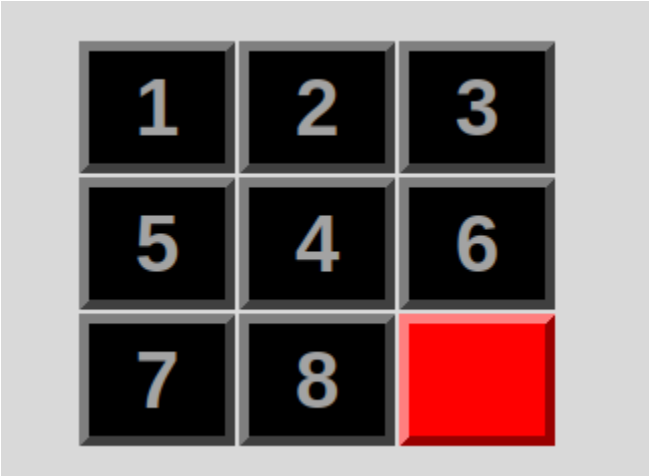
N puzzle Project

Team members:

Name	ID	Department	Section
مروان حاتم فهمي عليوة	20191700617	CS	4
نيره خالد محمد حسني	20191700721	No department	5
نيفين احمد إبراهيم محمد	20191700722	SC	5



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace N_Puzzle
{
    internal class Solvability
    {
        public bool Resolvable(List<int> l, int N)  // O(S²)
        {
             // this function check if the given puzzle is solvable or not
            int counter = 0;
            decimal index = 0;
            for (int i = 0; i < l.Count; i++)           // O(S) * O(S)
            {
                if (l[i] == 0)
                {
                    index = (i / N) + 1;
                    index = Math.Ceiling(index);
                    continue;
                }
                for (int j = i + 1; j < l.Count; j++)          // O(S)
{
                    if (l[j] == 0)           //O(S)
                    {

                        continue;
                    }
                    if (l[i] > l[j])         //O(S)
                    {
                        counter++;
                    }
                }

            }
            int c = counter;
            if (N % 2 != 0 && counter % 2 == 0)//odd o(1)
            {
                return true;
            }
            else if (N % 2 == 0 && counter % 2 != 0 && index % 2 != 0)//even o(1)
            {
                return true;
            }
            else if (N % 2 == 0 && counter % 2 == 0 && index % 2 == 0)//even o(1)
            {
                return true;
            }
            else    //o(1)
                return false;

        }
    }
}
/// Total Complexity O(S²)
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


struct Puzzle
{
    public List<int> matrix;
    public int NPuzzle;                                          → θ(1)
}
namespace N_Puzzle
{
    internal class A_Star
    {
        List<int> Initial_state;
        static int size;
        int depth = 0;                                           → θ(1)
        HashSet<string> exist;
        PriorityQueue<State, int> priorty;
        public A_Star(List<int> Initial_state, int size)  // θ(1)        {
            this.Initial_state = Initial_state;
            A_Star.size = size;
            exist = new HashSet<string>();
            priorty = new PriorityQueue<State, int>();

        }
        public void start_solver(List<int> Initial_state, int N, Char name) // O(S²)
        {
             // this function starts the A* algorithm after checking solvability
            Solvability solvability = new Solvability();  // θ(1)
            bool flag = true;                                  // O(1)
            flag = solvability.Resolvable(Initial_state, N); // O(S²)
            if (flag == true)   // O(E log(V))
            {
                Console.WriteLine(" This Puzzle is solvable");  // θ(1)
                DateTime dt = DateTime.Now;   // θ(1)
                int[] temp = FindBlank(Initial_state);   // O(S)
                int y = temp[0];
                int x = temp[1];               // θ(1)
                int temp_cost;
                if (name == 'h')  /// O(S)
                {
                    temp_cost = Calc_Hamming(Initial_state);   // O(S)
                }
                Else            // O(S)
                {
                    temp_cost = Calc_Manhatin(Initial_state);  // O(S)
                }
              State node=new State(Initial_state, temp_cost, 0, null, x, y); // θ(1)
                exist.Add(hash_fun(Initial_state));  // O(S)
                priorty.Enqueue(node, temp_cost);   // O(log(v))

                while (true) // O(E log(V))
                {
                    node = priorty.Dequeue();  // O(log(v))
                    depth = node.level;      // θ(1)
```

