Assignment: Implement Eight puzzle problem using A star algorithm
Batch:A1
Roll no:3330

```
package aiml;
import java.util.Comparator;
public class Node implements Comparator<Node> {

       int f, g, h;
       Node parent;
       int grid[][] = new int[3][3];

       public Node()
       {

       }

       public Node(int grid[][],int g, int h, Node parent)
        {
          f = g+h;
          this.g=g;
          this.h=h;
          this.parent=parent;
          this.grid=grid;
        }

        public void display()
        {
            int i, j;

            for(i=0;i<3;i++){
                 for(j=0;j<3;j++){
                    System.out.print(grid[i][j]+" ");
                 }
                  System.out.println();
             }

        System.out.println("g(n)="+g);
        System.out.println("h(n)="+h);
        System.out.println("f(n)="+f);
        System.out.println("**************");

        }

        @Override
        public int compare(Node n1, Node n2)
        {
        if(n1.f<n2.f)
        return -1;

        if(n1.f>n2.f)
        return 1;


        return 0;
        }
}

package aiml;
import java.util.Scanner;
import java.util.Stack;
import java.util.ArrayList;
import java.util.PriorityQueue;
public class EightPuzzle {

       public static void AStar(int input[][], int goal[][]) {

             ArrayList<int[][]> visited = new ArrayList<int[][]>();
             int i, j;
             int direction[][] = {{1,0},{-1,0},{0,-1},{0,1}};
             int flag=0;


             Node goalState = new Node();
             Node empty = new Node();//for parent
             Node start = new Node(input,0,findHValue(input, goal),empty);
             //constructor of Node(grid,g,h,parent)

             int size = 1000;

             PriorityQueue<Node> pq = new PriorityQueue<Node>(size, new Node());

             pq.add(start);
             visited.add(input);

             while(pq.isEmpty()==false)
              {

                  Node cur = pq.remove(); //get the state with minimum f(n) value
                  if(isequal(cur.grid,goal)){
                     flag = 1;
                     goalState = cur;
                     printStates(goalState, start);
                     break;
                   }
                  int itr=0;
                  //calculate the next possible states
                  int blank[]=findBlank(cur.grid);
                  int x=blank[0];
                  int y=blank[1];
                     while(itr<4)
                  {
                    int next[][] = new int[3][3];
                    next = copyMatrix(cur.grid, next);
                     i = x + direction[itr][0];
                     j = y + direction[itr][1];

                     if(i>=0 && j>=0 && i<3 && j<3){
                         next[x][y] = next[i][j];//move white space
                         next[i][j] = 0;
```

```
                       }

                     if(visited.contains(next)==false) { //if not already visited
                         Node child = new 
Node(next,cur.g+1,findHValue(next,goal),cur);
                         pq.add(child); //add next to the priority queue
                         visited.add(next); //mark it as visited
                      }

                     if(visited.contains(next)){//if already visited
                             Node child = new 
Node(next,cur.g+1,findHValue(next,goal),cur);
                            pq.add(child); //add next to the priority queue
                     }
                     itr++;
                  }//close while(itr<4)
              }//close while loop

             if(flag == 0)
              {
              System.out.println("State not reachable");
              }

      }

      public static void printStates(Node goalState, Node start)
      {
        Stack<Node> path = new Stack<Node>();
        Node s = goalState;

        while(s.grid!=start.grid)
         {
            path.push(s);
            s = s.parent;
         }
        path.push(start);

        System.out.println("The state transitions are as follows: ");

        while(path.isEmpty()==false){
           path.pop().display();
        }

        System.out.println("Reached goal state!");
      }

      //find the number of misplaced tiles
      public static int findHValue(int[][] input,int[][] goal)
       {
           int h=0;
             for(int i=0;i<3;i++){
                for(int j=0;j<3;j++){
                    if(input[i][j]!=0 && input[i][j]!=goal[i][j])
                       h++;
                }
             }
            return h;
       }

      //check if 2 matrices are equal
      public static boolean isequal(int[][] input,int[][] goal)
       {
             for(int i=0;i<3;i++){
                  for(int j=0;j<3;j++){
                        if(input[i][j]!=goal[i][j])
                            return false;
                  }
             }

           return true;
       }


      //copy matrix
      public static int[][] copyMatrix(int grid[][], int next[][])
       {
          for(int i=0;i<3;i++){

             for(int j=0;j<3;j++){
                 next[i][j] = grid[i][j];
              }
           }
            return next;
       }

      //find blank space location
      public static int[] findBlank(int[][] input)
       {
          int index[]=new int[2];
             for(int i=0;i<3;i++){
                 for(int j=0;j<3;j++){
                    if(input[i][j]==0){
                       index[0]=i;
                       index[1]=j;
                        break;
                   }
                 }
              }
             return index;
       }


      public static void main(String args[]) {

                   Scanner sc = new Scanner(System.in);

                   int i, j;

                   int initial[][] = new int[3][3]; //start state
                   int goal[][] = new int[3][3]; //Goal state
```

```
                   System.out.println("Enter the start state: ");
                   for(i=0;i<3;i++){
                      for(j=0;j<3;j++){
                             initial[i][j] = sc.nextInt();
                    }
                   }
                   System.out.println("Start state accepted");
                   System.out.println("Enter the goal state: ");
                   for(i=0;i<3;i++){
                       for(j=0;j<3;j++){
                          goal[i][j] = sc.nextInt();
                       }
                   }
                   System.out.println("Goal state accepted");

                   AStar(initial, goal);
      }
}
```

OUTPUT

```
Enter the start state:
1
2
3
0
4
6
7
5
8
Start state accepted
Enter the goal state:
1
2
3
4
5
6
7
8
0
Goal state accepted
The state transitions are as follows:
1 2 3
0 4 6
7 5 8
g(n)=0
h(n)=3
f(n)=3
**************
1 2 3
4 0 6
7 5 8
g(n)=1
h(n)=2
```

```
f(n)=3
**************
1 2 3
4 5 6
7 0 8
g(n)=2
h(n)=1
f(n)=3
**************
1 2 3
4 5 6
7 8 0
g(n)=3
h(n)=0
f(n)=3
**************
Reached goal state!
```