N Puzzle

Team Members :

- **Mohamed Emad [20191700564] [CS]**
- **Aya Hussien [20191700878] [IS]**
- **Ali Taher [20191700391] [IS]**

N Puzzle Definition

N-Puzzle Application is a program to solve N-Puzzle Game Using AI algorithm (A*&BFS) with some help from other Algorithms like Priority Queue and Dictionary.

About Project Logic Steps to Solve N-Puzzle:

1-Reading a puzzle from file.

2-Read the puzzle and pass the puzzle Data to Class Puzzle.

3- Decide If this Puzzle is solvable or not.

4-After deciding If the puzzle solvable, Puzzle details and data are sent to A*(AS) class which uses A* search Algorithm with help from priority Queue and Dictionary and solve it using Manhattan or hamming And BFS.

5-After Find the Goal, Goal Node return its Parents (Steps Path).

Priority Queue.CS :

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

namespace Puzzel
{
    class PriorityQueue
    {

        public List<puzzel> data;  //O(1)
        public PriorityQueue()
        {
            this.data = new List<puzzel>();//O(1)
        }
        public void Enqueue(puzzel item) //O(Log N)
        {
            data.Add(item);//O(1)
            heapifyup();//O(Log N)

        }
        public void heapifyup() //O(Log N)
        {
            int pos = data.Count - 1; //O(1)
            while (pos > 0) //O(Log N)
            {
                int parent_index = (pos - 1) / 2; //O(1)

                // If child item is larger than (or equal) parent so we're done
                if (data[pos].cost.CompareTo(data[parent_index].cost) >= 0) break; //O(1)
                // Else swap parent & child
                puzzel tmp = data[pos]; data[pos] = data[parent_index];
data[parent_index] = tmp; //O(1)
                pos = parent_index; //O(1)
            }
        }
        public puzzel Dequeue() //O(Log N)
        {
            // assumes pq is not empty; up to calling code
            int pos = data.Count - 1; // last index (before removal) //O(1)
            puzzel frontItem = data[0];   // fetch the front //O(1)
            data[0] = data[pos]; // last item be the root //O(1)
            data.RemoveAt(pos); //O(1)
            heapifydown();  //O(Log N)
            return frontItem;
        }
        //*********************************************************************
        public void heapifydown() //O(Log N)
        {
            int pos = data.Count - 1; //last index (after removal) //O(1)
```

```
            int parent_index = 0; // parent index. start at front of pq //O(1)
            while (true) //O(Log N)
            {
                int child_index = parent_index * 2 + 1; // left child index of parent
//O(1)
                if (child_index > pos) break;  // no children so done //O(1)
                int right_child = child_index + 1;     // right child //O(1)
                // if there is a right child , and it is smaller than left child, use the
rc instead
                if (right_child <= pos &&
data[right_child].cost.CompareTo(data[child_index].cost) < 0) //O(1)
                    child_index = right_child; //O(1)
                // If parent is smaller than (or equal to) smallest child so done
                if (data[parent_index].cost.CompareTo(data[child_index].cost) <= 0)
break; //O(1)
                // Else swap parent and child
                puzzel tmp = data[parent_index]; data[parent_index] = data[child_index];
data[child_index] = tmp; //O(1)
                parent_index = child_index; //O(1)
            }
        }
        public int Size() //O(1)
        {
            return data.Count; //O(1)
        }
        public Boolean is_empty() //O(1)
        {
            if (data.Count == 0) return true; //O(1)
            else return false; //O(1)
        }