```
                    if (node.cost - node.level == 0)  // θ(1)
                    {
                        break;
                    }
                    depth++;
                    find_childs(name, node); // O(S)
                }
                DateTime dt2 = DateTime.Now;  θ(1)
                TimeSpan result = dt2 - dt;
                Console.WriteLine("time to run this test in seconds:  " +
result.Seconds.ToString());
                Console.WriteLine("time to run this test in Milliseconds:  " +
result.Milliseconds.ToString());
                Console.WriteLine("number of steps " + "with " + name + " = " +
depth + "\n" +
                    "-------------------------------------------------------------
----- \n");
                if(size == 3)  // O(S log(v))
                {
                   this.get_path(node);  // O(S log(v))
                }
            }
            Else  // θ(1)
            {
                Console.WriteLine("The Puzzle is unsolvable \n " +
                    "---------------------------------------");
            }

        }

  public void find_childs(Char name, State node) // O(S)
        {
             // this function finds all possible children of a given state
            int x = node.x_zero;
            int y = node.y_zero;
            //Moves
            if(x + 1 < size) // O(log(V)
            {
                swap(node.matrix, y, x + 1, x, y);  // O(1)
                if (exist.Contains(hash_fun(node.matrix)) == false) // O(S)
                {
                    Add_child(node, name, x + 1, y);  // O(log(V))
                }
                swap(node.matrix, y, x, x + 1, y);  // O(1)
            }
            if(y - 1 >= 0)      // O(log(V)
            {
                swap(node.matrix, y - 1, x, x, y);
                if (exist.Contains(hash_fun(node.matrix)) == false)
                {
                    Add_child(node, name, x, y - 1);
                }
                swap(node.matrix, y, x, x, y - 1);
            }
            if(x - 1 >= 0)      // O(log(V)
            {
                swap(node.matrix, y, x - 1, x, y);
                if (exist.Contains(hash_fun(node.matrix)) == false)
```

```
                {
                    Add_child(node, name, x - 1, y);
                }
                swap(node.matrix, y, x, x - 1, y);
            }
            if(y + 1 < size)    // O(log(V)
            {
                swap(node.matrix, y + 1, x, x, y);
                if (exist.Contains(hash_fun(node.matrix)) == false)
                {
                    Add_child(node, name, x, y + 1);
                }
                swap(node.matrix, y, x, x, y + 1);
            }

        }

        public int Calc_Hamming(List<int> state)  // O(S)
        {
             // this function calculate hamming distance for a state
            int count = 0;
            for (int i = 0; i < size; i++)   // O(N) * O(N)
            {
                for (int j = 0; j < size; j++)   // O(N)
                {
                    int actual = state[i * size + j];
                    int expected = (i * size) + j + 1;
                    if (actual == 0 || actual == expected)
                    {
                        continue;
                    }
                    else
                    {
                        count++;
                    }

                }
            }
            return count + depth;
        }
        public int update_hamming(State node, int new_x_zero, int new_y_zero)// O(1)
        {
             //this function calculate hamming distance with respect to parent cost
            int temp = node.cost;
            if(node.matrix[ node.y_zero*size + node.x_zero ]==
node.y_zero*size+node.x_zero+1)
            {
                temp--;
            }
            else if(node.matrix[node.y_zero * size + node.x_zero]
==new_y_zero*size+new_x_zero+1 )
{
                temp++;
            }
            return temp + 1;
        }
```

→// θ(1)

→θ(1)

→O(1)

