**Assignment No: 6**

**Title**: 8 Puzzle Problem.

**Problem Statement**: Solve 8-puzzle problem using algorithm. Assume any initial configuration and define goal configuration clearly.

**Objectives**: Understanding and implementation 8-puzzle problem using algorithm.

**Outcome**: Students will be able to understand the logic of implementation for 8-puzzle problem.

### Theory:

algorithm is a best first search algorithm in which cost associated with a node is f(n) = g(n) + h(n) where

g(n) = cost of path from initial state to node „n‟. h(n) = cost from „n‟ to goal node.

algorithm guides an optimal path to goal if the heuristic function h(n) is admissible, meaning it never overestimates actual cost.

Example: Since airline distance never overestimates actual highway distance.

For 8-puzzle problem, algorithm using these evaluation functions can find optimal solutions. In addition, makes the most efficient use of the given heuristic function in the following sense: among all shortest path algorithms using the given heuristic function h(n), expands fewest number of nodes.

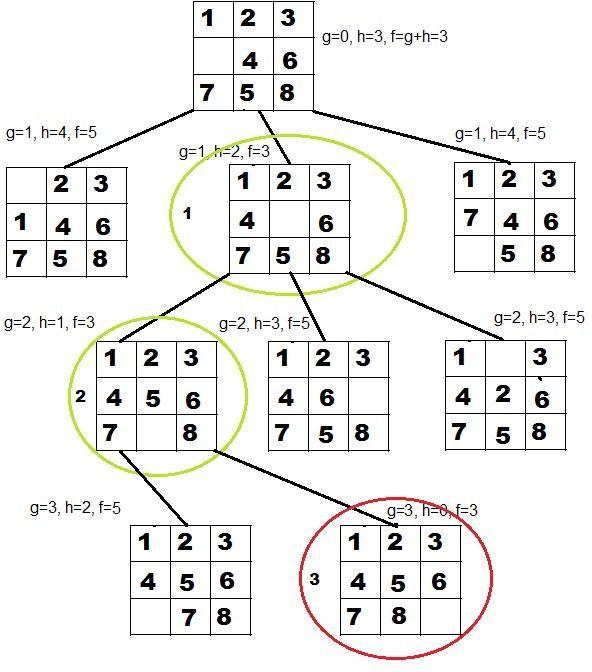
algorithm is very good search method but with complexity problems. To implement such a graph search procedure, we will need to use two lists of nodes.

1. OPEN: Nodes that have been generated and have the heuristic function applied to them but which has not been examined. OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
2. CLOSED: Nodes that have already been examined; we have to keep these nodes in memory. If we want to search a graph rather than a tree, since we need to check whether it has been generated before.

### How A\* solves the 8-Puzzle problem ?

We first move the empty space in all the possible directions in the start state and calculate **f-score** for each state. This is called expanding the current state. After expanding the current state, it is pushed into the **closed** list and the newly generated states are pushed into the **open** list. A state with the least f-score is selected and expanded again. This process continues until the goal state occurs as the current

state. Basically, here we are providing the algorithm a measure to choose its actions. The algorithm chooses the best possible action and proceeds in that path. This solves the issue of generating redundant child states, as the algorithm will expand the node with the least **f-score**.



### algorithm for 8-puzzle problem:

* 1. Put the start node as OPEN.
  2. If OPEN is empty, exit with failure.
  3. Remove from OPEN and place on CLOSED a node „n‟ has minimum „f‟.
  4. If „n‟ is a goal node exit successfully with a solution obtained by tracking back the pointers from „n‟ to „s‟.
  5. Otherwise expand „n‟ generating its children and directing pointers from each child node to „n‟.
  6. Evaluate h(n‟) and compute f(n‟) = g(n‟) + h(n‟)

= g(n) + c (n, n‟) + h(n)

* 1. If n‟ is already on OPEN or CLOSED compare its new f with the old f and attach the lowest to n‟.
  2. Go to step 2.

algorithm is commonly used for the common path finding problem in applications such as games but was originally designed as a general graph traversal algorithm.

### Code:

Board.java :-

package eightpuzzle;

import java.util.Scanner;

import javax.swing.JOptionPane;

//board class for eight puzzle matrix

public class Board {

private String board[][];

private int blankX,blankY; // co-ordinates for blank tile

public Board()

{

this.board = new String[3][3];

}

public Board(Board b) //constructor to initialise Board

{

this.board = b.board;

this.blankX = b.blankX;

this.blankY = b.blankY;

}

public void initBoard() //initialize the board

{

Scanner inp = new Scanner(System.in);

System.out.println("\nEnter one tile as '-' ie. Blank tile\n");

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

{

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

{

board[i][j] = JOptionPane.showInputDialog("Enter the value of tile ["+i+"]["+(j)+"] : ");

if(board[i][j].equals("-")) //store the location of blank symbol

{

blankX=i;

blankY=j;

}

}

}

}

public String[][] getBoard()

{

return board;

}

public void setBoard(String[][] board) // Set the board puzzle matrix

{

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

{

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

{

this.board[i][j] = board[i][j];

}

}

}

public int getBlankX()

{

return blankX;

}

public int getBlankY()