    }
}
```

Puzzle.CS:

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

namespace Puzzel
{
    class puzzel
    {
        private int Size; //O(1)

        private int Number_Of_Levels; //O(1)
        private int Row_index_of_zero; //O(1)
        private int Manhattan_Sum; //O(1)
        public string direction; //O(1)
        public int cost; //O(1)
```

```
private int Col_index_of_zero; //O(1)
private int Hamming_Sum; //O(1)
public int[,] Node2d; //O(1)
public puzzel parent; //O(1)
public string key = ""; //O(1)
//*****************************************************************
//initial constructor
public puzzel(int size, int[,] puzz, int r, int c) //O(N^2)
{
    this.Size = size; //O(1)
    this.Node2d = new int[Size, Size]; //O(1)
    for (int i = 0; i < size; i++) //O(N^2)
    {
        for (int j = 0; j < size; j++) //O(N)
        {
            this.Node2d[i, j] = puzz[i, j]; //O(1)
            this.key += puzz[i, j]; //O(1)
        }
    }
    this.direction = "Root"; //O(1)
    this.Row_index_of_zero = r;//O(1)
    this.Col_index_of_zero = c; //O(1)
    this.Number_Of_Levels = 0; //O(1)

    this.parent = null; //O(1)
}
//*****************************************************************
//child constructor
public puzzel(puzzel p) //O(N^2)
{
    this.Size = p.Size; //O(1)
    this.Node2d = new int[Size, Size]; //O(1)
    for (int i = 0; i < this.Size; i++) //O(N^2)
    {
        for (int j = 0; j < this.Size; j++) //O(1)
        {

            this.Node2d[i, j] = p.Node2d[i, j]; //O(1)
        }
    }
    this.Row_index_of_zero = p.Row_index_of_zero; //O(1)
    this.Col_index_of_zero = p.Col_index_of_zero; //O(1)
    this.Number_Of_Levels = p.Number_Of_Levels + 1; //O(1)
    this.parent = p.parent; //O(1)

}
//*****************************************************************
//Equal constructor
public puzzel(puzzel p, int _default) //O(N^2)
{
    this.Size = p.Size; //O(1)
    this.Node2d = new int[Size, Size]; //O(1)
    for (int i = 0; i < this.Size; i++) //O(N^2)
    {
        for (int j = 0; j < this.Size; j++) //O(N)
        {

            this.Node2d[i, j] = p.Node2d[i, j]; //O(1)
```

```
                }
            }
            this.Row_index_of_zero = p.Row_index_of_zero; //O(1)
            this.Col_index_of_zero = p.Col_index_of_zero; //O(1)
            this.Number_Of_Levels = p.Number_Of_Levels; //O(1)
            this.Hamming_Sum = p.Hamming_Sum; //O(1)
            this.Manhattan_Sum = p.Manhattan_Sum; //O(1)
            this.cost = p.cost; //O(1)
            this.parent = p; //O(1)
            this.key = p.key;//O(1)

        }

        //*********************************************************************
        //calc min cost using hamming
        public void Calc_Min_Cost_Hamm() //O(1)
        {
            this.cost = this.Number_Of_Levels + this.Hamming_Sum; //O(1)
        }
        //*********************************************************************
        //calc min cost using manhattan
        public void Calc_Min_Cost_Man() //O(1)
        {
            this.cost = this.Number_Of_Levels + this.Manhattan_Sum; //O(1)
        }

        //*********************************************************************
        public puzzel Move(string type) //O(1)
        {
            if (type == "Left") //O(1)
            {
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero] =
this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero - 1];
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero - 1] = 0;
//O(1)
                //catch blank_tile
                this.Col_index_of_zero = this.Col_index_of_zero - 1; //O(1)
            }
            else if (type == "Right") //O(1)
            {
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero] =
this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero + 1];
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero + 1] = 0;
//O(1)
                //catch blank_tile
                this.Col_index_of_zero = this.Col_index_of_zero + 1; //O(1)
            }
            else if (type == "Up") //O(1)
            {
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero] =
this.Node2d[this.Row_index_of_zero - 1, this.Col_index_of_zero];
                this.Node2d[this.Row_index_of_zero - 1, this.Col_index_of_zero] = 0;
//O(1)
                //catch blank_tile
                this.Row_index_of_zero = this.Row_index_of_zero - 1; //O(1)
            }
            else if (type == "Down") //O(1)
            {
```

```
                this.Node2d[this.Row_index_of_zero, this.Col_index_of_zero] =
this.Node2d[this.Row_index_of_zero + 1, this.Col_index_of_zero];
                this.Node2d[this.Row_index_of_zero + 1, this.Col_index_of_zero] = 0;
//O(1)
                //catch blank_tile
                this.Row_index_of_zero = this.Row_index_of_zero + 1; //O(1)
            }
            return this;
        }
        public bool Check(string type) //O(1)
        {
            if (type == "Left") //O(1)
            {
                if (this.Col_index_of_zero != 0) //O(1)
                    return true; //O(1)
                return false;
            }
            else if (type == "Right") //O(1)
            {
                if (this.Col_index_of_zero != this.Size - 1) //O(1)
                    return true;
                return false;
            }
            else if (type == "Up") //O(1)
            {
                if (this.Row_index_of_zero != 0) //O(1)
                    return true; //O(1)
                return false;
            }
            else if (type == "Down") //O(1)
            {
                if (this.Row_index_of_zero != this.Size - 1) //O(1)
                    return true; //O(1)
                return false;
            }
            else
            {
                return true; //O(1)
            }

        }
        //*********************************************************************
        public void display() //O(N^2)
        {
            for (int i = 0; i < this.Size; i++) //O(N^2)
            {
                for (int j = 0; j < this.Size; j++) //O(N)
                {
                    Console.Write(this.Node2d[i, j]); //O(1)
                    Console.Write(" ");
                }
                if (i == 0) { Console.WriteLine("         " + this.direction); } //O(1)
                else { Console.WriteLine(); }
            }
            Console.WriteLine();
        }

        //*********************************************************************
```

```
        public void Generate_Hamming() //O(N^2)
        {
            int counter = 1; //O(1)
            for (int i = 0; i < this.Size; i++) //O(1)
            {
                for (int j = 0; j < this.Size; j++) //O(1)
                {
                    this.key += this.Node2d[i, j]; //O(1)
                    if (this.Node2d[i, j] != counter && this.Node2d[i, j] != 0) {
this.Hamming_Sum++; } //O(1)
                    counter++; //O(1)
                }
            }
        }
        //*************************************************************************
        public void Generate_Man() //O(N^2)
        {
            int count = 0; //O(1)
            int expected = 0; //O(1)
            for (int row = 0; row < this.Size; row++) //ON^2)
            {

                for (int col = 0; col < this.Size; col++) //O(N)
                {
                    this.key += this.Node2d[row, col]; //O(1)
                    int value = this.Node2d[row, col]; //O(1)
                    expected++; //O(1)
                    if (value != 0 && value != expected) //O(1)
                    {

                        count += Math.Abs(row - ((value - 1) / this.Size)) //O(1)
                                + Math.Abs(col - ((value - 1) % this.Size));
                    }
                }
            }

            this.Manhattan_Sum = count; //O(1)
        }
        //*************************************************************************
        public bool Hamming_Rech_Goal()//O(1)
        {
            return this.Hamming_Sum == 0; //O(1)
        }
        //*************************************************************************
        public bool Manhattan_rech_goal() //O(1)
        {
            return this.Manhattan_Sum == 0; //O(1)
        }
        //*************************************************************************
    }
}
```

