**Algorithms Analysis & Design project documentation**

**Project N puzzle**

Team : T121

Team Members

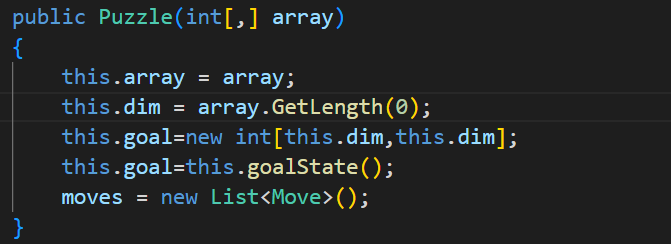
|  |  |  |  |
| --- | --- | --- | --- |
| Name | ID | Section | Department |
| Mahmoud Salama Mohamed AbdelRahman | 20191700601 | 4 | CS |
| Mahmoud Saeed Sayed Riad | 20191700599 | 4 | CS |
| Eman Ibrahim Yusuf Sam | 20191700150 | 1 | CS |

Note : suppose that dimension of puzzle = N

Where N^2 = S

S = (puzzle Size)

Puzzle class :



1. Total the complexity of constructor =: O(S)

Explain : line 2 complexity (GetLengh(0)) =: O(1)

: line 4 complexity (goalState()) =: O(S)

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of goalState function =: O(S)

Explain : line 2 to line 5 complexity (first loop ) =: O(S)

: line 8 to line 14 complexity (2 nested loop) =: O(S

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of manhattanDistance function =: O(S)

Explain : line 2 to line 12 complexity (2 nested loop) =: O(S)

A screenshot of a computer

Description automatically generated with medium confidence

4) Total the complexity of Hamming Distance function =: O(S)

Explain: line 2 to line 10 complexity (2 nested loop) =: O(S)

Text

Description automatically generated

5) Total the complexity of getZeroindex function =: O(S)

Explain : line 2 to line 13 complexity (2 nested loop) =: O(S) where Add() complexity =: O(1)

Text

Description automatically generated

1. Total the complexity of Actions function =: O(S)

Explain : line 1 complexity ( calling getZeroindex function ) =: O(S)

: line 2 complexity ( calling moveRight ) =: O(S)

: line 3 complexity ( calling moveLeft function ) =: O(S)

: line 4 complexity ( calling moveUp function ) =: O(S)

: line 5 complexity ( calling moveDown function ) =: O(S)

Text

Description automatically generated

1. Total the complexity of moveRight function =: O(S)

Explain : line 5 using Clone function =: O(S)

A screenshot of a computer

Description automatically generated

1. Total the complexity of moveLeft function =: O(S)

Explain : line 5 using Clone function =: O(S)

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of moveUp function =: O(S)

Explain : line 5 using Clone function =: O(S)

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of moveDown function =: O(S)

Explain : line 5 using Clone function =: O(S)

Text

Description automatically generated

1. Total the complexity of indexMove function =: O(S)

Explain : line 1 using Clone function =: O(S)

Text

Description automatically generated

1. Total the complexity of PrintPuzzel function =: O(S)

Explain : line 2 to line 9 complexity (2 nested loop) =: O(S)

Graphical user interface, text

Description automatically generated

1. Total the complexity of Solved function =: O(S)

Explain : complexity (calling manhattanDistance function ) =: O(S)

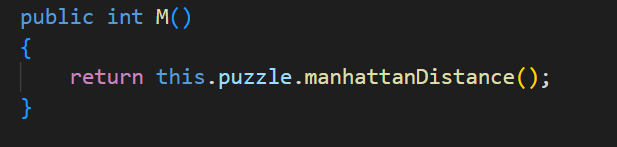
* Node class

Text

Description automatically generated with medium confidence

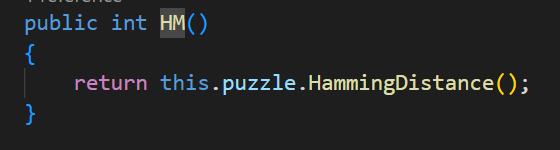
1. Total the complexity of AstareScore function =: O(S)

Explain : complexity (calling HM function or M function ) =: O(S)