```
public int Calc_Manhatin(List<int> state)   // O(S)
        {
             // this function calculate manhatan distance for a state

            int sum = 0;
            for (int i = 0; i < state.Count; i++) // O(S)
            {
                int actual = state[i];
                int expected = i + 1;
                if (actual == 0 || actual == expected)     // θ(1)
                {
                    continue;
                }
                Else                            // O(1)
                {
                    int X_orig = state[i] % size;
                    int Y_orig;
                    if (X_orig == 0)
                    {
                        X_orig = size;
                        Y_orig = (state[i] / size);
                    }
                    else                                                    // θ(1)
                    {
                        Y_orig = (state[i] / size) + 1;
                    }
                    int current_X = (i + 1) % size;
                    if (current_X == 0)
                    {
                        current_X = size;
                    }
                    int current_Y = (i / size) + 1;
                    int temperary = Math.Abs(X_orig - current_X) + Math.Abs(Y_orig -
current_Y);
                    sum += temperary;
                }
            }
            return sum + depth;  // →θ(1)
        }
        public int updaate_manh(State node, int new_x_zero, int new_y_zero)  // O(1)
        {
             //this function calculate manhatan distance with respect to parent cost
            int temp_cost = node.cost;
            int acutal = node.matrix[node.y_zero*size+node.x_zero];
            int X_orig = acutal % size;
            int Y_orig;
            if (X_orig == 0)
            {
                X_orig = size;
                Y_orig = (acutal / size);
            }
            else
            {
                Y_orig = (acutal / size) + 1;                                   // →θ(1)
            }
            X_orig--;
            Y_orig--;
            int old_cost = Math.Abs(X_orig - new_x_zero) +
```

```
            Math.Abs(Y_orig - new_y_zero);
        int new_cost = Math.Abs(X_orig - node.x_zero) +
                Math.Abs(Y_orig - node.y_zero);
        temp_cost -= old_cost;
        temp_cost += new_cost;
        return temp_cost +1;
    }
    public int[] FindBlank(List<int> state)  // O(S)
    {
         // this function searchs for the zero position in a state
        int[] temp = new int[2];
        for (int y = 0; y < size; y++)  // O(N) * O(N)
        {
            for (int x = 0; x < size; x++)  // O(N)
            {
                int actual = state[y * size + x];
                if (actual == 0)
                {
                    temp[0] = y;
                    temp[1] = x;
                }
            }
        }
        return temp;
    }
    //this fun takes (y,x,x,y)
    public List<int> swap(List<int>state,int i,int j,int x_zero,int y_zero)//O(1)
    {
         //swaps two elements in list
        int temperary = state[y_zero * size + x_zero];
        state[y_zero * size + x_zero] = state[i * size + j];
        state[i * size + j] = temperary;
        return state;
    }
    public void show(List<int> state)   // O(S)
    {
         // displays a given list as N*N matrix
        for (int i = 0; i < size; i++)  // O(N) * O(N)
        {
            for (int j = 0; j < size; j++)   // O(N)
            {
                Console.Write(state[i * size + j] + " ");
            }
            Console.WriteLine("\n");
        }
        Console.WriteLine("*****************************");
    }
    public void get_path(State node)  // O(S)
    {
         //stores all Vertices all the way up to the initial state
        List<List<int>> path = new List<List<int>>();
        while (node.Parent != null)  // O(log(V))
        {
            path.Add(node.matrix);  // →θ(1)
            node = node.Parent;    // →θ(1)
        }
        path.Add(node.matrix);  // →θ(1)
        for (int i = path.Count - 1; i >= 0; i--) // O(S log(v))
```

```
            {
                this.show(path[i]); // O(S)
            }
        }

        public int mh(Char name, State node , int new_x_zero, int new_y_zero)// O(1)
        {
             // decides what distance function to use according to name
            if (name == 'h')
            {
                return update_hamming(node, new_x_zero, new_y_zero);
            }                                                                  // θ(1)
            else
            {
                return updaate_manh(node, new_x_zero, new_y_zero);
            }
        }
        ///// function to calc coast
     public void Add_child(State node,Char name,int new_x_zero,int new_y_zero)
//O(log(v))
        {
             // adds new child
            List<int> temp_list = new List<int>(node.matrix);
            int cost = mh(name, node, new_x_zero, new_y_zero);  // O(1)
            exist.Add(hash_fun(temp_list));  // O(1)
            State temp_state = new State(temp_list, cost, depth, node, new_x_zero,
new_y_zero);
            priorty.Enqueue(temp_state, temp_state.cost); //O(log(v))

