```
Enter File Name:
C:\Users\Jamie\Desktop\AI Files\file4
11 2 5 10
3 0 9 8
6 4 7 1
15 12 13 14

11 2 5 10
0 3 9 8
6 4 7 1
15 12 13 14

11 2 5 10
6 3 9 8
0 4 7 1
15 12 13 14

11 2 5 10
6 3 9 8
4 0 7 1
15 12 13 14

11 2 5 10
6 3 9 8
4 12 7 1
15 0 13 14

11 2 5 10
6 3 9 8
4 12 7 1
15 13 0 14

11 2 5 10
6 3 9 8
4 12 0 1
15 13 7 14

11 2 5 10
6 3 0 8
4 12 9 1
15 13 7 14

11 2 5 10
6 3 8 0
4 12 9 1
15 13 7 14

11 2 5 0
6 3 8 10
4 12 9 1
15 13 7 14

11 2 0 5
6 3 8 10
```

```
 55 4 12 9 1
 56 15 13 7 14
 57
 58 ------------------------------------------------------------------
    -
 59 Enter File Name:
 60 C:\Users\Jamie\Desktop\AI Files\file3
 61 Running Algorithm
 62 0 4 8
 63 1 5 2
 64 6 3 7
 65
 66 1 4 8
 67 0 5 2
 68 6 3 7
 69
 70 1 4 8
 71 5 0 2
 72 6 3 7
 73
 74 1 0 8
 75 5 4 2
 76 6 3 7
 77
 78 0 1 8
 79 5 4 2
 80 6 3 7
 81
 82 5 1 8
 83 0 4 2
 84 6 3 7
 85
 86 5 1 8
 87 4 0 2
 88 6 3 7
 89
 90 5 1 8
 91 4 2 0
 92 6 3 7
 93
 94 5 1 8
 95 4 2 7
 96 6 3 0
 97
 98 5 1 8
 99 4 2 7
100 6 0 3
101
102 5 1 8
103 4 2 7
104 0 6 3
105
106 5 1 8
107 0 2 7
```

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

```
 1 import java.util.LinkedList;
 2
 3 //Solves Sliding Puzzle
 4 public class Sliding {
 5
 6     //Inner Board class
 7     private class Board{
 8         private int[][] array;                  //Board array
 9         private Board parent;                    //Parent board
10
11         //Constructor
12         private Board(int[][] array, int m, int n) {
13             this.array = new int[m][n];
14
15             for (int i = 0; i < m; i++)
16                 for (int j = 0; j < n; j++)
17                     this.array[i][j] = array[i][j];
18
19             this.parent = null;
20         }
21     }
22
23     private Board initial;
24     private Board goal;
25     private int m;
26     private int n;
27
28     //Constructor
29     public Sliding(int[][] initial, int [][] goal, int m, int n){
30         this.initial = new Board(initial,m,n);
31         this.goal = new Board(goal,m,n);
32         this.m = m;
33         this.n = n;
34     }
35
36     //Solve puzzle
37     public void solve(){
38         LinkedList<Board> openList = new LinkedList<Board>();
39         LinkedList<Board> closedList = new LinkedList<Board>();
40
41         openList.addFirst(initial);                 //add initial
   board to open list
42
43         while(!openList.isEmpty()){
44             Board board = openList.removeFirst();   //add boards that
   have been looked at
45
46             if(complete(board)){                    //If goal
47                 displayPath(board);                 //Display path to
   goal
48                 return;                             //Stop
49             }else{
50                 LinkedList<Board> children = generate(board);   //
   Create children
```

```
51
52                 for(int i = 0; i < children.size(); i ++){
53                     Board child = children.get(i);
54                     if(!exists(child, openList) && !exists(child,
   closedList))
55                         openList.addLast(child);        //Add if not
   in open or closed lists
56                 }
57             }
58         }
59         System.out.println("No Solution");
60     }
61
62
63     //Creates children of a board
64     private LinkedList<Board> generate(Board board){
65         int i = 0;
66         int j = 0;
67         boolean found = false;
68
69         for(i = 0; i < m; i ++){                    //find location
   of empty slot
70             for(j = 0; j < n; j++)
71                 if(board.array[i][j] == 0){
72                     found = true;
73                     break;
74                 }
75             if(found)
76                 break;
77         }
78
79         boolean north, south, east, west;           //decide whether
   empty slot
80         north = i == 0 ? false : true;              // has N, S, E, W
    neighbors
81         south = i == m - 1 ? false : true;
82         east = j == n - 1 ? false : true;
83         west = j == 0 ? false : true;
84
85         LinkedList<Board> children = new LinkedList<Board>(); //list
   of children
86
87         if(north) children.addLast(createChild(board, i , j, 'N'));
88         if(south) children.addLast(createChild(board, i , j, 'S'));
89         if(east) children.addLast(createChild(board, i , j, 'E'));
