|  |  |  |  |
| --- | --- | --- | --- |
|  | | Project report |  |
|  | |  |  |
| **Solving 8-puzzle using A\* algorithm** | | | |
|  | Project Guidance By | |  |
|  | **Dr. Dewan Ahmed** | |  |
|  | | Team details |  |

|  |  |  |
| --- | --- | --- |
| **Priyanka Pandit Jadhav** | **Maheswara Rao Lekkala** | **Bharadwaj Tirunagaru** |
| **pjadhav7@uncc.edu** | **mlekkala@uncc.edu** | **btirunag@uncc.edu** |
| SID: 801167642 | SID: 801167702 | SID: 801167910 |
| Ph No: 704-905-1666 | Ph No: 980-833-5010 | Ph No: 980-729-4465 |

**Date: 02/21/2020**

# **AIM**

To solve 8-puzzle problem using A\* algorithm using two heuristics Manhattan distance and Misplaced Tiles.

**PROBLEM STATEMENT**

Implement A\* search algorithm and apply it to 8-puzzle problem, provide state space representation, operators, g (cost) and two heuristic functions of the 8-puzzle problem.

**8- PUZZLE PROBLEM**

The 8-puzzle consists of a 3\*3 grid area. The grids in the puzzle are called tiles and each tile contains a number ranged from 1 to 8 in order to identify each tile uniquely. Tile adjacent to empty grid can be moved to the empty space. The tiles needs to be moved until the goal is reached.

**PROBLEM FORMULATION**

**Goal:** Goal State is initially given.

**States:** Integer locations of tiles.

**Actions:** Move the blank tile UP, DOWN, LEFT or RIGHT.

**Performance:** Number of total moves in the solution

# **A\* ALGORITHM**

A\* is an informed search algorithm used in path findings and graph traversals. It is a combination of uniform cost search and best first search, which avoids expanding expensive paths. A\* star uses admissible heuristics which is optimal as it never over-estimates the path to goal. The evaluation function A\* star uses for calculating distance is

**f(n) = g(n) + h(n)**

**g(n) = cost so far to reach n**

**h(n) = estimated cost from n to goal**

**f(n) = estimated total cost of path through n to goal**

**Heuristic Functions**

The heuristic function is a way to inform the search regarding the direction to a goal. It provides an information to estimate which neighboring node will lead to the goal. The two heuristic functions that we considered for solving 8-puzzle problem are

**Misplaced Tile**

The number of misplaced tiles calculated by comparing the current state and goal state.

**Manhattan Distance**

The distance between two tiles measured along the axes of right angles. It is the sum of absolute values of differences between goal state (i, j) coordinates and current state (l, m) coordinates respectively, i.e. |i - l|+ |j - m|

**Sample 8-Puzzle solved using A\* search Manhattan distance Initial State Goal State**

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |

**Level: 0**

1

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = g(n)+h(n) = 0+2=2**

**Level: 1**

2 3

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 |  |
| 7 | 8 | 6 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 |  | 2 |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = 1+1=2 f(n) = 1+3=4**

**Level: 2**

4 5 6

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 |  | 5 |
| 7 | 8 | 6 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = 2+0=2 f(n) = 2+2=4 f(n) = 2+2=4**

**(Goal State) (repeated state)**

**Using Misplaced Tiles**

**Sample 8-Puzzle solved using A\* search Misplaced Tiles**

**Initial State Goal State**

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |

**Level: 0**

1

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = g(n)+h(n) = 0+2=2**

**Level: 1**

2 3

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 |  |
| 7 | 8 | 6 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 |  | 2 |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = 1+1=2 f(n) = 1+3=4**

**Level: 2**

4 5 6

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 |  | 5 |
| 7 | 8 | 6 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 | 2 |  |
| 4 | 5 | 3 |
| 7 | 8 | 6 |
|  |  |  |

**f(n) = 2+0=2 f(n) = 2+2=4 f(n) = 2+2=4**

**(Goal State) (repeated state)**

**PROGRAM DESIGN AND EXPLANATION:**

**Global variables**

ArrayList<String> *visited : Arraylist to keep track of visited nodes*

PriorityQueue<State> *statesToBeExpanded* : Priority queue to keep track of nodes that can be expanded.