        }

        public string hash_fun(List<int> list)   // O(S)
        { //converts the list to string
            string result = " ";

            for (int i = 0; i < list.Count; i++) // O(S)
            {

                result += (list[i].ToString());

            }
            return result;
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace N_Puzzle
{
    internal class State    → Total of O(1)
    {
        public List<int> matrix;
        public int cost;
        public int level;
        public State Parent;
         public int x_zero, y_zero;

        public State(List<int> matrix, int cost, int level, State Parent, int
x_zero, int y_zero)
        {
            this.matrix = matrix;
            this.cost = cost;
            this.level = level;
            this.Parent = Parent;
             this.x_zero = x_zero;
            this.y_zero = y_zero;

        }


    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace N_Puzzle
{

    internal class filehelper
    {Puzzle ReadTest(string filename)//space include      // O(S)
        {
             //read test from file
            Solvability solvability = new Solvability();
            FileStream file;
            int N;
            StreamReader sr;
            string line;
            file = new FileStream(filename, FileMode.Open,                  //θ(1)
                                  FileAccess.Read);
            sr = new StreamReader(file);
            string[] temp;
            line = sr.ReadLine();
            N = int.Parse(line);
            line = sr.ReadLine();
            Puzzle state;
            List<int> puzzle = new List<int>();           → O(1)
            for (int i = 0; i < N; i++)      // O(N) * O(N)
            {
                line = sr.ReadLine();
                temp = line.Split(' ');
                for (int j = 0; j < N; j++)  // O(N)
                {
                    puzzle.Add(int.Parse(temp[j]));
                }
            }
            sr.Close();
            file.Close();                                   //θ(1)
            state.NPuzzle = N;
            state.matrix = puzzle;
            return state;
        }
        /// ///

        Puzzle ReadTest2(string filename)//space include  // O(S)
        {
            Solvability solvability = new Solvability();
            FileStream file;
            int N;
            StreamReader sr;
            string line;
            file = new FileStream(filename, FileMode.Open, FileAccess.Read);
            sr = new StreamReader(file);
            string[] temp;
            line = sr.ReadLine();
            N = int.Parse(line);
            Puzzle state;
            List<int> puzzle = new List<int>();
            for (int i = 0; i < N; i++)
```

```
            {
                line = sr.ReadLine();
                temp = line.Split(' ');
                for (int j = 0; j < N; j++)
                {
                    puzzle.Add(int.Parse(temp[j]));
                }
            }
            sr.Close();
            file.Close();
            state.NPuzzle = N;
            state.matrix = puzzle;
            return state;
        }


        void solvePuzzle(Puzzle puzzle, char theWay) //O(S²)

        {
             //create new puzzle from reading files and then solve it
            List<int> test = puzzle.matrix; //θ(1)
            int N = puzzle.NPuzzle;
            A_Star a_star = new A_Star(test, N);
            a_star.start_solver(test, N, theWay);  //O(S²)

        }


        public void read()
        {
             // connects the functions in class
            Char cont = 'y';


            while (cont == 'y')
            {

                Console.WriteLine("[1] Sample test cases\n[2] Manhatten
                                                    only test\n" +
                    "[3] Hamming & Manhatten test\n[4] V-Large test\n");
                Console.Write("\nEnter your choice [1-2-3-4]: ");
                char choice = Console.ReadLine()[0];

                switch (choice) //O(S²)
                {
                    case '1':
                        #region SAMPLE CASES
                        Puzzle p1 = ReadTest("Sample Test\\Solvable Puzzles\\8
Puzzle (1).txt");  //O(S)
                        Console.WriteLine("test file name: 8 Puzzle (1).");
                        solvePuzzle(p1, 'm');  //O(S²)
                        // test2
                        Puzzle p2 = ReadTest("Sample Test\\Solvable Puzzles\\8
Puzzle (2).txt");  //O(S)
                        Console.WriteLine("test file name: 8 Puzzle (2).");
                        solvePuzzle(p2, 'm');   //O(S²)
                        //// test3
```

//θ(1)