Bfs.CS :

```
using System;
using System.Collections.Generic;
```

```
using System.Diagnostics;

using System.IO;
using System.Linq;

namespace Puzzel
{
    class Bfs
    {
        private int row, column; //O(1)
        private int[,] board; //O(1)
        public List<Bfs> Main_List = new List<Bfs>(); //O(1)
        public List<Bfs> Based_List = new List<Bfs>(); //O(1)
        private int Value; //O(1)
        private List<Bfs> Neighboors; //O(1)
        private Bfs Parent; //O(1)
        private static FileStream file; //O(1)
        private static StreamReader str; //O(1)
        private static Dictionary<int, List<string>> Line; //O(1)
        private Bfs(int size, int val, int[,] brd)
        {
            this.Neighboors = new List<Bfs>(); //O(1)
            this.board = new int[size, size]; //O(1)
            Array.Copy(brd, this.board, size * size); //O(N^2)
            this.Value = val; //O(1)
        }
        public Bfs()
        {
            this.Neighboors = new List<Bfs>(); //O(1)

            this.Value = 0; //O(1)
        }

        public void Get_Chileds(int size)//O(N²)
        {
            for (int i = 0; i < 4; i++) //O(N^2)
            {
                // check for availabilty of move
                if (Can_Solve(this.row, this.column, i, size))//O(1)
                {
                    var ind = Tuple.Create(this.row, this.column); //O(1)
                    if (i == 0) //O(1)
                    {
                        ind = Tuple.Create(this.row + 1, this.column); //O(1)
                    }
                    else if (i == 1) //O(1)
                    {
                        ind = Tuple.Create(this.row - 1, this.column); //O(1)
                    }
                    else if (i == 2) //O(1)
                    {
                        ind = Tuple.Create(this.row, this.column + 1); //O(1)
                    }
                    else if (i == 3) //O(1)
                    {
                        ind = Tuple.Create(this.row, this.column - 1); //O(1)
                    }
```

```
                    Bfs child = new Bfs(size, board[ind.Item1, ind.Item2],
this.board);//O(N^2)
                    Array.Copy(this.board, child.board, size * size);//O(N²)
                    int n = child.board[this.row, this.column]; //O(1)
                    child.board[this.row, this.column] = child.board[ind.Item1,
ind.Item2]; //O(1)
                    child.board[ind.Item1, ind.Item2] = n; //O(1)
                    child.row = ind.Item1; //O(1)
                    child.column = ind.Item2; //O(1)
                    child.Parent = this; //O(1)
                    var temp = this.Parent; //O(1)
                    if (!(temp != null && temp.column == child.column && temp.row ==
child.row)) //O(1)
                    {
                        this.Neighboors.Add(child);//O(1)
                    }
                }
            }
        }
        public bool found(Bfs child) //O(1)
        {
            return Main_List.Contains(child) && !Based_List.Contains(child); //O(1)
        }
        public Bfs BFS(Bfs start, int size, int[,] goalboard) //O(N^2)
        {
            Main_List.Add(start); //O(1)
            while (Main_List.Count != 0)//O(N^2)
            {
                Bfs current = Main_List[0]; //O(1)
                current.Get_Chileds(size);//O(N²)
                foreach (var child in current.Neighboors)//O(1)
                {
                    if (Check_Reach_Goal(child.board, goalboard, size))//O(N²)
                    {
                        return child; //O(1)
                    }
                    if (!found(child)) //O(N)
                    {
                        child.Parent = current; //O(1)
                        Main_List.Add(child);//O(1)
                    }
                }
                int index = 0; //O(1)
                Main_List.RemoveAt(index); //O(1)
                Based_List.Add(current); //O(1)
            }
            return null;
        }
        bool Check_Reach_Goal(int[,] first, int[,] second, int size)//O(N²)
        {
            int count = 0; //O(1)
            for (int i = 0; i < size; i++) //O(N^2)
            {