**int** *nodeGenerated : counter to keep track of generated nodes.*

**Classes used**

**State:**

Class used to store the related information of each state. It contains constructor, getter and setter functions for the member variables. It implements the Comparable<> Interface so that the states added to the priority queue will be added according to ascending order of the f(n) value.

Contains all the methods used for creating the node with the default values, calculating cost, heuristic cost, total cost, retrieving the parent node, updating the current node with its parent node, updating the latest heuristic cost and updating the latest cost value.

**Main:**

Contains methods used to solve the 8-puzzle problem, to calculate the heuristics, to print the solution path, to find the possible next states i.e. children of provided node.

**Functions Used:**

**static void solve8PuzzleProblem(int[][], int[][], int) :** acts as a driver function to solve the 8-puzzle problem.

**static void printPath(ArrayList<State>) :** prints the final solution path.

**static int findHeuristicValue(int[][], int[][], int** **) :** acts as wrapper function to calculate heuristic.

**static ArrayList<int[][]> findNextState(int[][]) :** finds next possible nodes/children.

**static int manhattanDistance(int[][], int[][]) :** calculates heuristic Manhattan distance.

**static int misplacedTiles(int[][], int[][]):** calculates heuristic Misplaced Tiles.

**main function**

· Taking the Initial state and Final state input from user

· Checks for the valid input

· Initializing the Priority queue

**Queue:**

*statesToBeExpanded*: contains nodes that are yet to be expanded.

*Visited* : contains nodes that are already visited.

**Source Code:**

**Main.java:**

This file consists of the logic to accept the initial and goal states from user, logic to calculate heuristic, finding child nodes, printing the solution path.

**package** com.company;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.ArrayList;

**import** java.util.PriorityQueue;

**import** java.util.Scanner;

**public class** Main {

**static** ArrayList<String> *visited* = **new** ArrayList<>();

**static** PriorityQueue<State> *statesToBeExpanded* = **new** PriorityQueue<>();

**static boolean** *debug* = **false**;

**static int** *nodeGenerated* = 0;

**public static void** main(String[] args) **throws** IOException {

**int** choice = -1;

*//getting input from user*

**while** (choice != 0) {

*visited*.clear();

*statesToBeExpanded*.clear();

*nodeGenerated* = 0;

System.***out***.println(**"8-puzzle problem"**);

System.***out***.println(**"Choose heuristic function to solve the puzzle"**);

System.***out***.println(**"1.Manhattan Distance"**);

System.***out***.println(**"2.Misplaced Tiles"**);

System.***out***.println(**"0.Exit"**);

Scanner scan = **new** Scanner(System.***in***);

choice = scan.nextInt();

**if** (choice == 0)

System.*exit*(0);

**if** (choice != 1 && choice != 2 && choice != 0) {

System.***out***.println(**"Invalid choice of heuristic function. Please select correct heuristic function and try again!"**);

System.*exit*(0);

}

**int**[][] input\_state = **null**, goal\_state = **null**;

**try** {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));

System.***out***.println(**"Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8"**);

String inputString = br.readLine();

input\_state = Util.*convertStringTo2DArray*(inputString.trim());

System.***out***.println(**"Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8"**);

String goalString = br.readLine();

goal\_state = Util.*convertStringTo2DArray*(goalString.trim());

} **catch** (Exception ex) {

System.***out***.println(**"Error reading user input, provide input again"**);

System.*exit*(0);

}

*//printing input and user state which are provided by user.*

System.***out***.println(**"Input state is: "**);

Util.*printMat*(input\_state);

System.***out***.println(**"Goal state is: "**);

Util.*printMat*(goal\_state);

*solve8PuzzleProblem*(input\_state, goal\_state, choice);

}

}

*//acts as a driver function to solve 8puzzle problem using two heuristics*

**private static void** solve8PuzzleProblem(**int**[][] input\_state, **int**[][] goal\_state, **int** heuristic) {

**int** g = 0;

**int** f = 0;

**int** h = 0;

h = *findHeuristicValue*(input\_state, goal\_state, heuristic);

State inputState = **new** State(Util.*copyMat*(input\_state), g, h, (g + h));

*statesToBeExpanded*.add(inputState);

**boolean** goalReached = **false**;

ArrayList<String> path = **new** ArrayList<>();

