#### **Assignment 2**

#### **Problem Statement**:

#### Solve 8-puzzle problem using A\* algorithm. Assume any initial configuration and define goal configuration clearly.

## Theory

### 8 Puzzle Problem

#### The puzzle consists of an area divided into a grid, 3 by 3 for the 8-puzzle, 4 by 4 for the 15-puzzle. On each grid square is a tile, expect for one square which remains empty. Thus, there are eight tiles in the 8-puzzle and 15 tiles in the 15-puzzle. A tile that is next to the empty grid square can be moved into the empty space, leaving its previous position empty in turn. Tiles are numbered, 1 through 8 for the 8-puzzle, so that each tile can be uniquely identified.

#### The aim of the puzzle is to achieve a given configuration of tiles from a given (different) configuration by sliding the individual tiles around the grid as described above. This problem can be solved by searching for a solution, which

#### is a sequence of actions (tile moves) that leads from the initial state to the goal state. Two possible states of the 8-puzzle are shown in figure 1. The state on the right is a typical goal state. The state on the left is a configuration that represents a worst case: transforming this state into the goal state requires at least 31 actions, which is the diameter of the search space. For search algorithms the problem is often to find the shortest solution, that is, one which consists of the least number of tile moves.

### A Solution

#### We now describe an algorithmic solution to the problem that illustrates a general artificial intelligence methodology known as the A\* search algorithm. We define a state of the game to be the board position, the number of moves made to reach the board position, and the previous state. First, insert the initial state (the initial board, 0 moves, and a null previous state) into a priority queue. Then, delete from the priority queue the state with the minimum priority, and insert onto the priority queue all neighboring states (those that can be reached in one move).

#### Repeat this procedure until the state dequeued is the goal state. The success of this approach hinges on the choice of priority function for a state. We consider two priority functions:

#### Hamming priority function. The number of blocks in the wrong position, plus the number of moves made so far to get to the state. Intuitively, a state with a small number of blocks in the wrong position is close to the

•

#### goal state, and we prefer a state that have been reached using a small number of moves.

#### Manhattan priority function. The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state

•

#### . For example, the Hamming and Manhattan priorities of the initial state below are 5 and 10, respectively.

#### We make a key oberservation: to solve the puzzle from a given state on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using either the Hamming or Manhattan priority function. (For Hamming priority, this is true because each block that is out of place must move at least once to reach its goal position. For Manhattan priority, this is true because each block must move its Manhattan distance from its goal position. Note that we do not count the blank tile when computing the Hamming or Manhattan priorities.)

#### Consequently, as soon as we dequeue a state, we have not only discovered a sequence of moves from the initial board to the board associated with the state, but one that makes the fewest number of moves. (Challenge for the mathematically inclined: prove this fact.)

## Source Code

import java.io.\*; import java.util.\*; classinfo

{

inth,g,f;

String s=newString(); info(String s,inth,intg,intf)

{

this.s=s; this.h=h; this.g=g; this.f=f;

}

}

classepuzzle

{

intn; charstart[][],goal[][],current[][]; Scanner sc;

ArrayList<info> closed =newArrayList<info>(); ArrayList<info> open =newArrayList<info>(); epuzzle()

{

sc=newScanner(System.in);

}

publicvoidcequals()

{

for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

current[i][j]=start[i][j];

}

}

}

publicvoidinput()

{

System.out.println("\nEnterthesizeofinputmatrix"); n=sc.nextInt();

start=newchar[n][n]; goal=newchar[n][n]; current=newchar[n][n];

System.out.println("\nEnterthestartmatrix"); for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

start[i][j]=sc.next().charAt(0);

}

}

System.out.println("\nEnterthegoalmatrix"); for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

goal[i][j]=sc.next().charAt(0);

}

}

}

publicvoidoutput()

{

System.out.println("Thestartmatrixis:\n"); for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

System.out.print(start[i][j]+"\t");

}

System.out.println("\n");

}

System.out.println("Thegoalmatrixis:\n");

for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

System.out.print(goal[i][j]+"\t");

}

System.out.println("\n");

}

}

publicintfind()

{

intcount=0; for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

if(current[i][j]!=goal[i][j]) count++;

}

}

returncount;

}

publicvoidchanges(String s,intith ,intjth,chardir)

{

if(dir==’U’)

{

current[ith][jth]=current[ith-1][jth]; current[ith-1][jth]=’s’;