                for (int j = 0; j < size; j++) //O(N)
                {
                    if (first[i, j] != second[i, j]) //O(1)
                    {
```

```
                count++; //O(1)
                break; //O(1)
            }
        }

    }
    return count == 0; //O(1)
}



private static bool Can_Solve(int x, int y, int i, int size)//O(1)
{
    if (i == 0) //O(1)
    {
        if (x + 1 >= size) //O(1)
        {
            return false; //O(1)
        }
        else
        {
            return true; //O(1)
        }
    }
    else if (i == 1) //O(1)
    {
        if (x - 1 < 0) //O(1)
        {
            return false; //O(1)
        }
        else
        {
            return true; //O(1)
        }
    }
    else if (i == 2) //O(1)
    {
        if (y + 1 >= size) //O(1)
        {
            return false; //O(1)
        }
        else
        {
            return true; //O(1)
        }
    }
    else if (i == 3) //O(1)
    {
        if (y - 1 < 0) //O(1)
        {
            return false; //O(1)
        }
        else
        {
            return true; //O(1)
        }
    }
    return false; //O(1)
```

```
}
public static void readAndCalc(string t)
{
    file = new FileStream(t, FileMode.Open, FileAccess.Read); //O(1)
    str = new StreamReader(file); //O(1)
    string line = str.ReadLine(); //O(1)
    int size = int.Parse(line); //O(1)
    int[,] goal = new int[size, size]; //O(1)
    line = str.ReadLine(); //O(1)
    Line = new Dictionary<int, List<string>>(); //O(1)
    int ii = 0, jj = 0; //O(1)
    for (int i = 0; i < size;) //O(N^2)
    {
        if (line == "") //O(1)
        {
            line = str.ReadLine(); //O(1)
            i = 0; //O(1)
            continue;
        }
        Line.Add(i, new List<string>()); //O(1)
        List<string> vertices = line.Split(' ').ToList(); //O(1)
        Line[i] = vertices; //O(1)
        for (int j = 0; j < size; j++) //O(N)
        {
            int g = i * size + (j + 1); //O(1)
            goal[i, j] = g; //O(1)
            jj = j; //O(1)
        }
        ii = i; //O(1)
        i++; //O(1)
        line = str.ReadLine(); //O(1)
    }

    goal[ii, jj] = 0; //O(1)
    str.Close(); //O(1)
    file.Close(); //O(1)
    int[,] board = new int[size, size]; //O(1)
    Bfs Bfs_start = new Bfs(); //O(1)
    for (int i = 0; i < size; i++) //O(N)
    {
        foreach (var element in Line[i]) //O(1)
        {
            int j = Line[i].IndexOf(element); //O(1)
            board[i, j] = int.Parse(element); //O(1)
        }
    }
    Bfs[,] graph = new Bfs[size, size]; //O(1)
    Bfs firstnode = new Bfs(); //O(1)
    for (int i = 0; i < size; i++) //O(N^3)
    {
        foreach (var element in Line[i]) //O(N^2)
        {
            int j = Line[i].IndexOf(element); //O(1)
            graph[i, j] = new Bfs(size, int.Parse(element), board) //O(N^2)
            {
                Parent = null, //O(1)
                row = i, //O(1)
                column = j, //O(1)
```

```
                };
                board[i, j] = int.Parse(element); //O(1)
                if (element.Equals("0")) //O(1)
                {
                    firstnode = graph[i, j]; //O(1)
                }
            }
        }
        Stopwatch bfswatch = new Stopwatch(); //O(1)
        bfswatch.Start(); //O(1)
        Bfs Goal = firstnode.BFS(firstnode, size, goal); //O(N^2)
        Bfs_start = Goal; //O(1)
        bfswatch.Stop(); //O(1)

        int step = 0; //O(1)
        List<Bfs> nodes = new List<Bfs>(); //O(1)
        Console.WriteLine("Childs From Down To Top");
        while (Bfs_start != null) //O(N^2)
        {
            Console.WriteLine("Chiled ", step);
            step++; //O(1)
            for (int i = 0; i < size; i++) //O(N^2)
            {
                for (int j = 0; j < size; j++) //O(N)
                {
                    Console.Write(Bfs_start.board[i, j]); //O(1)
                    Console.Write(" ");
                }
                Console.WriteLine();
            }
            Console.WriteLine();
            Bfs_start = Bfs_start.Parent; //O(1)
        }
        Console.WriteLine("Time : " + bfswatch.Elapsed);
    }


}
}
```