ArrayList<State> statePath = **new** ArrayList<>();

**while** (!*statesToBeExpanded*.isEmpty()) {

State polledState = *statesToBeExpanded*.poll();

**if** (*debug*) {

System.***out***.println(**"Polled state is : "**);

System.***out***.println(polledState.toString());

}

**if** (polledState.getH() == 0) {

*//we have found the goal state, so break the loop*

statePath.addAll(polledState.getStatePath());

statePath.add(polledState);

goalReached = **true**;

**break**;

} **else** {

*visited*.add(Util.*matToString*(polledState.getState()));

ArrayList<**int**[][]> nextPossibleStateList = *findNextState*(polledState.getState());

**if** (*debug*) {

System.***out***.println(**"There are "** + nextPossibleStateList.size() + **" possible next states for this state"**);

}

**for** (**int**[][] nextState : nextPossibleStateList) {

**if** (*debug*) {

Util.*printMat*(nextState);

}

h = *findHeuristicValue*(nextState, goal\_state, heuristic);

g = polledState.getG() + 1;

f = h + g;

**if** (*debug*) {

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

System.***out***.println();

}

State s = **new** State(nextState, g, h, f);

s.getStatePath().addAll(polledState.getStatePath());

s.getStatePath().add(polledState);

*statesToBeExpanded*.add(s);

*nodeGenerated*++;

}

}

}

**if** (goalReached) {

System.***out***.println(**"goal state found"**);

System.***out***.println(**"Nodes Generated : "** + *nodeGenerated*);

System.***out***.println(**"Nodes expanded : "** + *visited*.size());

System.***out***.println(**"Path cost : "** + (statePath.size()-1));

System.***out***.println(**"Path :"**);

*printPath*(statePath);

} **else** {

System.***out***.println(**"goal state not found"**);

}

}

**private static void** printPath(ArrayList<State> pathList) {

**for**(State state: pathList){

*//Util.printMat(Util.stringToMatrix(state));*

state.toString();

System.***out***.println(**"------------------------------------------------------"**);

}

}

*//wrapper function used to call respective heuristic function*

**private static int** findHeuristicValue(**int**[][] input\_state, **int**[][] goal\_state, **int** heuristic) {

**int** dist = -1;

**if** (heuristic == 1)

dist = *manhattanDistance*(input\_state, goal\_state);

**else if** (heuristic == 2)

dist = *misplacedTiles*(input\_state, goal\_state);

**return** dist;

}

*//function to find out children of provided state and returns ArrayList containing children states.*

**private static** ArrayList<**int**[][]> findNextState(**int**[][] input\_state) {

ArrayList<**int**[][]> list = **new** ArrayList<**int**[][]>();

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

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

**if** (input\_state[i][j] == 0) {

**if** (i + 1 < 3) {

**int**[][] nextstate = Util.*copyMat*(input\_state);

**int** temp = nextstate[i + 1][j];

nextstate[i + 1][j] = nextstate[i][j];

nextstate[i][j] = temp;

**if** (!*visited*.contains(Util.*matToString*(nextstate)))

list.add(nextstate);

}

**if** (i - 1 >= 0) {

**int**[][] nextstate = Util.*copyMat*(input\_state);

**int** temp = nextstate[i - 1][j];

nextstate[i - 1][j] = nextstate[i][j];

nextstate[i][j] = temp;

**if** (!*visited*.contains(Util.*matToString*(nextstate)))

list.add(nextstate);

}

**if** (j + 1 < 3) {

**int**[][] nextstate = Util.*copyMat*(input\_state);

**int** temp = nextstate[i][j + 1];

nextstate[i][j + 1] = nextstate[i][j];

nextstate[i][j] = temp;

**if** (!*visited*.contains(Util.*matToString*(nextstate)))

list.add(nextstate);

}

**if** (j - 1 >= 0) {

**int**[][] nextstate = Util.*copyMat*(input\_state);

**int** temp = nextstate[i][j - 1];

nextstate[i][j - 1] = nextstate[i][j];

nextstate[i][j] = temp;

**if** (!*visited*.contains(Util.*matToString*(nextstate)))

list.add(nextstate);