String k=s.concat("U"); open.add(newinfo(k,find(),k.length(),find()+k.length())); current[ith-1][jth]=current[ith][jth]; current[ith][jth]=’s’;

}

if(dir==’L’)

{

current[ith][jth]=current[ith][jth-1]; current[ith][jth-1]=’s’;

String k=s.concat("L"); open.add(newinfo(k,find(),k.length(),find()+k.length())); current[ith][jth-1]=current[ith][jth]; current[ith][jth]=’s’;

}

if(dir==’R’)

{

current[ith][jth]=current[ith][jth+1]; current[ith][jth+1]=’s’;

String k=s.concat("R"); open.add(newinfo(k,find(),k.length(),find()+k.length())); current[ith][jth+1]=current[ith][jth];

current[ith][jth]=’s’;

}

if(dir==’D’)

{

current[ith][jth]=current[ith+1][jth]; current[ith+1][jth]=’s’;

String k=s.concat("D"); open.add(newinfo(k,find(),k.length(),find()+k.length())); current[ith+1][jth]=current[ith][jth]; current[ith][jth]=’s’;

}

}

publicvoidgetdirections(String s,intith,intjth)

{

if(ith==0&&jth==0)

{

changes(s,ith,jth,’R’);

changes(s,ith,jth,’D’);

}

elseif(ith==0&&jth==n-1)

{

changes(s,ith,jth,’L’);

changes(s,ith,jth,’D’);

}

elseif(ith==n-1&&jth==0)

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’R’);

}

elseif(ith==n-1&&jth==n-1)

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’L’);

}

elseif(ith==0)

{

changes(s,ith,jth,’R’);

changes(s,ith,jth,’D’);

changes(s,ith,jth,’L’);

}

elseif(ith==n-1)

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’R’);

changes(s,ith,jth,’L’);

}

elseif(jth==0)

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’D’);

changes(s,ith,jth,’R’);

}

elseif(jth==n-1)

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’D’);

changes(s,ith,jth,’L’);

}

else

{

changes(s,ith,jth,’U’);

changes(s,ith,jth,’D’);

changes(s,ith,jth,’L’);

changes(s,ith,jth,’R’);

}

}

publicvoidgets(String s)

{

intith=0,jth=0; for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

if(current[i][j]==’s’)

{

ith=i; jth=j; break;

}

}

}

getdirections(s,ith,jth);

}

publicvoidput(String s)

{

if(s.length()!=0)

{

intith=0,jth=0;

for(inti=0;i<s.length();i++)

{

for(intp=0;p<n;p++)

{

for(intq=0;q<n;q++)

{

if(current[p][q]==’s’)

{

ith=p;

jth=q; break;

}

}

}

if(s.charAt(i)==’U’)

{

current[ith][jth]=current[ith-1][jth]; current[ith-1][jth]=’s’;

}

if(s.charAt(i)==’L’)

{

current[ith][jth]=current[ith][jth-1]; current[ith][jth-1]=’s’;

}

if(s.charAt(i)==’R’)

{

current[ith][jth]=current[ith][jth+1]; current[ith][jth+1]=’s’;

}

if(s.charAt(i)==’D’)

{

current[ith][jth]=current[ith+1][jth]; current[ith+1][jth]=’s’;

}

}

}

gets(s);

}

publicvoidsort()

{

for(inti=0;i<open.size()-1;i++)

{

for(intj=0;j<open.size()-1;j++)

{

if((open.get(j).f)>(open.get(j+1).f))

{

info c=open.get(j); open.remove(j); open.add(j+1,c);

}

}

}

}

publicvoidsolve()

{

String k=""; cequals(); intch=0;

open.add(newinfo("",find(),k.length(),find()+k.length())); while(open.get(0).h!=0)

{

put(open.get(0).s); closed.add(open.get(0)); open.remove(0);

sort(); cequals();

}

System.out.println("Thepathfromstarttogoalis\n=>"); System.out.println(open.get(0).s); System.out.println("Thenumberofstepsfromstarttogoalis

\n=>");

System.out.println(open.get(0).s.length());

}

publicstaticvoidmain(String args[])

{

epuzzle obj=newepuzzle(); obj.input();

obj.output();

obj.solve();

}

}

## Output

## 

## Conclusion

#### The 8 puzzle problem is analysed and a solution is proposed, wherein an appro- priate solution is found.