{

return blankY;

}

public void setBlankX(int x)

{

blankX = x;

}

public void setBlankY(int y)

{

blankY = y;

}

public void display()

{

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

{

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

{

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

}

System.out.println();

}

}

public Board nextMove(int gn, Board goal) //method to check possible moves and select optimum

{

Board temp = new Board();

Board next = new Board();

int minFn = 999;

System.out.println("\nPossible moves are : ");

if(blankY>0) // Condition for possible left move

{

temp.setBoard(board);

temp.swap(blankX, blankY, blankX, blankY-1); // Swap blank tile

int fn = (temp.getHn(goal)+gn); // Calculate fn = hn + gn

System.out.println("\nFor Fn = "+fn+" : ");

temp.display();

if(fn < minFn) // Check for minimum fn and set the next board accordingly

{

minFn = fn;

next.setBoard(temp.board);

next.setBlankX(blankX);

next.setBlankY(blankY-1);

}

}

if(blankY<2) // Condition for possible right move

{

temp.setBoard(board);

temp.swap(blankX, blankY, blankX, blankY+1);

int fn = (temp.getHn(goal)+gn);

System.out.println("\nFor Fn = "+fn+" : ");

temp.display();

if(fn < minFn)

{

minFn = fn;

next.setBoard(temp.board);

next.setBlankX(blankX);

next.setBlankY(blankY+1);

}

}

if(blankX>0) // Condition for possible up move

{

temp.setBoard(board);

temp.swap(blankX, blankY, blankX-1, blankY);

int fn = (temp.getHn(goal)+gn);

System.out.println("\nFor Fn = "+fn+" : ");

temp.display();

if(fn < minFn)

{

minFn = fn;

next.setBoard(temp.board);

next.setBlankX(blankX-1);

next.setBlankY(blankY);

}

}

if(blankX<2) // Condition for possible down move

{

temp.setBoard(board);

temp.swap(blankX, blankY, blankX+1, blankY);

int fn = (temp.getHn(goal)+gn);

System.out.println("\nFor Fn = "+fn+" : ");

temp.display();

if(fn < minFn)

{

minFn = fn;

next.setBoard(temp.board);

next.setBlankX(blankX+1);

next.setBlankY(blankY);

}

}

return next; // return board with min fn

}

public void swap(int i1, int j1, int i2, int j2) // Swap tile values

{

String temp = board[i1][j1];

board[i1][j1] = board[i2][j2];

board[i2][j2] = temp;

}

public boolean equals(Board b) // check for board equality

{

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

{

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

{

if(!this.board[i][j].equals(b.board[i][j]))

{

return false;

}

}

}

return true;

}

public int getHn(Board goal) // get hn by Hamming method

{

int hn = 0;

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

{

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

{

if(!this.board[i][j].equals(goal.board[i][j]))

{

hn++;

}

}

}

return hn;

}

}

EightPuzzle.java :-

package eightpuzzle;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Vector;

import Board;

public class EightPuzzle {

/\*\*

\* @param args the command line arguments

\*/

private int gn=0; // Initialize gn ie. no. of moves to 0

private Board start;

private Board goal;

public void initStart() //Accept and display start board

{

System.out.println("\n\n Enter start Board : ");

start=new Board();

start.initBoard();

System.out.println("\n\nThe given start board is : ");

start.display();

}

public void initGoal() //Accept and display goal board

{

System.out.println("\n\n Enter goal Board : ");

goal=new Board();

goal.initBoard();

System.out.println("\n\nThe given goal board is : ");

goal.display();

}

public void solve() // Solve puzzle using A\* algorithm

{

Board cur = start;

while(true)

{

System.out.println("\n\nBoard after "+gn+" moves : ");

cur.display();

if(cur.equals(goal)) //Check if goal is achieved nad return

{

System.out.println("\nGoal state achieved.");

return;

}

gn++; // Increment gn as per moves

cur = cur.nextMove(gn, goal); // get the board after next move

}

}

public static void main(String[] args) {

// TODO code application logic here

EightPuzzle ep = new EightPuzzle(); // Instantiate and solve the puzzle

ep.initStart();

ep.initGoal();

System.out.println("\n\nThe board is solved as : \n");

ep.solve();

}

}

/\*

Output:

Enter start Board :

Enter one tile as '-' ie. Blank tile

The given start board is :

- a c

h b d

g f e

Enter goal Board :

Enter one tile as '-' ie. Blank tile

The given goal board is :

a b c

h - d

g f e

The board is solved as :

Board after 0 moves :

- a c

h b d

g f e

Possible moves are :

For Fn = 3 :

a - c

h b d

g f e

For Fn = 5 :

h a c

- b d

g f e

Board after 1 moves :

a - c

h b d

g f e

Possible moves are :

For Fn = 5 :

- a c

h b d

g f e

For Fn = 5 :

a c -

h b d

g f e

For Fn = 2 :

a b c

h - d

g f e

Board after 2 moves :

a b c

h - d

g f e

Goal state achieved.

BUILD SUCCESSFUL (total time: 36 seconds)

\*/

### Conclusion:

Hence, 8-puzzle problem is solved using algorithm.