}

**return** list;

}

}

}

**return null**;

}

*// function to calculate Manhatten distance*

**private static int** manhattanDistance(**int**[][] input\_state, **int**[][] goal\_state) {

**int** dist = 0;

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

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

**if** (input\_state[i][j] != goal\_state[i][j]) {

dist += *findInversion*(input\_state[i][j], i, j, goal\_state);

}

}

}

**return** dist;

}

*//function to find out the no of steps needed by a tile to acquire its correct position in the goal state*

**private static int** findInversion(**int** num, **int** indexi, **int** indexj, **int**[][] goal\_state) {

**int** inv = 0;

**if** (num == 0) **return** 0;

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

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

**if** (num == goal\_state[i][j]) {

inv = Math.*abs*(indexi - i) + Math.*abs*(indexj - j);

**return** inv;

}

}

}

**return** 0;

}

*//function to calculate misplaced tiles heuristic*

**private static int** misplacedTiles(**int**[][] input\_state, **int**[][] goal\_state) {

**int** misplacedTiles = 0;

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

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

**if** (input\_state[i][j] != goal\_state[i][j]) {

misplacedTiles += 1;

}

}

}

**return** misplacedTiles;

}

}

**State.java:**

This file contains class used to store the information about current state and its heuristic values and information about its parent.

**package** com.company;

**import** java.util.ArrayList;

**public class** State **implements** Comparable<State> {

**private int**[][] **state**;

**private int h**;

**private int g**;

**private int f**;

**private** ArrayList<State> **statePath**;

**public** State(**int**[][] state, **int** g, **int** h, **int** f) {

**this**.**state** = state;

**this**.**h** = h;

**this**.**g** = g;

**this**.**f** = f;

**this**.**statePath** = **new** ArrayList<>();

}

@Override

**public int** compareTo(State o) {

**if**(**this**.**f**>o.**f**) **return** 1;

**else if**(**this**.**f**<o.**f**) **return** -1;

**else return** 0;

}

@Override

**public** String toString() {

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

System.***out***.print(**"\t"**);

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

System.***out***.print(**state**[i][j] + **" "**);

}

System.***out***.println();

}

System.***out***.println(**"g:"** + **g** + **"\th:"** + **h** +**"\tf:"** + **f**);

**return ""**;

}

**public int**[][] getState() {

**return state**;

}

**public void** setState(**int**[][] state) {

**this**.**state** = state;

}

**public int** getH() {

**return h**;

}

**public void** setH(**int** h) {

**this**.**h** = h;

}

**public int** getG() {

**return g**;

}

**public void** setG(**int** g) {

**this**.**g** = g;

}

**public int** getF() {

**return f**;

}

**public void** setF(**int** f) {

**this**.**f** = f;

}

**public** ArrayList<State> getStatePath() {

**return statePath**;

}

**public void** setStatePath(ArrayList<State> statePath) {

**this**.**statePath** = statePath;

}

}

**Utility.java**

This file consists logic of some Utility functions such as parsing user input and creating 2D array, copy 2D array,

Print 2D array etc.

**package** com.company;

**public class** Util {

**public static int**[][] convertStringTo2DArray(String inputString) **throws** Exception{

**int**[][] arr = **new int**[3][3];

inputString=inputString.replace(**"{"**,**""**);

inputString=inputString.substring(0,inputString.length()-2);

*//System.out.println(inputString);*

String rows[] = inputString.split(**"},"**);

**for**(**int** i=0;i<rows.**length**;i++) {

*//System.out.println(rows[i]);*

String cols[] = rows[i].trim().split(**","**);

**for**(**int** j=0;j<cols.**length**;j++) {

arr[i][j] = Integer.*parseInt*(cols[j].trim());

}

}

**return** arr;

}

**public static boolean** isMatched(**int**[][] state, **int**[][] goal\_state) {

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

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

**if**(state[i][j]!=goal\_state[i][j])

**return false**;

}

}

**return true**;

}

**public static int**[][] copyMat(**int**[][] input\_state) {

**int**[][] nextstate = **new int**[3][3];

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

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

nextstate[i][j] = input\_state[i][j];

}

}

**return** nextstate;

}

**public static void** printMat(**int**[][] nextstate) {

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

System.***out***.print(**"\t"**);

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

System.***out***.print(nextstate[i][j] + **" "**);

