Lab 4

Problem Statement: A* Algorithm

NAME: Harshita Bhagat ROLLNO: 31

CLASS: TY - IT A BATCH: 2

Code:

```
package AI;

import java.util.*;


public class AstarAlgo {

  static int mov = 0;

  static int[][] board = new int[3][3];

  static int[][] goal  = new int[3][3];

  static int[][] moves = new int[4][9];

  // for the 4 possible value {up, down, right, left} in 1D


  static Node createNode(int[][] initial, int[][] movepos, double ftemp, int gtemp) {

    return new Node(initial, movepos, ftemp, gtemp);

  }


  static Node insertNode(Node node, int[][] initial, int[][] movepos, double ftemp, int gtemp) {

    Node temp = createNode(initial, movepos, ftemp, gtemp);

    if (node == null)  // empty linked list

    {

      node = temp;
```

```
        node.next = null;
    }
    else if (node.fa >= ftemp)  // insert at beginning
    {
        temp.next = node;
        node = temp;
    }
    else // insert in between
    {
        Node start = node;
        while (start.next != null && start.next.fa < ftemp) {
            start = start.next;
        }
        temp.next = start.next;
        start.next = temp;
    }

    // inserting at end wouldn't preserve the order of nodes!

    return node;
}

static Node insertClosed(Node node, int[][] initial, int[][] movepos, double ftemp, int gtemp) {
    Node temp = createNode(initial, movepos, ftemp, gtemp);
    if (node == null) // empty
    {
            node = temp;
            node.next = null;
        }
        else // insert at the end
        {
            Node start = node;
            while (start.next != null) {
                start = start.next;
            }
            temp.next = start.next;
            start.next = temp;
        }
        return node;
    }


    public static void updateParent(Node CLOSED, Node OPEN, int[][] newParent, int[][] parent, int
gtemp) {
        Node temp = CLOSED; // closed traverse
        Node temp1 = OPEN;
        while (temp != null) {
            if (Arrays.deepEquals(temp.ibp, parent))
                // .deepEquals {array comparision} ibp equals to parent
            {
                // System.out.println("updating!!"); // Test karna hai
                while (temp1 != null) {
```

```
                temp1.fa = temp1.fa - temp1.ga + gtemp + 1; // gtemp: new cost, +1 assume kar rahe
                temp1.ga = gtemp + 1;
                temp1 = temp1.next;
            }
            temp.ibp = newParent;
            temp.fa = temp.fa - temp.ga + gtemp;
            temp.ga = gtemp;
        }
        temp = temp.next;
    }
}

static double findHeuristicValue(int[][] goal, int[][] boardpos)
{
    // Heuristic Function used: Euclidean Distance
    double heuristicval = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            heuristicval += Math.pow(goal[i][j] - boardpos[i][j], 2);
        }
    }
    return Math.sqrt(heuristicval);
}

static Node heuristic(Node open, int[][] moves, int[][] goal, int gtemp, int[][] initial, Node closed) {
    int di = 0; // possible moves index
```

```
while (di < mov) // mov: possible moves
{
    // 1D moves --> 2D arr
    int dj = 0;
    int[][] arr = new int[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            arr[i][j] = moves[di][dj];
            dj++;
        }
    }
    // d[di] = findHeuristicValue(goal, arr);
    double ftemp = findHeuristicValue(goal, arr) + gtemp;
    boolean check = false;
    Node temp = open;
    Node temp1 = closed;
    if (temp == null && temp1 == null) {
        open = insertNode(open, initial, arr, ftemp, gtemp);
    } else {
        while (temp != null) {
            boolean isDifferent = !isEqual(temp.father, arr);
            if (isDifferent == false) {
                if (temp.fa > ftemp) {
                    temp.ibp = initial;
                    temp.fa = ftemp;
                    temp.ga = gtemp;
        }
        check = true;
        break;
    }
    temp = temp.next;
}
if (check == false) {
    while (temp1 != null) {
        boolean isDifferent = !isEqual(temp1.father, arr);
        if (isDifferent == false) {
            if (temp1.fa > ftemp) {
                // System.out.println(ftemp + " " + gtemp + " " + temp1.fa + " " + temp1.ga);
                // System.out.println("updating closed");
                updateParent(closed, open, initial, temp1.ibp, gtemp);
            }
            check = true;
            break;
        }
        temp1 = temp1.next;
    }
}
if (check == false) {
    open = insertNode(open, initial, arr, ftemp, gtemp);
} else {
    check = false;
}
```

```
            }
            di++;
        }
        // matix with all generated child

//        System.out.println("\nFinal matrix: ");
//        for (int i = 0; i < mov; i++) {
//        for (int j = 0; j < 9; j++) {
//        System.out.print(moves[i][j] + " ");
//        }
////        System.out.print("-> distance: " + d[i]);
//        System.out.println();
//        }
        return open;
    }

    // successor state, (x,y) -> (p,q)
    static int[][] createMatrix(int[][] board, int x, int y, int p, int q, int fi) {
        int fj = 0;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (i == x && j == y) {
                    moves[fi][fj++] = board[p][q];
                } else if (i == p && j == q) {
                    moves[fi][fj++] = 0;