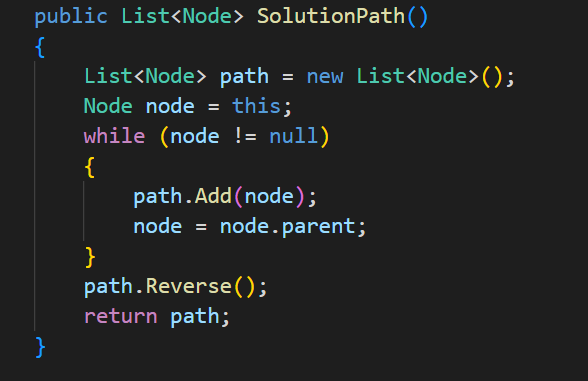
1. Total the complexity of M function =: O(N^2)

Explain : complexity (calling manhattanDistance function ) =: O(S)



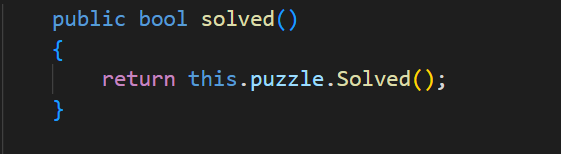
1. Total the complexity of HM function =: O(S)

Explain : complexity (calling HamingDistance function ) =: O(S)



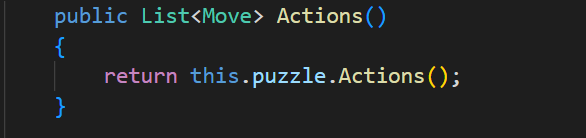
1. Total the complexity of SolutionPath function =: O(S)

Explain : complexity (Reverse function ) =: O(L) where L is number of element will be return



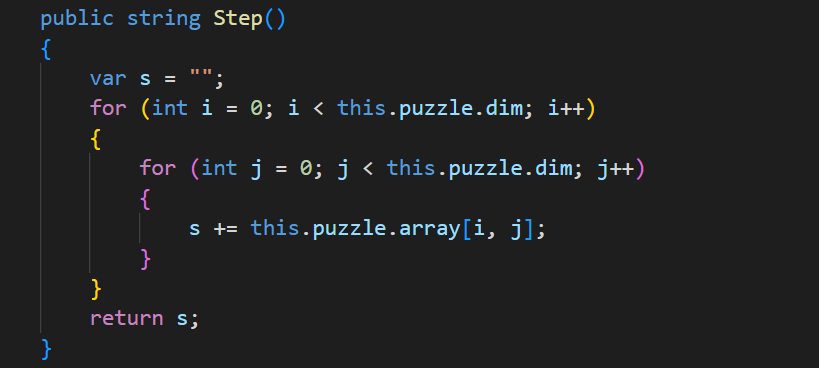
1. Total the complexity of solved function =: O(S)

Explain : complexity (calling Solved function ) =: O(S)



1. Total the complexity of Actions function =: O(S)

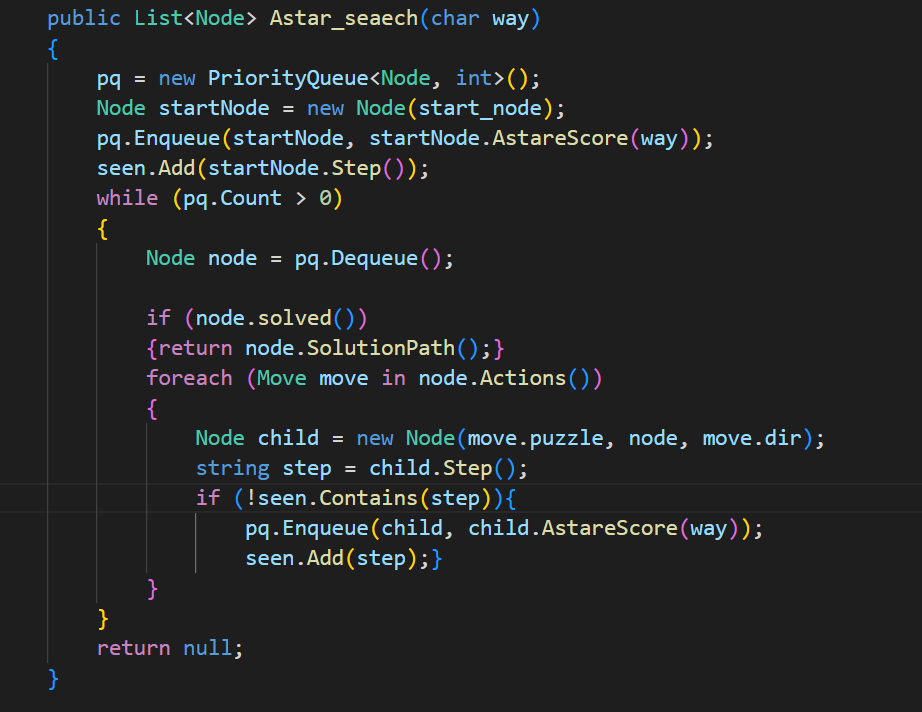
Explain : complexity (calling Actions function ) =: O(S)