```
                    Puzzle p3 = ReadTest("Sample Test\\Solvable Puzzles\\8
Puzzle (3).txt");   //O(S)
                    Console.WriteLine("test file name: 8 Puzzle (3).");
                    solvePuzzle(p3, 'm');  //O(S²)
                    //////// test4
                    Puzzle p4 = ReadTest("Sample Test\\Solvable Puzzles\\15
Puzzle - 1.txt");
                    Console.WriteLine("test file name: 15 Puzzle - 1.");
                    solvePuzzle(p4, 'm');
                    //////// test5
                    Puzzle p5 = ReadTest("Sample Test\\Solvable Puzzles\\24
Puzzle 1.txt");
                    Console.WriteLine("test file name: 24 Puzzle 1.");
                    solvePuzzle(p5, 'm');
                    //////// test6
                    Puzzle p6 = ReadTest("Sample Test\\Solvable Puzzles\\24
Puzzle 2.txt");
                    Console.WriteLine("test file name: 24 Puzzle 2.");
                    solvePuzzle(p6, 'm');
                    /////// unsolvable
                    ////// test1
                    Puzzle pp1 = ReadTest("Sample Test\\Unsolvable Puzzles\\8
Puzzle - Case 1.txt");
                    Console.WriteLine("test file name: 8 Puzzle - Case 1.");
                    solvePuzzle(pp1, 'm');
                    //// test2
                    Puzzle pp2 = ReadTest("Sample Test\\Unsolvable Puzzles\\8
Puzzle(2) - Case 1.txt");
                    Console.WriteLine("test file name: 8 Puzzle(2) - Case 1.");
                    solvePuzzle(pp2, 'm');
                    //// test3
                    Puzzle pp3 = ReadTest("Sample Test\\Unsolvable Puzzles\\8
Puzzle(3) - Case 1.txt");
                    Console.WriteLine("test file name: 8 Puzzle(3) - Case 1.");
                    solvePuzzle(pp3, 'm');
                    //// test4
                    Puzzle pp4 = ReadTest("Sample Test\\Unsolvable Puzzles\\15
Puzzle - Case 2.txt");
                    Console.WriteLine("test file name: 15 Puzzle - Case 2.");
                    solvePuzzle(pp4, 'm');
                    //// test5
                    Puzzle pp5 = ReadTest("Sample Test\\Unsolvable Puzzles\\15
Puzzle - Case 3.txt");
                    Console.WriteLine("test file name: 15 Puzzle - Case 3.");
                    solvePuzzle(pp5, 'm');
                    break;
                #endregion
                case '3':
                    #region Manhattan & Hamming
                    Console.WriteLine("[1] Hamming test \n[2] Manhatten
test\n");
                    Console.Write("\nEnter your choice [1-2]: ");
                    char Choice = Console.ReadLine()[0];
                    // test1
                    switch (Choice)
                    {
                        case '1':
                            //(hamming)
```

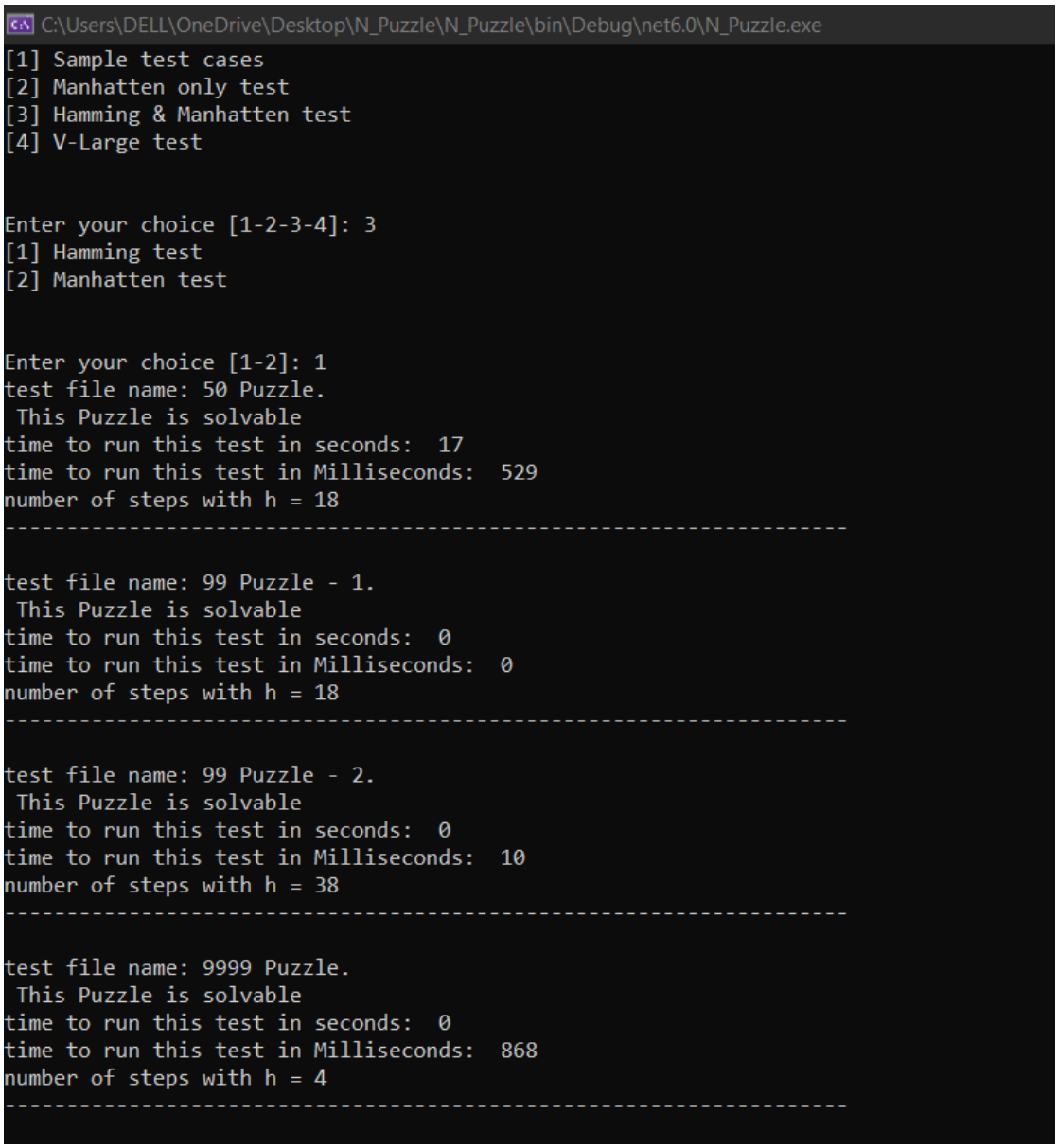
```
                                Puzzle ppp1 = ReadTest2("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\50 Puzzle.txt");
                                Console.WriteLine("test file name: 50 Puzzle.");
                                solvePuzzle(ppp1, 'h');