                } else {
```

```
                moves[fi][fj++] = board[i][j];
            }
        }
    }
    return moves;
}

// possible of current board positions
static void findPosition(int i, int j, int[][] arr) {
    int k = 0;
    if ((3 > (i - 1) && i - 1 >= 0) && (3 > j && j >= 0)) {
        arr[k][0] = i - 1;
        arr[k][1] = j;
        k++;
    }
    if (3 > i + 1 && i + 1 >= 0 && 3 > j && j >= 0) {
        arr[k][0] = i + 1;
        arr[k][1] = j;
        k++;
    }
    if ((3 > i && i >= 0) && (3 > (j + 1) && j + 1 >= 0)) {
        arr[k][0] = i;
        arr[k][1] = j + 1;
        k++;
    }
    if ((3 > i && i >= 0) && (3 > j - 1 && j - 1 >= 0)) {
        arr[k][0] = i;
        arr[k][1] = j - 1;
        k++;
    }
}

static void printQueue(Node node) {
    if (node == null) {
        System.out.println("\nList is empty");
        return;
    }
    Node ptr = node;
    while (ptr != null) {
        System.out.print("\nf(A): " + ptr.fa);
        System.out.println("\tg(A): " + ptr.ga);
        System.out.println("Parent Node");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(ptr.ibp[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("Child Node");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(ptr.father[i][j] + " ");
            }
            System.out.println();
        }
        ptr = ptr.next;
    }
}

public static void storePosition(int x, int y, int arr[][], int bpos[][]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if ((i == x && j == y) && ((i == 0 || i == 2) && (j == 0 || j == 2))) {
                mov = 2; // corner pos.
                findPosition(i, j, arr); // find x,y cords. to replace with 0
            } else if ((i == x && j == y) && (i == 1 && j == 1)) {
                mov = 4; // center
                findPosition(i, j, arr);
            } else if ((i == x && j == y)) {
                mov = 3;
                findPosition(i, j, arr);
            }
        }
    }
    // stored childs of bpos in moves[][]
    int fi = 0;
    for (int i = 0; i < mov; i++) {
        createMatrix(bpos, x, y, arr[i][0], arr[i][1], fi);
        fi++;
    }
}

static boolean isEqual(int[][] arr1, int[][] arr2) {
    boolean flag = true;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (arr1[i][j] != arr2[i][j]) {
                flag = false;
                break;
            }
        }
    }
    return flag;
}

    public static void printPath(Node CLOSED, int[][] initial, int[][] goal) {
    if (isEqual(initial, goal)) {
        System.out.println("Initial Node: f(A): " + CLOSED.fa + " and g(A): " + CLOSED.ga);
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(initial[i][j] + " ");
            }
            System.out.println();
        }
        return;
    }
    Node temp = CLOSED;
    while (temp != null && temp.next != null && !isEqual(temp.father, goal)) {
        temp = temp.next;
    }
    if (temp != null) {
        System.out.println("Node: f(A): " + temp.fa + " and g(A): " + temp.ga);
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(temp.father[i][j] + " ");
            }
            System.out.println();
        }
        printPath(CLOSED, initial, temp.ibp);
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter Initial state: ");
    int x = 0, y = 0; // 0-index
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
```

```
        board[i][j] = sc.nextInt();
        if (board[i][j] == 0) {
            x = i;
            y = j;
        }
    }
}

System.out.println("Enter Goal state: ");
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        goal[i][j] = sc.nextInt();
    }
}

Node OPEN = null;
int ga = 0;
int[][] a = { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };
OPEN = insertNode(OPEN, a, board, findHeuristicValue(goal, board) + ga, ga);
Node CLOSED = null;
int[][] arr = new int[4][2];
boolean flag = false;
while (OPEN != null) {
    Node current = OPEN;
    OPEN = OPEN.next;
    current.next = null;
    CLOSED = insertClosed(CLOSED, current.ibp, current.father, current.fa, current.ga);

    if (current.fa - current.ga == 0) {
        flag = true;
        break;
    }
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (current.father[i][j] == 0) {
                x = i;
                y = j;
            }
        }
    }
    storePosition(x, y, arr, current.father);
    OPEN = heuristic(OPEN, moves, goal, current.ga + 1, current.father, CLOSED);
}
if (flag == false) {
    System.out.println("Goal state can not be reached!");
} else {
    System.out.println("\nOPEN Linked List");
    printQueue(OPEN);
    System.out.println("\nCLOSED Linked List ");
    printQueue(CLOSED);
    System.out.println("\nPath {from goal to initial state}");
```

```
            printPath(CLOSED, CLOSED.father, goal);
        }
    }
}

class Node {
    int[][] ibp; // current bp
    int[][] father; // father
    int ga; // depth
    double fa; // f(A) = g(A) + h(A)
    Node next;

    Node(int[][] initial, int[][] movepos, double ftemp, int gtemp) {
        ibp = new int[3][3];
        father = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                ibp[i][j] = initial[i][j];
                father[i][j] = movepos[i][j];
            }
        }
        ga = gtemp;
        fa = ftemp;
        next = null;
    }
}
```