1. Total the complexity of Step function =: O(S)

Explain : line 2 to line 8 complexity (2 nested loop) =: O(S)

* Astar class

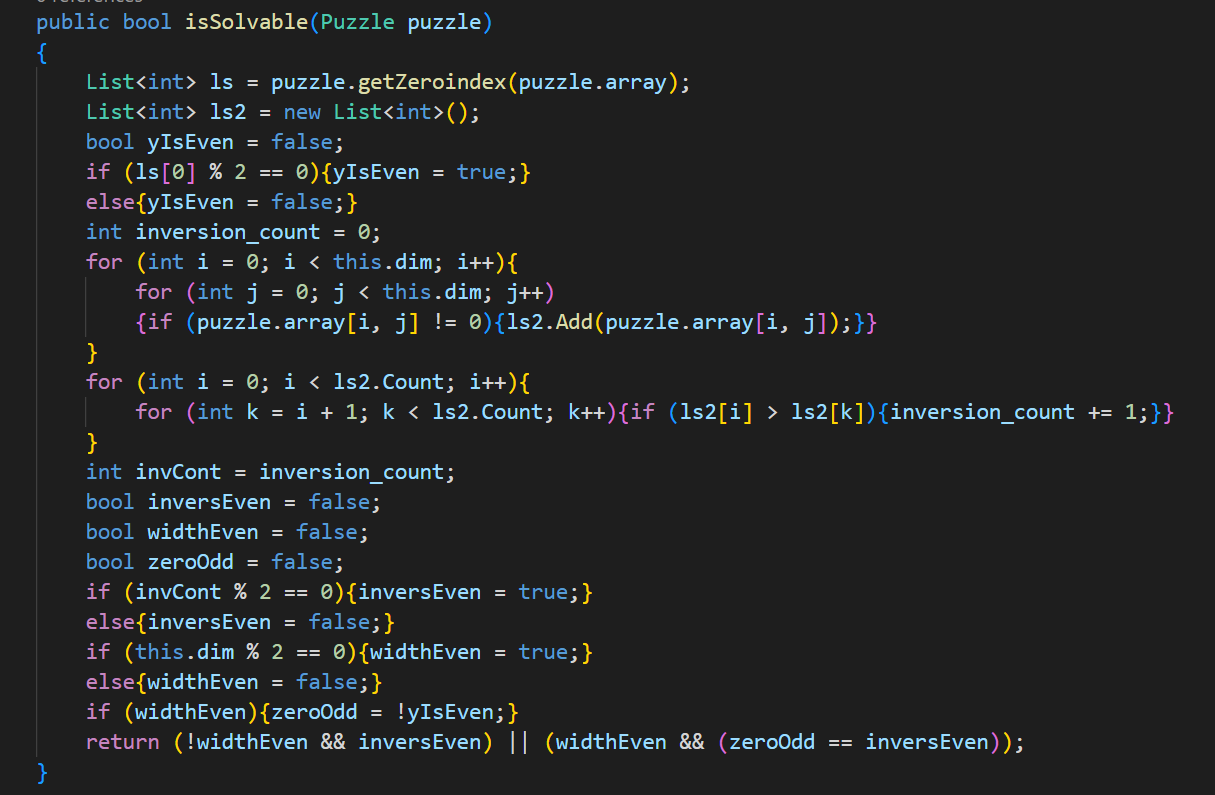


1. Total the complexity of Astar\_search function =: O(Log(V)\*S)

Explain : line 2 to line 5 complexity (first loop ) =: O(S)