}

System.***out***.println();

}

}

**public static** String matToString(**int**[][] nextstate) {

StringBuilder sb=**new** StringBuilder();

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

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

sb.append(nextstate[i][j]);

}

}

**return** sb.toString();

}

**public static int**[][] stringToMatrix(String s){

**int**[][] mat = **new int**[3][3];

**int** index =0;

**int** i=0;

**while**(index<s.length()){

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

mat[i][j]=Integer.*parseInt*(s.charAt(index)+**""**);

index++;

}

i++;

}

**return** mat;

}

}

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE: 1**

**System Generated Output for Misplaced Tiles**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

1

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1, 2, 3}, {7, 4,5}, {6,8, 0}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1,2,3},{8,6,4},{7,5,0}}

Input state is:

1 2 3

7 4 5

6 8 0

Goal state is:

1 2 3

8 6 4

7 5 0

goal state found

Nodes Generated : 18

Nodes expanded : 9

Path cost : 8

Path :

1 2 3

7 4 5

6 8 0

g:0 h:8 f:8

------------------------------------------------------

1 2 3

7 4 0

6 8 5

g:1 h:7 f:8

------------------------------------------------------

1 2 3

7 0 4

6 8 5

g:2 h:6 f:8

------------------------------------------------------

1 2 3

7 8 4

6 0 5

g:3 h:5 f:8

------------------------------------------------------

1 2 3

7 8 4

0 6 5

g:4 h:4 f:8

------------------------------------------------------

1 2 3

0 8 4

7 6 5

g:5 h:3 f:8

------------------------------------------------------

1 2 3

8 0 4

7 6 5

g:6 h:2 f:8

------------------------------------------------------

1 2 3

8 6 4

7 0 5

g:7 h:1 f:8

------------------------------------------------------

1 2 3

8 6 4

7 5 0

g:8 h:0 f:8

------------------------------------------------------

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE: 1**

**System Generated Output for Manhattan Distance**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

2

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1, 2, 3}, {7, 4,5}, {6,8, 0}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1,2,3},{8,6,4},{7,5,0}}

Input state is:

1 2 3

7 4 5

6 8 0

Goal state is:

1 2 3

8 6 4

7 5 0

goal state found

Nodes Generated : 42

Nodes expanded : 22

Path cost : 8

Path :

1 2 3

7 4 5

6 8 0

g:0 h:5 f:5

------------------------------------------------------

1 2 3

7 4 0

6 8 5

g:1 h:6 f:7

------------------------------------------------------

1 2 3

7 0 4

6 8 5

g:2 h:5 f:7

------------------------------------------------------

1 2 3

7 8 4

6 0 5

g:3 h:5 f:8

------------------------------------------------------

1 2 3

7 8 4

0 6 5

g:4 h:5 f:9

------------------------------------------------------

1 2 3

0 8 4

7 6 5

g:5 h:4 f:9

------------------------------------------------------

1 2 3

8 0 4

7 6 5

g:6 h:3 f:9

------------------------------------------------------

1 2 3

8 6 4

7 0 5

g:7 h:2 f:9

------------------------------------------------------

1 2 3

8 6 4

7 5 0

g:8 h:0 f:8

------------------------------------------------------

**SAMPLE: 2**

**System Generated Output for Manhattan Distance**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

1

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{2, 8, 1}, {3, 4,6}, {7,5,0}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{3,2,1},{8,0,4},{7,5,6}}

Input state is:

2 8 1

3 4 6

7 5 0

Goal state is:

3 2 1

8 0 4

7 5 6

goal state found

Nodes Generated : 12

Nodes expanded : 6

Path cost : 6

Path :

2 8 1

3 4 6

7 5 0

g:0 h:6 f:6

------------------------------------------------------

2 8 1

3 4 0

7 5 6

g:1 h:5 f:6

------------------------------------------------------

2 8 1

3 0 4

7 5 6

g:2 h:4 f:6

------------------------------------------------------

2 0 1

3 8 4

7 5 6

g:3 h:3 f:6

------------------------------------------------------

0 2 1

3 8 4

7 5 6

g:4 h:2 f:6

------------------------------------------------------

3 2 1

0 8 4

7 5 6

g:5 h:1 f:6

------------------------------------------------------

3 2 1

8 0 4

7 5 6

g:6 h:0 f:6

------------------------------------------------------

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE: 2**

**System Generated Output for Manhattan Distance**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

2

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{2, 8, 1}, {3, 4,6}, {7,5,0}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{3,2,1},{8,0,4},{7,5,6}}

