

ngb.add(n1);

}

if (p % 3 != 2) {

State n1 = new State(s);

n1.arr[p] = n1.arr[p + 1];

n1.arr[p + 1] = 0;

n1.h = h(n1);

ngb.add(n1);

}

}

int lowestscore() {

int i = 0, min = 999;

for (int j = 0; j < ngb.size(); j++) {

if (min > ngb.get(j).h) {

min = ngb.get(j).h;

i = j;

}

}

return i;

}

State hillclimbing() {

int low = 0, done = 0;

State n, nn;

sstate.h = h(sstate);

sstate.paraent = null;

n = sstate;

Movegen(n);

low = lowestscore();

nn = ngb.get(low);

display(n);

System.out.println();

while (nn.h < n.h) {

display(nn);

System.out.println();

nn.paraent = n;

n = nn;

Movegen(n);

low = lowestscore();

nn = ngb.get(low);

}

return nn;

}

public static void main(String[] args) {

HillClimbingAlgo ob = new HillClimbingAlgo();

ob.input();

System.out.println("Intermediate States");

ob.hillclimbing();

}

}

State.java

public class State {

int arr[];

State paraent;

int h = 0;

State(State s1) {

this.arr = new int[9];

for (int i = 0; i < s1.arr.length; i++) {

this.arr[i] = s1.arr[i];

}

}

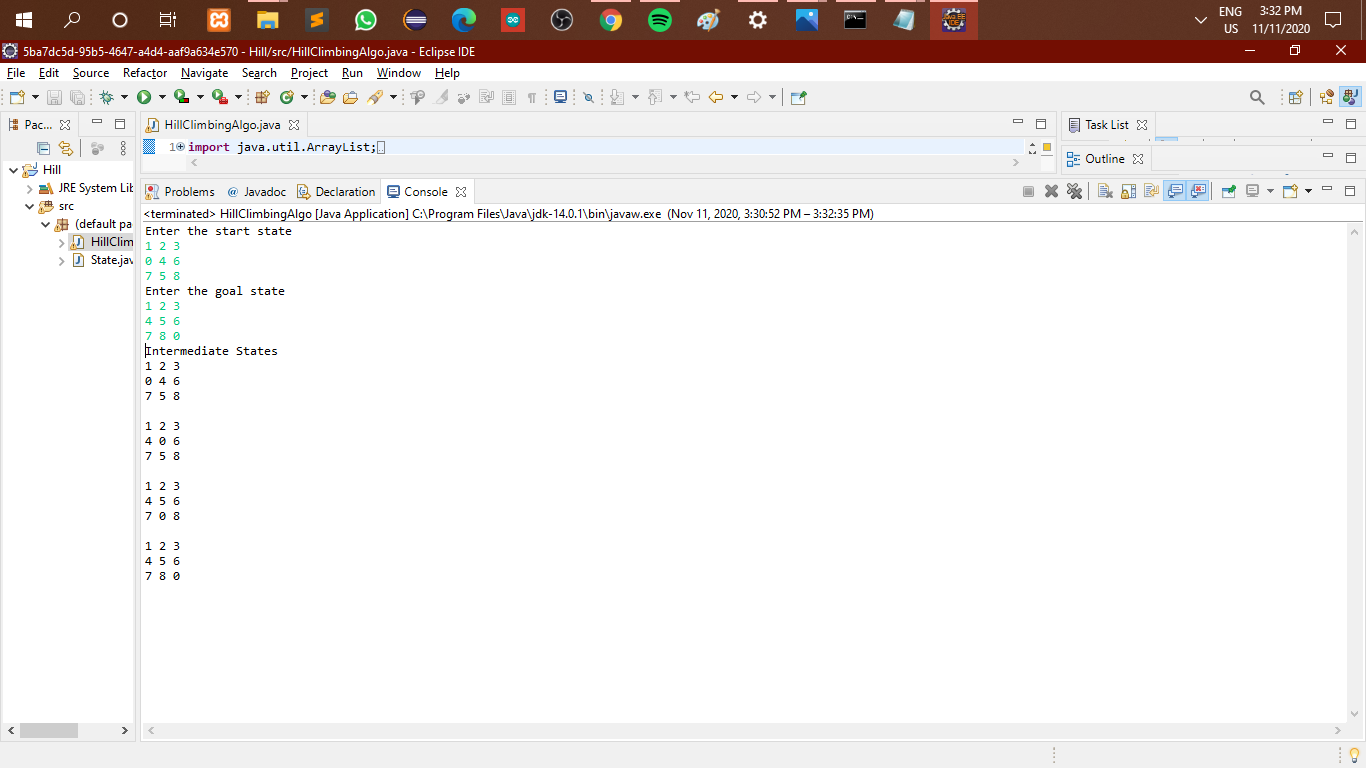
State() {

arr = new int[9];

}

}

**Output:**

****