: line 8 to line 14 complexity (2 nested loop) =: O(S)

* Solvable class



1. Total the complexity of isSolveable function =: O(S)

Explain : complexity (calling getZeroindex function ) =: O(S)

: line 7 to line 10 complexity (first nested loop ) =: O(S)

: line 11 to line 13 complexity (2 nested loop) =: O(S)

* BFS class

Text

Description automatically generated

1. Total the complexity of BFSSolve function =: O(V \* L)

V: num of nodes in graph

Explain : complexity (calling step function ) =: O(S)

: complexity (getAdj ) =: O(S)

: complexity (Solved ) =: O(S)

: complexity (SlutionPath ) =: O( L )

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of getAdj function =: O(S)

Explain : complexity (calling Actions function ) =: O(S)

: complexity (calling Step function ) =: O(S)

* DFS class

Text

Description automatically generated

1. Total the complexity of DFSSolve function =: O(V \* L)

V: num of nodes in graph

Explain : complexity (calling step function ) =: O(S)

: complexity (getAdj ) =: O(S)

: complexity (Solved ) =: O(S)

: complexity (SlutionPath ) =: O( L )

A screenshot of a computer

Description automatically generated with medium confidence

1. Total the complexity of getAdj function =: O(S)

Explain : complexity (calling Actions function ) =: O(S)

: complexity (calling Step function ) =: O(S)

Comparison Manhattan vs Hamming

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Name** | **Manhattan** | | **Hamming** | |
|  | Steps | time | Steps | Time |
| 50 puzzle | 18 | 1:177ms | 18 | 12:53ms |
| 99 puzzle 1 | 18 | 1:177ms | 18 | 12:45ms |
| 99 puzzle 2 | 38 | 1:179ms | 38 | 12:72ms |
| 9999 puzzle | 4 | 1:175ms | 4 | 12:52ms |
| Summary | Same number of Steps but run time of Manhattan is lower than hamming | | | |

Code

Class puzzle

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NPuzzle

{

    class Move

    {

        public Puzzle puzzle;

        public string dir;

    }

    internal class Puzzle

    {

        public int[,] array;

        public int dim;

        public List<Move> moves;

        int [,] goal;

        //List<int> zeroIndex = new List<int> ();

        // O(N^2)

        public Puzzle(int[,] array)

        {

            this.array = array;

            this.dim = array.GetLength(0);  // O(1)

            this.goal=new int[this.dim,this.dim];

            this.goal=this.goalState();  // O(N^2)

            moves = new List<Move>();

        }

        // O(N^2)

        public int[,] goalState()

        {

            int[] arr = new int[this.dim \* this.dim];

            for (int i = 0 ,v=1; i < this.dim \* this.dim; i++,v++)

            {

                arr[i] = v;

            }

            arr[(this.dim \* this.dim) - 1] = 0;

            int[,] output = new int[this.dim, this.dim];

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j <this.dim; j++)

                {

                    output[i, j] = arr[i \* this.dim + j];

                }

            }

            return output;

        }

       // O(N^2)

        public int manhattanDistance()

        {

            int distance = 0;

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j < this.dim; j++)

                {

                    if (this.array[i, j] != 0)

                    {

                        int x = (this.array[i, j] - 1) / this.dim;

                        int y = (this.array[i, j] - 1) % this.dim;

                        distance += Math.Abs(x - i) + Math.Abs(y - j);

                    }

                }

            }

            return distance;

        }

        // O(N^2)

        public int HammingDistance()

        {

            int count=0;

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j < this.dim; j++)

                {

                    if(this.array[i,j]!=this.goal[i,j]){

                        count++;

                    }

                }

            }

            return count;

        }

        // O(N^2)

        public List<int> getZeroindex(int[,] arr)

        {

            List<int> ls = new List<int>();

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j < this.dim; j++)

                {

                    if (arr[i, j] == 0)

                    {

                        ls.Add(i);  // O(1)

                        ls.Add(j);  // O(1)

                        return ls;

                    }

                }

            }

            return null;

        }

        // O(N^2)

        public List<Move> Actions()

        {

            List<int> ls = getZeroindex(array); // O(N^2)

            moveRight(array, ls[0], ls[1]);     // O(N^2)

            moveLeft(array, ls[0], ls[1]);      // O(N^2)

            moveUp(array, ls[0], ls[1]);        // O(N^2)

            moveDown(array, ls[0], ls[1]);      // O(N^2)

            return this.moves;