Input state is:

2 8 1

3 4 6

7 5 0

Goal state is:

3 2 1

8 0 4

7 5 6

goal state found

Nodes Generated : 14

Nodes expanded : 7

Path cost : 6

Path :

2 8 1

3 4 6

7 5 0

g:0 h:6 f:6

------------------------------------------------------

2 8 1

3 4 0

7 5 6

g:1 h:5 f:6

------------------------------------------------------

2 8 1

3 0 4

7 5 6

g:2 h:3 f:5

------------------------------------------------------

2 0 1

3 8 4

7 5 6

g:3 h:4 f:7

------------------------------------------------------

0 2 1

3 8 4

7 5 6

g:4 h:3 f:7

------------------------------------------------------

3 2 1

0 8 4

7 5 6

g:5 h:2 f:7

------------------------------------------------------

3 2 1

8 0 4

7 5 6

g:6 h:0 f:6

------------------------------------------------------

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE: 3**

**System Generated Output for Misplaced Tiles**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

1

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{7, 2, 4}, {5, 0,6}, {8,3,1}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1, 2, 3}, {4, 5,6}, {7, 8,0}}

Input state is:

7 2 4

5 0 6

8 3 1

Goal state is:

1 2 3

4 5 6

7 8 0

goal state found

Nodes Generated : 413

Nodes expanded : 251

Path cost : 20

Path :

7 2 4

5 0 6

8 3 1

g:0 h:14 f:14

------------------------------------------------------

7 2 4

5 3 6

8 0 1

g:1 h:13 f:14

------------------------------------------------------

7 2 4

5 3 6

8 1 0

g:2 h:12 f:14

------------------------------------------------------

7 2 4

5 3 0

8 1 6

g:3 h:13 f:16

------------------------------------------------------

7 2 4

5 0 3

8 1 6

g:4 h:12 f:16

------------------------------------------------------

7 2 4

0 5 3

8 1 6

g:5 h:11 f:16

------------------------------------------------------

0 2 4

7 5 3

8 1 6

g:6 h:10 f:16

------------------------------------------------------

2 0 4

7 5 3

8 1 6

g:7 h:11 f:18

------------------------------------------------------

2 4 0

7 5 3

8 1 6

g:8 h:10 f:18

------------------------------------------------------

2 4 3

7 5 0

8 1 6

g:9 h:9 f:18

------------------------------------------------------

2 4 3

7 0 5

8 1 6

g:10 h:10 f:20

------------------------------------------------------

2 4 3

7 1 5

8 0 6

g:11 h:9 f:20

------------------------------------------------------

2 4 3

7 1 5

0 8 6

g:12 h:8 f:20

------------------------------------------------------

2 4 3

0 1 5

7 8 6

g:13 h:7 f:20

------------------------------------------------------

2 4 3

1 0 5

7 8 6

g:14 h:6 f:20

------------------------------------------------------

2 0 3

1 4 5

7 8 6

g:15 h:5 f:20

------------------------------------------------------

0 2 3

1 4 5

7 8 6

g:16 h:4 f:20

------------------------------------------------------

1 2 3

0 4 5

7 8 6

g:17 h:3 f:20

------------------------------------------------------

1 2 3

4 0 5

7 8 6

g:18 h:2 f:20

------------------------------------------------------

1 2 3

4 5 0

7 8 6

g:19 h:1 f:20

------------------------------------------------------

1 2 3

4 5 6

7 8 0

g:20 h:0 f:20

------------------------------------------------------

**SAMPLE INPUT/OUTPUT CASES**

**SAMPLE: 3**

**System Generated Output for Manhattan Distance**

8-puzzle problem

Choose heuristic function to solve the puzzle

1.Manhattan Distance

2.Misplaced Tiles

0.Exit

2

Provide input state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{7, 2, 4}, {5, 0,6}, {8,3,1}}

Provide goal state in the form {{x,x,x},{x,x,x},{x,x,x}} using digits 0 to 8

{{1, 2, 3}, {4, 5,6}, {7, 8,0}}

Input state is:

7 2 4

5 0 6

8 3 1

Goal state is:

1 2 3

4 5 6

7 8 0

goal state found

Nodes Generated : 6270

Nodes expanded : 3974

Path cost : 20

Path :

7 2 4

5 0 6

8 3 1

g:0 h:7 f:7

------------------------------------------------------

7 2 4

5 3 6

8 0 1

g:1 h:7 f:8

------------------------------------------------------

7 2 4

5 3 6

8 1 0

g:2 h:6 f:8

------------------------------------------------------

7 2 4

5 3 0

8 1 6

g:3 h:8 f:11

------------------------------------------------------

7 2 4

5 0 3

8 1 6

g:4 h:8 f:12

------------------------------------------------------

7 2 4

0 5 3

8 1 6

g:5 h:7 f:12

------------------------------------------------------

0 2 4

7 5 3

8 1 6

g:6 h:7 f:13

------------------------------------------------------

2 0 4

7 5 3

8 1 6

g:7 h:8 f:15

------------------------------------------------------

2 4 0

7 5 3

8 1 6

g:8 h:8 f:16

------------------------------------------------------

2 4 3

7 5 0

8 1 6

g:9 h:7 f:16

------------------------------------------------------

2 4 3

7 0 5

8 1 6

g:10 h:8 f:18

------------------------------------------------------

2 4 3

7 1 5

8 0 6

g:11 h:8 f:19

------------------------------------------------------

2 4 3

7 1 5

0 8 6

g:12 h:7 f:19

------------------------------------------------------

2 4 3

0 1 5

7 8 6

g:13 h:6 f:19

------------------------------------------------------

2 4 3

1 0 5

7 8 6

g:14 h:6 f:20

------------------------------------------------------

2 0 3

1 4 5

7 8 6

g:15 h:6 f:21

------------------------------------------------------

0 2 3

1 4 5

7 8 6

g:16 h:5 f:21

------------------------------------------------------

1 2 3

0 4 5

7 8 6

g:17 h:4 f:21

------------------------------------------------------

1 2 3

4 0 5

7 8 6

g:18 h:3 f:21

------------------------------------------------------

1 2 3

4 5 0

7 8 6

g:19 h:2 f:21

------------------------------------------------------

1 2 3

4 5 6

7 8 0

g:20 h:0 f:20

-----------------------------------------------------

**Inferences:**

**While computing the algorithm, we are not generating the nodes which are already generated.**

**RESULTS**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial and Goal States** | | | | | | | | | **Manhattan Distance Heuristic** | **Misplaced Tiles Heuristic** |
| **Initial State** | | | | **Goal State** | | | |  | Number of Nodes Generated: **18**  Number of Nodes Expanded: **9**  Path cost: **8** | Number of Nodes Generated: **42**  Number of Nodes Expanded: **22**  Path cost: **8** |
|  | **1** | **2** | **3** |  | **1** | **2** | **3** |  |
|  | **7** | **4** | **5** | **8** | **6** | **4** |
|  | **6** | **8** | **0** | **7** | **5** | **0** |
| **Initial State** | | | | **Goal State** | | | |  | Number of Nodes Generated: **12**  Number of Nodes Expanded: **6**  Path cost: **6** | Number of Nodes Generated: **14**  Number of Nodes Expanded: **7**  Path cost: **6** |
|  | **2** | **8** | **1** |  | **3** | **2** | **1** |  |
|  | **3** | **4** | **6** | **8** | **0** | **4** |
|  | **7** | **5** | **0** | **7** | **5** | **6** |
| **Initial State** | | | | **Goal State** | | | |  | Number of Nodes Generated: **413**  Number of Nodes Expanded: **251**  Path cost: **20** | Number of Nodes Generated: **6270**  Number of Nodes Expanded: **3974**  Path cost: **20** |
|  | **7** | **2** | **4** |  | **1** | **2** | **3** |  |
|  | **5** | **0** | **6** | **4** | **5** | **6** |
|  | **8** | **3** | **1** | **7** | **8** | **0** |

**Observations:**

From the above experimental results, we see that the number of nodes generated when Misplaced Tile heuristic is used are more compared to Manhattan Distance heuristic