Program.CS :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;
namespace Puzzel
{
    class Program
    {
```

```
        //*************************************************************************
        public static int moves;//To save the goal steps that i want to reach in the
final //O(1)
        static public bool check_solvable(int[] arr, int size, int rowofzero)//check if
the puzzle can solved or not //O(N^2)
        {

            int inv_Count = 0;//to count the number of ele that greatter than and next to
it
            size *= size;//to get the total size of the matrix
            for (int i = 0; i < size - 1; i++) //O(N^2)
            {
                for (int j = i + 1; j < size; j++) //O(N)
                {
                    if (arr[i] > arr[j] && arr[j] != 0)//check if the value is greatter
than any next of it //O(1)
                    {
                        inv_Count++; //O(1)
                    }
                }
            }
            int _ind = size - rowofzero + 2;//to show the index from down //O(1)
            if ((size % 2 != 0 && inv_Count % 2 == 0))//lw el size Odd w el count even
//O(1)
            {
                return true; //O(1)
            }
            else if ((size % 2 == 0 && inv_Count % 2 != 0 && _ind % 2 == 0)) //O(1)
            {//lw el size even w count off w el index mn down even
                return true; //O(1)
            }
            else if ((size % 2 == 0 && inv_Count % 2 == 0 && _ind % 2 != 0))//lw size
even w el count even w mn down odd //O(1)
            {
                return true; //O(1)
            }
            else return false; //O(1)