        }

        // O(N^2)

        public void moveLeft(int[,] p, int idx\_row, int idx\_col)

        {

            Move move = new Move();

            if (idx\_row >= 0 && idx\_col - 1 >= 0 &&

                idx\_row < this.dim && idx\_col - 1 < this.dim)

            {

                int[,] pc = (int[,])p.Clone();  // O(N^2)

                int temp = pc[idx\_row, idx\_col - 1];

                pc[idx\_row, idx\_col - 1] = pc[idx\_row, idx\_col];

                pc[idx\_row, idx\_col] = temp;

                Puzzle puzzle = new Puzzle(pc);

                move.puzzle = puzzle;

                move.dir = "Left";

                this.moves.Add(move);  // O(1)

            }

        }

        // O(N^2)

        public void moveRight(int[,] p, int idx\_row, int idx\_col)

        {

            Move move = new Move();

            if (idx\_row >= 0 && idx\_col + 1 >= 0 &&

                idx\_row < this.dim && idx\_col + 1 < this.dim)

            {

                int[,] pc = (int[,])p.Clone();  // O(N^2)

                int temp = pc[idx\_row, idx\_col + 1];

                pc[idx\_row, idx\_col + 1] = pc[idx\_row, idx\_col];

                pc[idx\_row, idx\_col] = temp;

                Puzzle puzzle = new Puzzle(pc);

                move.puzzle = puzzle;

                move.dir = "Right";

                this.moves.Add(move);  // O(1)

            }

        }

        // O(N^2)

        public void moveDown(int[,] p, int idx\_row, int idx\_col)

        {

            Move move = new Move();

            if (idx\_row + 1 >= 0 && idx\_col >= 0 &&

                idx\_row + 1 < this.dim && idx\_col < this.dim)

            {

                // make a copy

                int[,] pc = (int[,])p.Clone();  // O(N^2)

                int temp = pc[idx\_row + 1, idx\_col];

                pc[idx\_row + 1, idx\_col] = pc[idx\_row, idx\_col];

                pc[idx\_row, idx\_col] = temp;

                Puzzle puzzle = new Puzzle(pc);

                move.puzzle = puzzle;

                move.dir = "Down";

                this.moves.Add(move);  // O(1)

            }

        }

        // O(N^2)

        public void moveUp(int[,] p, int idx\_row, int idx\_col)

        {

            Move move = new Move();

            if (idx\_row - 1 >= 0 && idx\_col >= 0 &&

                idx\_row - 1 < this.dim && idx\_col < this.dim)

            {

                // make a copy

                int[,] pc = (int[,])p.Clone();  // O(N^2)

                int temp = pc[idx\_row - 1, idx\_col];

                pc[idx\_row - 1, idx\_col] = pc[idx\_row, idx\_col];

                pc[idx\_row, idx\_col] = temp;

                Puzzle puzzle = new Puzzle(pc);

                move.puzzle = puzzle;

                move.dir = "UP";

                this.moves.Add(move);   // O(1)

            }

        }

        public Puzzle indexMove(List<int> at, List<int> to)

        {

            int[,] pc = (int[,])this.array.Clone();

            int i, j, r, c;

            i = at[0];

            j = at[1];

            r = to[0];

            c = to[1];

            int temp = pc[i, j];

            pc[i, j] = pc[r, c];

            pc[r, c] = temp;

            return new Puzzle(pc);

        }

        // O(N^2)

        public void PrintPuzzle()

        {

            Console.WriteLine();

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j < this.dim; j++)

                {

                    Console.Write(this.array[i, j] + " ");

                }

                Console.WriteLine();

            }

        }

        // O(N^2)

        public bool Solved()

        {

            return (this.manhattanDistance() == 0);  // O(N^2)

        }

    }

}

Class Node

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NPuzzle

{

    internal class Node

    {

        public Puzzle puzzle;

        Node parent;

        public string action;

        int level;

       public int color;

        // O(1)

        public Node(Puzzle puzzle, Node parent = null, string action = "None",int color=0)

        {

            this.puzzle = puzzle;

            this.parent = parent;

            this.action = action;

            this.color=color;

            if (this.parent != null)

