**N-Puzzle**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Team ID: **T061**
* Team Members:

|  |  |  |
| --- | --- | --- |
| Name | ID | Department |
| **علاء عادل درويش حسين** | **20191700388** | **CS** |
| **محمد حسن ابراهيم حسن** | **20191700522** | **CS** |
| **مروان صلاح محمد محمود** | **20191700618** | **CS** |

* **Time Execution for Each Test Case:**

1- Sample Test Case:

|  |  |
| --- | --- |
| File Name | Time (Second & Millisecond) |
| Solvable | |
| 8 Puzzle (1) | 2 milliseconds  0 seconds |
| 8 Puzzle (2) | 1 millisecond  0 second |
| 8 Puzzle (3) | 0 millisecond  0 second |
| 15 Puzzle - 1 | 0 millisecond  0 second |
| 24 Puzzle 1 | 0 millisecond  0 second |
| 24 Puzzle 2 | 0 millisecond  0 second |
| Unsolvable | |
| 8 Puzzle - Case 1 | 0 millisecond  0 second |
| 8 Puzzle(2) - Case 1 | 0 millisecond  0 second |
| 8 Puzzle(3) - Case 1 | 0 millisecond  0 second |
| 15 Puzzle - Case 2 | 0 millisecond  0 second |
| 15 Puzzle - Case 3 | 0 millisecond  0 second |

2- Complete Test Case with Manhattan Vs Hamming:

* Manhattan: 35 milliseconds.
* Hamming: 559 milliseconds.
* #Moves In Hamming Are Equal #Moves in Manhattan.

3- Complete Test Case Manhattan Only:

|  |  |
| --- | --- |
| File Name | Time (Second & Millisecond) |
| Solvable | |
| 15 Puzzle 1 | 1608 milliseconds  1 seconds |
| 15 Puzzle 3 | 1433 milliseconds  1 seconds |
| 15 Puzzle 4 | 487 milliseconds  0 second |
| 15 Puzzle 5 | 6254 milliseconds  6 seconds |

4- V. Large test case:

|  |  |
| --- | --- |
| File Name | Time (Second & Millisecond) |
| Solvable | |
| TEST | 0 min 28 seconds 28866 milliseconds |

* **Complexity Analysis:**

1- Analysis of solvePuzzle.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Puzzel

{

class solvePuzzel

{

private void swap2ndheursitc(State cur, ref State child, int newxZero, int newyZero)

{

child.matrix = cur.matrix.Clone() as int[,];

int num1 = cur.matrix[newxZero, newyZero];

child.matrix[cur.xZero, cur.yZero] = num1;

child.matrix[newxZero, newyZero] = 0;

child.xZero = newxZero; child.yZero = newyZero;

//if(State.size==3)

child.correctPath = cur.correctPath + num1.ToString() + '|';

child.depth = cur.depth + 1;

if (State.type == 'H')

Hamming(child);

else if (State.type == 'M')

Manhatendistance(child);

child.totalCost = child.depth + child.heursitcValue;

}

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

Complexity: θ(n^2).

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

public PriorityQueue<State, int> notVistedState;

public Queue<State> queu;

HashSet<int> visitedState;

public solvePuzzel()

{

visitedState = new HashSet<int>();

notVistedState = new PriorityQueue<State, int>();

queu = new Queue<State>();

}

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

Complexity: θ(1).

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

public State Bfs(State cur)

{

while (queu.Count > 0)

{

State temp = queu.Dequeue();

if (temp.heursitcValue == 0) return temp;

visitedState.Add(temp.hash);

Expand(temp);

}

return cur;

}

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

s = queue Size.

Complexity: O(s\*(n^2)).

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

public State Asatr(State cur)

{

while (notVistedState.Count > 0)

{

State temp = notVistedState.Dequeue();

if (temp.heursitcValue == 0) return temp;

visitedState.Add(temp.hash);

Expand(temp);

}

return cur;

}

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

s = queue Size.

Complexity: θ(s\*(n^2)).

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

void Expand(State cur)

{

if (cur.xZero + 1 < State.size)//Down

{

State child = new State();

swap2ndheursitc(cur, ref child, cur.xZero + 1, cur.yZero);

if (IsInClosed(child.hash) == false)

{

//child.CorrectPath.Add(cur.elements);//Take Copy from list correct path

if (State.typeofAlgorithmis == 'A')

notVistedState.Enqueue(child, child.totalCost);

else if (State.typeofAlgorithmis == 'B')

queu.Enqueue(child);

}

}

if (cur.xZero - 1 >= 0)//Up

{

State child = new State();

swap2ndheursitc(cur, ref child, cur.xZero - 1, cur.yZero);

if (IsInClosed(child.hash) == false)

{

// child.CorrectPath.Add(cur.elements);

if (State.typeofAlgorithmis == 'A')

notVistedState.Enqueue(child, child.totalCost);

else if (State.typeofAlgorithmis == 'B')

queu.Enqueue(child);