        }
        //*************************************************************************
        static int Rowofzero;//store the index of zero value in row //O(1)
        static int ColOfZero;//store the index of zero value in col //O(1)
        public static void Ali()
        {
            Console.WriteLine(">>>>> Member 1 : ");
            Console.WriteLine();
            Console.WriteLine("                    *          *          *         * *
* *       *       *   *    * * *     * * *");
            Console.WriteLine("                   * *         *          *
*         * *      *   *    *         *   *");
            Console.WriteLine("                  * * *        *          *
*        * * *     * * *    * * *     * * ");
            Console.WriteLine("                 *     *       *          *
*       *     *    *   *    *         *  *");
            Console.WriteLine("                *       *      * * * *    *
*      *       *   *   *    * * *     *   *");
            Console.WriteLine();
```

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

        }
        public static void mohammed()
        {
            Console.WriteLine(">>>>> Member 2 : ");
            Console.WriteLine();
            Console.WriteLine("                      *        *                      * * *
*        *        *           *       * * *  ");
            Console.WriteLine("                     * *      *  *                     *
*  *     * *         * *      *    * ");
            Console.WriteLine("                   *   *    *    *     * * *          * * *
*      *    *   *   *       * * *     *     *");
            Console.WriteLine("                     *     *  *      *    *   *          *
*      * *     *     *     *    *    * ");
            Console.WriteLine("                  *       *         *   * * *          * * *
*    *        *       *   *       *   * * *  ");
            Console.WriteLine();
            Console.WriteLine("------------------------------------------------------------
----------------------------------------");

        }
        public static void Aya()
        {
            Console.WriteLine(">>>>> Member 3 : ");
            Console.WriteLine();
            Console.WriteLine("                    *     *    *     *         *     *   *
*   * * *   * * *   * * *   *   *     *         ");
            Console.WriteLine("                     * *     *  *     * *        *     *   *
*   *       *       *       *   * *   *       ");
            Console.WriteLine("                    * * *     *      * * *       * * * *   *
*   * * *   * * *   * * *   *   *  *  *         ");
            Console.WriteLine("                   *     *    *     *     *      *     *   *
*       *       *   *       *   *   * *      ");
            Console.WriteLine("                *       *   *    *       *     *     *   *
* * *   * * *   * * *   * * *   *   *    *         ");
            Console.WriteLine();
            Console.WriteLine("------------------------------------------------------------
----------------------------------------");

        }
  public static void Main(string[] args)
        {
            // Read Board From File
            bool manhattan = false;//check if user choose the manhattan test O(1)
            Console.WriteLine("************************************    N Puzzle
**********************************");//O(1)
            string[] Tests = new string[]//store all tests name in it O(1)
            {
                "8 Puzzle (1).txt",
                "8 Puzzle (2).txt",
                "8 Puzzle (3).txt",
                "15 Puzzle - 1.txt",
                "24 Puzzle 1.txt",
                "24 Puzzle 2.txt",
                "8 Puzzle - Case 1.txt",
                "8 Puzzle(2) - Case 1.txt",
```

```
                "8 Puzzle(3) - Case 1.txt",
                "15 Puzzle - Case 2.txt",
                "15 Puzzle - Case 3.txt",
                "50 Puzzle.txt",
                "99 Puzzle - 1.txt",
                "99 Puzzle - 2.txt",
                "9999 Puzzle.txt",
                "man_15 Puzzle 1.txt",
                "man_15 Puzzle 3.txt",
                "man_15 Puzzle 4.txt",
                "man_15 Puzzle 5.txt",
                "15 Puzzle 1 - Unsolvable.txt",
                "99 Puzzle - Unsolvable Case 2.txt",
                "9999 Puzzle_not_solve.txt",
                "TEST.txt"
            };
            string target_test = "";//string to store the test name that the user choose
O(1)
            Console.WriteLine(">> 1- Sample Test cases "); //O(1)
            Console.WriteLine(">> 2- Complete Test cases "); //O(1)
            Console.WriteLine(">> 3- Team members "); //O(1)
            string ans = Console.ReadLine();//read the choose from the user O(1)
            if (ans == "1") //O(1)
            {
                Console.WriteLine("************************************  Sample Tests
**********************************");
                Console.WriteLine(">> 1- Solveable Tests ");
                Console.WriteLine(">> 2- Unsolveable Tests");
                Console.Write(">> Press 1 Or 2:  ");
                string ans1 = Console.ReadLine();//read the choose from the user O(1)
                if (ans1 == "1")
                {
                    Console.WriteLine("****************************************  Test
Cases  ******************************");

                    Console.WriteLine(">>  Test 1 ");
                    Console.WriteLine(">>  Test 2 ");
                    Console.WriteLine(">>  Test 3 ");
                    Console.WriteLine(">>  Test 4 ");
                    Console.WriteLine(">>  Test 5 ");
                    Console.WriteLine(">>  Test 6 ");
                    Console.Write(">> ");
                    string an = Console.ReadLine();//read the choose from the user O(1)
                    if (an == "1") //O(1)
                    {