            {

                this.level = parent.level + 1;

            }

            else

                this.level = 0;

        }

        public int AstareScore(char way)

        {

            if (way == 'h')

                return this.level + this.HM();

            else

                return this.level + this.M();

        }

        // O(N^2)

        public int M()

        {

            return this.puzzle.manhattanDistance();  // O(N^2)

        }

        // O(N^2)

        public int HM()

        {

            return this.puzzle.HammingDistance();  // O(N^2)

        }

        // O(L)

        public List<Node> SolutionPath()

        {

            List<Node> path = new List<Node>();

            Node node = this;

            while (node != null)

            {

                path.Add(node);

                node = node.parent;

            }

            path.Reverse();  // O(d)

            return path;

        }

        // O(N^2)

        public bool solved()

        {

            return this.puzzle.Solved(); // O(N^2)

        }

        // O(N^2)

        public List<Move> Actions()

        {

            return this.puzzle.Actions(); // O(N^2)

        }

        // O(N^2)

        public string Step()

        {

            var s = "";

            for (int i = 0; i < this.puzzle.dim; i++)

            {

                for (int j = 0; j < this.puzzle.dim; j++)

                {

                    s += this.puzzle.array[i, j];

                }

            }

            return s;

        }

    }

}

Class Solvable

namespace NPuzzle

{

    internal class Solvable

    {

        Puzzle puzzle;

        int dim;

        public Solvable(Puzzle puzzle)

        {

            this.puzzle=puzzle;

            this.dim=this.puzzle.dim;

        }

        public bool isSolvable()

        {

            List<int> ls = puzzle.getZeroindex(puzzle.array);

            List<int> ls2 = new List<int>();

            bool yIsEven = false;

            if (ls[0] % 2 == 0)

                yIsEven = true;

            else

                yIsEven = false;

            int inversion\_count = 0;

            // O(N^2)

            for (int i = 0; i < this.dim; i++)

            {

                for (int j = 0; j < this.dim; j++)

                {

                    if (puzzle.array[i, j] != 0)

                        ls2.Add(puzzle.array[i, j]);

                }

            }

            // O(N^2)

            for (int i = 0; i < ls2.Count; i++)

            {

                for (int k = i + 1; k < ls2.Count; k++)

                {

                    if (ls2[i] > ls2[k])

                    {

                        inversion\_count += 1;

                    }

                }

            }

            int invCont = inversion\_count;

            bool inversEven = false;

            bool widthEven = false;

            bool zeroOdd = false;

            if (invCont % 2 == 0)

            {

                inversEven = true;

            }

            else

                inversEven = false;

            if (this.dim % 2 == 0)

                widthEven = true;

            else

                widthEven = false;

            if (widthEven)

                zeroOdd = !yIsEven;

            return (!widthEven && inversEven) || (widthEven && (zeroOdd == inversEven));

        }

    }

}

Class A star

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NPuzzle

{

    internal class AStar

    {

        Puzzle start\_node;

        int dim;

        HashSet<string> seen = new HashSet<string>();

        PriorityQueue<Node, int> pq;

        public AStar(Puzzle start\_node, int dim)

        {

            this.start\_node = start\_node;

            this.dim = dim;

        }

        // O(log(L)\*N^2)

        public List<Node> Astar\_seaech(char way)

        {

            pq = new PriorityQueue<Node, int>();

            Node startNode = new Node(start\_node);

            pq.Enqueue(startNode, startNode.AstareScore(way)); //  // O(N^2)

            seen.Add(startNode.Step());

            while (pq.Count > 0)

            {

                Node node = pq.Dequeue();  // log(L)

                if (node.solved()) // O(N^2)

                {

                    return node.SolutionPath(); // O(L)

                }

                //List<Move> list= new List<Move>();

                //list=node.Actions();// O(N^2)

                foreach (Move move in node.Actions())

                {

                    Node child = new Node(move.puzzle, node, move.dir);

                    string step = child.Step(); // O(N^2)

                    if (!seen.Contains(step)) // O(1)

                    {

                        pq.Enqueue(child, child.AstareScore(way)); // O(N^2)

                        seen.Add(step); // O(1)

                    }

                }

            }

            return null;

        }

    }

}

Class DFS

namespace NPuzzle

{

    internal class DFSSearch

    {

        HashSet<string> set = new HashSet<string>();