                                Puzzle ppp3 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\99 Puzzle - 1.txt");
                                Console.WriteLine("test file name: 99 Puzzle - 1.");
                                solvePuzzle(ppp3, 'h');
                                Puzzle ppp5 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\99 Puzzle - 2.txt");
                                Console.WriteLine("test file name: 99 Puzzle - 2.");
                                solvePuzzle(ppp5, 'h');
                                Puzzle ppp7 = ReadTest2("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\9999 Puzzle.txt");
                                Console.WriteLine("test file name: 9999 Puzzle.");
                                solvePuzzle(ppp7, 'h');
                                break;
                            case '2':
                                //(manhattan)
                                Puzzle ppp2 = ReadTest2("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\50 Puzzle.txt");
                                Console.WriteLine("test file name: 50 Puzzle.");
                                solvePuzzle(ppp2, 'm');
                                // (manhattan)
                                Puzzle ppp4 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\99 Puzzle - 1.txt");
                                Console.WriteLine("test file name: 99 Puzzle - 1.");
                                solvePuzzle(ppp4, 'm');
                                ////(manhattan)
                                Puzzle ppp6 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\99 Puzzle - 2.txt");
                                Console.WriteLine("test file name: 99 Puzzle - 2.");
                                solvePuzzle(ppp6, 'm');
                                ////(manhattan)
                                Puzzle ppp8 = ReadTest2("Complete Test\\Solvable
puzzles\\Manhattan & Hamming\\9999 Puzzle.txt");
                                Console.WriteLine("test file name: 9999 Puzzle.");
                                solvePuzzle(ppp8, 'm');
                                break;
                        }
                        break;
                    #endregion
                    case '2':
                        #region Manhattan only
                        // /////Manhattan Only
                        // //////test1
                        Puzzle pppp1 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan Only\\15 Puzzle 1.txt");
                        Console.WriteLine("test file name: 15 Puzzle 1.");
                        solvePuzzle(pppp1, 'm');
                        //////test2
                        Puzzle pppp2 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan Only\\15 Puzzle 3.txt");
                        Console.WriteLine("test file name: 15 Puzzle 3.");
                        solvePuzzle(pppp2, 'm');
                        ////////test3
                        Puzzle pppp5 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan Only\\15 Puzzle 4.txt");
```