Output:

```
Enter Initial state:
1 2 3
5 6 0
7 8 4
Enter Goal state:
1 2 3
5 8 6
0 7 4

OPEN Linked List

f(A): 11.602325267042627        g(A): 3
Parent Node
1 2 3
5 8 6
7 0 4
Child Node
1 2 3
5 8 6
7 4 0

f(A): 12.38083151964686         g(A): 3
Parent Node
1 0 3
5 2 6
```

```
7 8 4

Child Node

0 1 3

5 2 6

7 8 4


f(A): 12.797958971132712        g(A): 3

Parent Node

1 0 3

5 2 6

7 8 4

Child Node

1 3 0

5 2 6

7 8 4


f(A): 12.94427190999916         g(A): 4

Parent Node

0 2 3

1 5 6

7 8 4

Child Node

2 0 3

1 5 6

7 8 4
```

```
f(A): 13.045361017187261        g(A): 2
Parent Node
1 2 3
5 6 4
7 8 0
Child Node
1 2 3
5 6 4
7 0 8

f(A): 14.295630140987 g(A): 4
Parent Node
0 1 2
5 6 3
7 8 4
Child Node
5 1 2
0 6 3
7 8 4

f(A): 14.832159566199232        g(A): 3
Parent Node
1 0 2
5 6 3
7 8 4
Child Node
```

```
1 6 2
5 0 3
7 8 4

f(A): 15.224972160321824        g(A): 4
Parent Node
1 2 3
7 5 6
0 8 4
Child Node
1 2 3
7 5 6
8 0 4

CLOSED Linked List

f(A): 9.486832980505138         g(A): 0
Parent Node
0 0 0
0 0 0
0 0 0
Child Node
1 2 3
5 6 0
7 8 4
```

```
f(A): 9.48528137423857        g(A): 1
Parent Node
1 2 3
5 6 0
7 8 4
Child Node
1 2 0
5 6 3
7 8 4

f(A): 9.602325267042627       g(A): 1
Parent Node
1 2 3
5 6 0
7 8 4
Child Node
1 2 3
5 6 4
7 8 0

f(A): 10.246211251235321      g(A): 2
Parent Node
1 2 0
5 6 3
7 8 4
Child Node
```

```
1 0 2

5 6 3

7 8 4


f(A): 11.12403840463596          g(A): 3

Parent Node

1 0 2

5 6 3

7 8 4

Child Node

0 1 2

5 6 3

7 8 4


f(A): 11.677078252031311         g(A): 1

Parent Node

1 2 3

5 6 0

7 8 4

Child Node

1 2 3

5 0 6

7 8 4


f(A): 11.16515138991168          g(A): 2

Parent Node
```

```
1 2 3
5 0 6
7 8 4
Child Node
1 2 3
0 5 6
7 8 4

f(A): 6.741657386773941          g(A): 3
Parent Node
1 2 3
0 5 6
7 8 4
Child Node
1 2 3
7 5 6
0 8 4

f(A): 11.486832980505138         g(A): 2
Parent Node
1 2 3
5 0 6
7 8 4
Child Node
1 0 3
5 2 6
```

```
7 8 4

f(A): 11.717797887081348        g(A): 3
Parent Node
1 2 3
0 5 6
7 8 4
Child Node
0 2 3
1 5 6
7 8 4

f(A): 11.899494936611665        g(A): 2
Parent Node
1 2 3
5 0 6
7 8 4
Child Node
1 2 3
5 8 6
7 0 4

f(A): 3.0          g(A): 3
Parent Node
1 2 3
5 8 6
```

```
7 0 4
Child Node
1 2 3
5 8 6
0 7 4

Path {from goal to initial state}
Node: f(A): 3.0 and g(A): 3
1 2 3
5 8 6
0 7 4
Node: f(A): 11.899494936611665 and g(A): 2
1 2 3
5 8 6
7 0 4
Node: f(A): 11.677078252031311 and g(A): 1
1 2 3
5 0 6
7 8 4
Initial Node: f(A): 9.486832980505138 and g(A): 0
1 2 3
5 6 0
7 8 4

Process finished with exit code 0
```