                        target_test = Tests[0];//test1
                        moves = 8;
                    }
                    else if (an == "2") //O(1)
                    {
                        target_test = Tests[1];  //O(1)
                        moves = 20; //O(1)
                    }
                    else if (an == "3") //O(1)
                    {
                        target_test = Tests[2]; //O(1)
                        moves = 14; //O(1)
                    }
```

```
                else if (an == "4") //O(1)
                {
                    target_test = Tests[3]; //O(1)
                    moves = 5; //O(1)
                }
                else if (an == "5") //O(1)
                {
                    target_test = Tests[4]; //O(1)
                    moves = 11; //O(1)
                }
                else if (an == "6") //O(1)
                {
                    target_test = Tests[5]; //O(1)
                    moves = 24; //O(1)
                }
                else
                {
                    Console.WriteLine("Wrong Choice !!!"); //O(1)
                    Main(null); //O(1)
                }
            }
            else if (ans1 == "2")
            {
                Console.WriteLine("*****************************************
UnSolvable Cases ******************************");
                Console.WriteLine(">> Test 1 ");
                Console.WriteLine(">> Test 2 ");
                Console.WriteLine(">> Test 3 ");
                Console.WriteLine(">> Test 4 ");
                Console.WriteLine(">> Test 5 ");
                string an = Console.ReadLine(); //O(1)
                if (an == "1") //O(1)
                {
                    target_test = Tests[6]; //O(1)
                }
                else if (an == "2") //O(1)
                {
                    target_test = Tests[7]; //O(1)
                }
                else if (an == "3") //O(1)
                {
                    target_test = Tests[8]; //O(1)
                }
                else if (an == "4") //O(1)
                {
                    target_test = Tests[9]; //O(1)
                }
                else if (an == "5") //O(1)
                {
                    target_test = Tests[10]; //O(1)
                }
                else
                {
                    Console.WriteLine("Wrong Choice !!!"); //O(1)
                    Main(null); //O(1)
                }
            }
            else
```

```
                {
                    Console.WriteLine("Wrong Choice !!"); //O(1)
                    Main(null); //O(1)
                }

            }
            else if (ans == "2") //O(1)
            {
                Console.WriteLine("************************************  Complete
Tests  *******************************");
                Console.WriteLine("1- Solvable Tests");
                Console.WriteLine("2- Unsoveable Tests");
                Console.WriteLine("3- Very large Test");
                string ans2 = Console.ReadLine(); //O(1)
                if (ans2 == "1") //O(1)
                {
                    Console.WriteLine("************************************  Solvable
Tests  *******************************");
                    Console.WriteLine("1- Manhattan & Hamming");
                    Console.WriteLine("2- Manhattan Only");
                    string anss = Console.ReadLine(); //O(1)
                    if (anss == "1") //O(1)
                    {
                        Console.WriteLine("************************************
Manhattan & Hamming   *********************************");
                        Console.WriteLine(">> Test 1 ");
                        Console.WriteLine(">> Test 2 ");
                        Console.WriteLine(">> Test 3 ");
                        Console.WriteLine(">> Test 4 ");
                        string a = Console.ReadLine(); //O(1)
                        if (a == "1") //O(1)
                        {
                            target_test = Tests[11]; //O(1)
                            moves = 18; //O(1)
                        }
                        else if (ans == "2") //O(1)
                        {
                            target_test = Tests[12]; //O(1)
                            moves = 18; //O(1)
                        }
                        else if (ans == "3") //O(1)
                        {
                            target_test = Tests[13]; //O(1)
                            moves = 38; //O(1)
                        }
                        else if (ans == "4") //O(1)
                        {
                            target_test = Tests[14]; //O(1)
                            moves = 4; //O(1)
                        }
                        else
                        {
                            Console.WriteLine("Wrong Choice !"); //O(1)
                            Main(null); //O(1)
                        }
                    }
                    else if (anss == "2") //O(1)
                    {
```

```
                    manhattan = true;
                    Console.WriteLine("************************************
Manhattan Only   *********************************");
                    Console.WriteLine(">> Test 1 ");
                    Console.WriteLine(">> Test 2 ");
                    Console.WriteLine(">> Test 3 ");
                    Console.WriteLine(">> Test 4 ");
                    string a = Console.ReadLine(); //O(1)
                    if (a == "1") //O(1)
                    {
                        target_test = Tests[15]; //O(1)
                        moves = 46; //O(1)
                    }
                    else if (ans == "2") //O(1)
                    {
                        target_test = Tests[16]; //O(1)
                        moves = 38; //O(1)
                    }
                    else if (ans == "3") //O(1)
                    {