}

}

if (cur.yZero - 1 >= 0)

{

State child = new State();

swap2ndheursitc(cur, ref child, cur.xZero, cur.yZero - 1);

if (IsInClosed(child.hash) == false)

{

if (State.typeofAlgorithmis == 'A')

notVistedState.Enqueue(child, child.totalCost);

else if (State.typeofAlgorithmis == 'B')

queu.Enqueue(child);

}

}

if (cur.yZero + 1 < State.size)

{

State child = new State();

swap2ndheursitc(cur, ref child, cur.xZero, cur.yZero + 1);

if (IsInClosed(child.hash) == false)

{

if (State.typeofAlgorithmis == 'A')

notVistedState.Enqueue(child, child.totalCost);

else if (State.typeofAlgorithmis == 'B')

queu.Enqueue(child);

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Complexity: O(n^2).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bool IsInClosed(int hash)

{

if (visitedState.Contains(hash)) return true;

return false;

}

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

Complexity: θ(1).

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

public void Hamming(State state)

{

for (int row = 0; row < State.size; row++)

{

for (int colum = 0; colum < State.size; colum++)

{

state.hash = state.hash \* 31 + state.matrix[row, colum];

if (state.matrix[row, colum] == 0)

continue;

if (State.goolState[state.matrix[row, colum]].row != row || State.goolState[state.matrix[row, colum]].col != colum)

state.heursitcValue++;

}

}

}

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

Complexity: θ(n^2).

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

public void Manhatendistance(State state)

{

for (int i = 0; i < State.size; i++)

{

for (int x = 0; x < State.size; x++)

{

state.hash = state.hash \* 31 + state.matrix[i, x];

if (state.matrix[i, x] == 0) continue;

int row = State.goolState[state.matrix[i, x]].row;

int colum = State.goolState[state.matrix[i, x]].col;

state.heursitcValue += Math.Abs(row - i) + Math.Abs(colum - x);

}

}

}

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

Complexity: θ(n^2).

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

2- Analysis of State.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static Puzzel.Program;

namespace Puzzel

{

class State

{

public static char type;

public static char typeofAlgorithmis;

public static int size;

public int xZero, yZero;

public static Dictionary<int, Coordinate> goolState;

public int[,] matrix;

public int heursitcValue=0, totalCost, depth=0;

public string correctPath;

public int hash=1;

}

}

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

Complexity: O (1).

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

3- Analysis of Checksolvable.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace Solvable

{

class Checksolvable

{

public Boolean Check\_puzz(int size,List<int> number, int BlankPOstion)

{

int inversioCount = getInversion(number, size);

if ((size % 2 != 0 && inversioCount % 2 == 0)||((size % 2 == 0 && inversioCount % 2 == 0 && BlankPOstion % 2 == 0))||((size % 2 == 0 && inversioCount % 2 != 0 && BlankPOstion % 2 != 0)))

return true;

return false;

}

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

Complexity: θ(n^2).

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

private int getInversion(List<int> number, int size)

{

int inversionCount = 0;

for(int i=0;i<number.Count-1;i++)

{

for(int x=i;x<number.Count;x++)

{

if (number[x] == 0) continue;

if ( number[i] > number[x])

inversionCount++;

}

}

return inversionCount;

}

}

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

Complexity: θ(n^2).

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

}

3- Analysis of Form1.cs:

public int[,] getMatrix(string path)

{

List<string> file = File.ReadAllLines(path).ToList();

sizeOfMatrix = int.Parse(file[0]);

int[,] matrix = new int[sizeOfMatrix, sizeOfMatrix];

int row = 0, col;

for (int x = 1; x < file.Count; x++)

{

col = 0;

if (file[x] == "") continue;

foreach (string val in file[x].Split(" "))

{

if (val == "") continue;

matrix[row, col] = int.Parse(val);

if (int.Parse(val) == 0)

{

postionX = row;

postionY = col;

}

col++;

}

row++;

}

if(sizeOfMatrix==3)

print(matrix, sizeOfMatrix);

return matrix;

}

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

Complexity: θ(n^2)

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

Dictionary<int, Coordinate> getGoolState(int[,] matrix)

{

var goolState = new Dictionary<int, Coordinate>();

number = new List<int>();

int counter = 1;

// string elements = string.Empty;

for (int row = 0; row < sizeOfMatrix; row++)

{

for (int colum = 0; colum < sizeOfMatrix; colum++)

{

number.Add(matrix[row, colum]);

if (counter == sizeOfMatrix \* sizeOfMatrix)

counter = 0;

Coordinate coordinate = new Coordinate();

coordinate.row = row;

coordinate.col = colum;

goolState.Add(counter, coordinate);

counter++;

}

}

return goolState;

}

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

Complexity: θ(n^2)

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

void print(int[,] matrix,int size)