        Queue<Node> queue = new Queue<Node>();

        Node root;

        int N;

        public DFSSearch(Node root, int N)

        {

            this.root = root;

            this.N = N;

            this.root.color = 1;

        }

        public List<Node> DfsSolve()

        {

            this.queue.Enqueue(this.root);

            //this.set.Add(this.root.Step());

            while (this.queue.Count > 0)

            {

                Node u = this.queue.Dequeue();

                if (u.solved())

                {   return u.SolutionPath();  }

                if (this.set.Contains(u.Step()))

                {  continue;  }

                this.set.Add(u.Step());

                List<Node> adj = new List<Node>();

                adj = this.getAdj(u);

                foreach (Node i in adj)

                {

                    if (i.color == 0)

                    {

                        i.color = 1;

                        this.queue.Enqueue(i);

                    }

                }

                u.color = 2;

            }

            return null;

        }

        public List<Node> getAdj(Node node)

        {

            List<Node> adj = new List<Node>();

            foreach (Move move in node.Actions()) // O(N^2)

            {

                Node child = new Node(move.puzzle, node, move.dir);

                //string step = child.Step(); // O(N^2)

                //this.set.Add(step); // O(1)

                adj.Add(child);

            }

            return adj;

        }

    }

}

Class BFS

using System.Net.Sockets;

namespace NPuzzle

{

    internal class BFSSearch

    {

        HashSet<string> s = new HashSet<string>();

        Queue<Node> queue = new Queue<Node>();

        Node root;

        int N;

        public BFSSearch(Node root, int N)

        {

            this.root = root;

            this.N = N;

            this.root.color = 1;

        }

        public List<Node> BfsSolve()

        {

            this.queue.Enqueue(this.root);

            this.s.Add(this.root.Step());

            while (this.queue.Count > 0)

            {

                Node u = this.queue.Dequeue();

                if (u.solved())

                {

                    return u.SolutionPath();  // o( L )

                }

                List<Node> adj = new List<Node>();

                adj = this.getAdj(u);                // o( s)

                foreach (Node i in adj)

                {

                    if (i.color == 0)

                    {

                        i.color = 1;

                        this.queue.Enqueue(i); // o( 1 )

                    }

                }

                u.color = 2;

            }

            return null;

        }

        public List<Node> getAdj(Node node)

        {

            List<Node> adj = new List<Node>();

            foreach (Move move in node.Actions()) // O(N^2)

            {

                Node child = new Node(move.puzzle, node, move.dir);

                string step = child.Step(); // O(N^2)

                if (!this.s.Contains(step)) // O(1)

                {

                    this.s.Add(step); // O(1)

                    adj.Add(child);

                }

            }

            return adj;

        }

    }

}

Class Program

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Globalization;

using System.Linq.Expressions;

using System.Threading;

namespace NPuzzle

{

    class Program

    {

        public static List<int[,]> getAllCases(string folderPath)

        {

            string path = folderPath;

            string[] lines;

            int sz;

            int[,] arr;

            int i = 0, j = 0;

            List<int[,]> cases = new List<int[,]>();

            DirectoryInfo dir = new DirectoryInfo(path);

            foreach (FileInfo flInfo in dir.GetFiles())

            {

                String name = flInfo.Name;

                long size = flInfo.Length;

                DateTime creationTime = flInfo.CreationTime;

                lines = System.IO.File.ReadAllLines(path + '/' + name);

                sz = Convert.ToInt32(lines[0]);

                arr = new int[sz, sz];

                i = 0;

                j = 0;

                foreach (string line in lines)

                {

                    if (i > 1)

                    {

                        string[] s = lines[i].Split(' ');

                        foreach (string ele in s)

                        {

                            if (ele != "")

                            {

                                arr[i - 2, j] = int.Parse(ele);

                                j++;

                            }

                        }

                    }

                    i++;

                    j = 0;

                }

                cases.Add(arr);

            }

            return cases;

        }

        public static void Main(string[] args)

        {

            bool loop = true;

            while (loop)

            {

                string folder = "";

                Console.WriteLine("Sample   Test Solvable Puzzles ?                     => 1");

                Console.WriteLine("Sample   Test Unsolvable Puzzles ?                   => 2");