```
                        Console.WriteLine("test file name: 15 Puzzle 4.");
                        solvePuzzle(pppp5, 'm');
                        //test1
                        Puzzle pppp3 = ReadTest("Complete Test\\Solvable
puzzles\\Manhattan Only\\15 Puzzle 5.txt");
                        Console.WriteLine("test file name: 15 Puzzle 5.");
                        solvePuzzle(pppp3, 'm');
                        /////unsolvable
                        //////test1
                        Puzzle unsolv1 = ReadTest("Complete Test\\Unsolvable
puzzles\\15 Puzzle 1 - Unsolvable.txt");
                        Console.WriteLine("test file name: 15 Puzzle 1 -
Unsolvable.");
                        solvePuzzle(unsolv1, 'm');
                        //////test2
                        Puzzle unsolv2 = ReadTest("Complete Test\\Unsolvable
puzzles\\99 Puzzle - Unsolvable Case 1.txt");
                        Console.WriteLine("test file name: 99 Puzzle - Unsolvable
Case 1.");
                        solvePuzzle(unsolv2, 'm');
                        //////test3
                        Puzzle unsolv3 = ReadTest("Complete Test\\Unsolvable
puzzles\\99 Puzzle - Unsolvable Case 2.txt");
                        Console.WriteLine("test file name: 99 Puzzle - Unsolvable
Case 2.");
                        solvePuzzle(unsolv3, 'm');
                        //////test4
                        Puzzle unsolv4 = ReadTest("Complete Test\\Unsolvable
puzzles\\9999 Puzzle.txt");
                        Console.WriteLine("test file name: 9999 Puzzle.");
                        solvePuzzle(unsolv4, 'm');
                        break;
                    #endregion
                    case '4':
                        #region V-Large
                        /// very large
                        Puzzle veryLarge = ReadTest("Complete Test\\V. Large test
case\\TEST.txt");
                        Console.WriteLine("test file name: TEST.");
                        solvePuzzle(veryLarge, 'm');
                        break;
                    #endregion

                }

Console.WriteLine("**************************************************\n" +
                    "\n Do you want to continue (y-n)");
                cont = Console.ReadLine()[0];
                Console.WriteLine("\n\n");
            }
        }
```

```
/// main
using N_Puzzle;

filehelper filehelper=new filehelper();   //O(1)
filehelper.read();  //O(S²)
```

Manhattan VS Hamming

1. **Manhattan Distance** is used to calculate the distance between two data points in a grid like path.



2. **Hamming distance** is used to measure how many attributes must be changed in order to match one another.