{

int row = 0, col = 0;

int[] y = { 14, 14, 14,123,123,123,215,215,215 };

int[] x = { 14, 116, 225, 14, 116, 225, 14, 116, 225 };

for(int i=0;i<x.Length;i++)

{

foreach (Button button in panel1.Controls)

{

button.Enabled = true;

if (button.Location.X == x[i] && button.Location.Y== y[i])

{

button.BackgroundImage = Image.FromFile($"{matrix[row, col]}.JPG");

col++;

if (col == size)

{

row++;

col = 0;

}

break;

}

}

}

}

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

Complexity: θ(n^2)

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

void printPath(int[,] matrix,ref State state,int num)

{

if (state.xZero - 1 >= 0 && matrix[state.xZero - 1, state.yZero] == num)

{

int num1 = matrix[state.xZero - 1, state.yZero];

matrix[state.xZero, state.yZero] = num1;

matrix[state.xZero - 1, state.yZero] = 0;

state.xZero = state.xZero - 1;

print(matrix,State.size);

}

else if (state.xZero + 1 < sizeOfMatrix && matrix[state.xZero + 1, state.yZero] == num)

{

int num1 = matrix[state.xZero + 1, state.yZero];

matrix[state.xZero, state.yZero] = num1;

matrix[state.xZero + 1, state.yZero] = 0;

state.xZero = state.xZero + 1;

print(matrix, State.size);

}

else if (state.yZero + 1 < sizeOfMatrix && matrix[state.xZero, state.yZero + 1] == num)

{

int num1 = matrix[state.xZero, state.yZero + 1];

matrix[state.xZero, state.yZero] = num1;

matrix[state.xZero, state.yZero + 1] = 0;

state.yZero = state.yZero + 1;

print(matrix, State.size);

}

else if (state.yZero - 1 >= 0 && matrix[state.xZero, state.yZero - 1] == num)

{

int num1 = matrix[state.xZero, state.yZero - 1];

matrix[state.xZero, state.yZero] = num1;

matrix[state.xZero, state.yZero - 1] = 0;

state.yZero = state.yZero - 1;

print(matrix, State.size);

}

}

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

Complexity: θ(n^2).

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

int counter = 0;

private void solve\_Click(object sender, EventArgs e)

{

if (State.size == 3)

{

string []num = state.correctPath.Split("|");

if (counter < num.Length && num[counter]!="")

printPath(copyMatrix, ref state,int.Parse(num[counter]));

counter++;

}

}

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

Complexity: θ(n^2).

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

private void button1\_Click(object sender, EventArgs e)

{

Stopwatch stopwatch = new Stopwatch();

state = new State();

try

{

state.matrix = getMatrix(file\_name.Text.ToString());

}

catch(Exception ex)

{

minute.Text = "";

sec.Text = "";

millisec.Text = "";

moves.Text = "";

path.Text = "";

solvable.Text = "";

MessageBox.Show("Invalid File Name!!");

return;

}

State.goolState = getGoolState(state.matrix);

Checksolvable checksolvable = new Checksolvable();

if (checksolvable.Check\_puzz(sizeOfMatrix, number, postionX + 1)&&(hamming.Checked||manhattan.Checked)&&(A\_star.Checked||bfs.Checked))

{

solvePuzzel solvePuzzel = new solvePuzzel();

copyMatrix= state.matrix;

State.size = sizeOfMatrix;

state.xZero = postionX;

state.yZero = postionY;

state.totalCost = state.heursitcValue;

if (hamming.Checked)

{

State.type = 'H';

solvePuzzel.Hamming(state);

}

else if (manhattan.Checked)

{

State.type = 'M';

solvePuzzel.Manhatendistance(state);

}

if (A\_star.Checked)

{

State.typeofAlgorithmis = 'A';

solvePuzzel.notVistedState.Enqueue(state, state.totalCost);

stopwatch.Start();

state = solvePuzzel.Asatr(state);

}

else if (bfs.Checked)

{

State.typeofAlgorithmis = 'B';

solvePuzzel.queu.Enqueue(state);

stopwatch.Start();

state = solvePuzzel.Bfs(state);

}

stopwatch.Stop();

solvable.Text = "Puzzle Can be Solvable";

minute.Text = stopwatch.Elapsed.Minutes.ToString();

sec.Text = stopwatch.Elapsed.Seconds.ToString();

millisec.Text = stopwatch.Elapsed.Milliseconds.ToString();

moves.Text = state.depth.ToString();

path.Text = state.correctPath;

state.xZero = postionX;

state.yZero = postionY;

}

else

{

minute.Text = "";

sec.Text = "";

millisec.Text = "";

moves.Text = "";

path.Text = "";

solvable.Text = "Puzzle Can't be Solvable";

}

}

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

s = queue Size.

Complexity: θ (s \*n^2 ).

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Bonus Tasks:

1. Solve Puzzle By BFS.
2. GUI.