                Console.WriteLine("Complete Test Solvable puzzles Manhattan & Hamming ? => 3");

                Console.WriteLine("Complete Test Solvable puzzles Manhattan Only ?      => 4");

                Console.WriteLine("Complete Test V. Large test case ?                   => 5");

                Console.WriteLine("Complete Test Unsolvable puzzles ?                   => 6");

                Console.Write("Enter The Tests Number                               => ");

                int v = int.Parse(Console.ReadLine());

                char way;

                switch (v)

                {

                    case 1:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Sample Solvable Puzzles";

                        break;

                    case 2:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Sample Unsolvable Puzzles";

                        break;

                    case 3:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Complete Solvable puzzles\Manhattan & Hamming";

                        break;

                    case 4:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Complete Solvable puzzles\Manhattan Only";

                        break;

                    case 5:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Complete V. Large test case";

                        break;

                    case 6:

                        folder = @"E:\computer science\level3\term 2\algo\NPuzzle\NPuzzle\test\_cases\Complete Unsolvable puzzles";

                        break;

                }

                Stopwatch stopwatch = new Stopwatch();

                if (folder != "")

                {

                    List<int[,]> cases = new List<int[,]>();

                    cases = getAllCases(folder);

                    Console.WriteLine("Astar algorithm    ?                       => 1");

                    Console.WriteLine("BFS   algorithm    ?                       => 2");

                    Console.WriteLine("DFS   algorithm    ?                       => 3");

                    Console.Write("Enter The Alog Number                      => ");

                    int algo = int.Parse(Console.ReadLine());

                    //int algo=1;

                    Console.WriteLine("We Have " + cases.Count + " Tests");

                    way = 'm';

                    if (algo == 1)

                    {

                        if (v == 1 || v == 3)

                        {

                            Console.Write("which method you want  [M , H] ?");

                            way = char.Parse(Console.ReadLine());

                        }

                    }

                    for (int x = 0; x < cases.Count; x++)

                    {

                        for (int i = 0; i < cases[x].GetLength(0); i++)

                        {

                            for (int j = 0; j < cases[x].GetLength(0); j++)

                            {

                                Console.Write(cases[x][i, j] + " ");

                            }

                            Console.WriteLine();

                        }

                        Puzzle p = new Puzzle(cases[x]);

                        Node n = new Node(p);

                        Solvable solvable = new Solvable(p);

                        AStar aStar = new AStar(p, cases[x].GetLength(0));

                        BFSSearch bFSSearch = new BFSSearch(n, cases[x].GetLength(0));

                        DFSSearch dFSSearch = new DFSSearch(n, cases[x].GetLength(0));

                        if (solvable.isSolvable())

                        {

                            Console.WriteLine("Is  Solveble............ :)");

                            List<Node> ll = new List<Node>();

                            if (algo == 1)

                            {

                                stopwatch.Start();

                                ll = aStar.Astar\_seaech(way);

                                stopwatch.Stop();

                            }

                            else if (algo == 2)

                            {

                                stopwatch.Start();

                                ll = bFSSearch.BfsSolve();

                                stopwatch.Stop();

                            }

                            else if (algo == 3)

                            {

                                stopwatch.Start();

                                ll = dFSSearch.DfsSolve();

                                stopwatch.Stop();

                            }

                            // foreach (Node node in ll)

                            // {

                            //     Console.WriteLine(node.action);

                            //     node.puzzle.PrintPuzzle();

                            // }

                            Console.WriteLine("Total number of steps in : " + (ll.Count - 1));

                            Console.WriteLine("Total amount of time in search:  {0}  Second  ,  {1}  MS", stopwatch.ElapsedMilliseconds / 1000, stopwatch.ElapsedMilliseconds % 1000);

                            Console.WriteLine("---------------------------------------------------------------------------------------------");

                        }

                        else

                        {

                            Console.WriteLine("Not solveble............ :(");

                            Console.WriteLine("---------------------------------------------------------------------------------------------");

                        }

                    }

                }

                char k = ' ';

                Console.Write("You Want To Mack Another Operation [y,n] ");

                k = char.Parse(Console.ReadLine());

                if (k == 'y')

                {

                    loop = true;

                }

                else if (k == 'n')

                {

                    loop = false;

                }

            }

        }

    }

}

Thanks ……😊