90         if(west) children.addLast(createChild(board, i , j, 'W'));
91
92         return children;
93     }
94
95     //Swaps places to create children
96     private Board createChild(Board board, int i , int j, char
   direction){
97         Board child = copy(board);
```

```
 98
 99         if(direction == 'N'){
100             child.array[i][j] = child.array [i-1][j];
101             child.array[i-1][j] =0;
102         }else if(direction == 'S'){
103             child.array[i][j] = child.array [i+1][j];
104             child.array[i+1][j] =0;
105         }else if(direction == 'E'){
106             child.array[i][j] = child.array [i][j+1];
107             child.array[i][j+1] =0;
108         }else{
109             child.array[i][j] = child.array [i][j-1];
110             child.array[i][j-1] =0;
111         }
112 
113         child.parent = board;                   //assign parent
114         return child;
115     }
116 
117     //Creates copy of board
118     private Board copy(Board board){
119         return new Board(board.array, m, n);
120     }
121 
122     //Decides if board is complete
123     private boolean complete(Board board){
124         return identical(board, goal);
125     }
126 
127     //If a board exists in a list
128     private boolean exists(Board board, LinkedList<Board> list){
129         for (int i = 0; i < list.size(); i++)       //compare board
    with each element
130             if(identical(board, list.get(i)))
131                 return true;
132         return false;
133     }
134 
135     //if two boards identical
136     private boolean identical(Board p, Board q){
137         for(int i = 0; i < m; i++)
138             for(int j = 0; j < n; j++)
139                 if(p.array[i][j] != q.array[i][j])
140                     return false;       //If there is a mismatch then
     false
141 
142         return true;
143     }
144 
145     //Displays path form initial to current board
146     private void displayPath(Board board){
147         LinkedList<Board> list = new LinkedList<Board>();
148 
149         Board pointer = board;      //start at current
```

```
150         while(pointer != null){
151             list.addFirst(pointer);     //add to list
152 
153             pointer = pointer.parent;
154         }
155 
156         for(int i = 0; i < list.size(); i++)
157             displayBoard(list.get(i));
158     }
159 
160     //Displays board
161     private void displayBoard(Board board){
162         for(int i = 0; i < m; i++){
163             for(int j = 0; j < n; j++ )
164                 System.out.print(board.array[i][j] + " ");
165             System.out.println();
166         }
167         System.out.println();
168     }
169 }
170 
```

```
 1 public class HardTester {
 2     public static void main(String[] args){
 3         int[][] initial = {{5,7,1},{2,0,8},{4,6,3}};
 4         int[][] goal = {{5,0,1},{4,7,8},{6,2,3}};
 5 
 6         Sliding s = new Sliding(initial,goal,3, 3);
 7         s.solve();
 8     }
 9 }
10 
```

```
 1 import java.util.Scanner;
 2 import java.io.*;
 3
 4 public class SlidingTester {
 5     public static void main(String[] args)throws IOException{
 6         Scanner keyIn = new Scanner(System.in);
 7
 8         System.out.println("Enter File Name:");
 9         String fileName = keyIn.nextLine();
10
11         File file = new File(fileName);
12         Scanner sc = new Scanner(file);
13
14         String firstLine[] = sc.nextLine().split(" ");
15
16         int m = Integer.parseInt(firstLine[0]);
17
18         int n = Integer.parseInt(firstLine[1]);
19
20         sc.nextLine();
21         int[][] initial = new int[m][n];
22         for(int i = 0; i < m; i++){
23             String line[] = sc.nextLine().split("\\s+");
24             for(int j = 0; j < n; j++ ){
25                 initial[i][j] = Integer.parseInt(line[j]);
26             }
27         }
28
29         sc.nextLine();
30         int[][] goal = new int[m][n];
31         for(int i = 0; i < m; i++){
32             String line[] = sc.nextLine().split("\\s+");
33             for(int j = 0; j < n; j++ ){
34                 goal[i][j] = Integer.parseInt(line[j]);
35             }
36         }
37         Sliding s = new Sliding(initial, goal, m, n);
38         s.solve();
39     }
40 }
41
```