                        target_test = Tests[17]; //O(1)
                        moves = 44; //O(1)
                    }
                    else if (ans == "4") //O(1)
                    {
                        target_test = Tests[18]; //O(1)
                        moves = 45; //O(1)
                    }
                    else
                    {
                        Console.WriteLine("Wrong Choice !"); //O(1)
                        Main(null);
                    }
                }
                else
                {
                    Console.WriteLine("Wrong Choice !"); //O(1)
                    Main(null); //O(1)
                }
            }
  else if (ans2 == "2") //O(1)
            {
                Console.WriteLine("************************************
Unsolvable Cases   *********************************");
                Console.WriteLine(">> Test 1 ");
                Console.WriteLine(">> Test 2 ");
                Console.WriteLine(">> Test 3 ");
                string a = Console.ReadLine(); //O(1)
                if (a == "1") //O(1)
                {
                    target_test = Tests[19]; //O(1)
                }
                else if (ans == "2") //O(1)
                {
                    target_test = Tests[20]; //O(1)
                }
                else if (ans == "3") //O(1)
                {
```

```
                    target_test = Tests[21]; //O(1)
                }
                else
                {
                    Console.WriteLine("Wrong Choice !"); //O(1)
                    Main(null); //O(1)
                }
            }
            else if (ans2 == "3") //O(1)
            {
                Console.WriteLine("************************************  Final
Case   *********************************");
                target_test = Tests[22]; //O(1)
                moves = 56; //O(1)
            }
            else
            {
                Console.WriteLine(" Wrong Choice !");

                Main(null); //O(1)
            }

        }
        else if (ans == "3") //O(1)
        {
            Ali();
            mohammed();
            Aya();
            Main(null);
        }
        else
        {
            Console.WriteLine("Wrong Choice !");
            Main(null);
        }

Console.WriteLine("************************************************************************"
);
        //Read the target test
        FileStream fs = new FileStream(target_test, FileMode.Open, FileAccess.Read);
//O(1)
        StreamReader sr = new StreamReader(fs); //O(1)
        while (sr.Peek() != -1)//read line by line from the text file O(N^2)
        {
            String s = sr.ReadLine();//read the line //O(1)
            String[] fields;//store element by element except space //O(1)
            fields = s.Split(' '); //O(1)
            int N; //O(1)
            N = int.Parse(fields[0]);//store the size of the matrix //O(1)
            int val; //O(1)
            int[] arr1d = new int[N * N];//to store the node in 1 d array //O(1)
            int[,] Node2d = new int[N, N];//to store the node in 2d array //O(1)
            int c = 0; //O(1)
            for (int i = 0; i < N; i++) //O(N^2)
            {
                s = sr.ReadLine(); //O(1)
                fields = s.Split(' '); //O(1)
                for (int j = 0; j < N; j++) //O(N)
```

```
                    {
                        val = int.Parse(fields[j]); //O(1)
                        if (val == 0) //O(1)
                        {
                            Rowofzero = i;  //O(1)
                            ColOfZero = j;  //O(1)
                        }
                        Node2d[i, j] = val; //O(1)
                        arr1d[c++] = val; //O(1)
                    }
                }
                char ch; //O(1)
                if (!manhattan)//check if the user choose manhattan or not //O(N^2)
                {
                    //if he is not choose manhattan
                    Console.WriteLine(" - Press [1] To Using Hamming .");  //O(1)
                    Console.WriteLine(" - Press [2] To Using Manhattan ."); //O(1)
                    Console.WriteLine(" - Press [3] To using BFS ."); //O(1)
                    Console.Write(" - Enter Your Choice : "); //O(1)
                    ch = char.Parse(Console.ReadLine()); //O(1)
                }
                else
                {
                    // if user choose test manhattan only mack the choice 2
                    ch = '2'; //O(1)
                }


                if (ch == '1')
                {
                    if (check_solvable(arr1d, N, Rowofzero))//check the puzzle solved or
not //O(N^2)
                    {
                        Console.WriteLine("Solving .........");
                        Console.WriteLine();
                        puzzel start = new puzzel(N, Node2d, Rowofzero,
ColOfZero);//create the first node  //O(N^2)
                        AS A = new AS();//creat object from A star class
                        A.A_Star(start, "Hamming");//Choose the hamming algo to solve the
puzzle //O(E Log V)
                    }
                    else
                    {//if the puzzle can not solve
                        Console.WriteLine("No Feasible Solution For The Given Board ");
                        Console.WriteLine();
                        Main(null);//going to the start page and running again
                    }
                }
                else if (ch == '2') //O(1)
                {

                    if (check_solvable(arr1d, N, Rowofzero))//check the puzzle solved or
not //O(N^2)
                    {
                        Console.WriteLine("Solving .........");
                        Console.WriteLine();
                        puzzel start = new puzzel(N, Node2d, Rowofzero,
ColOfZero);//create the first node //O(N^2)
```

```
                        AS A = new AS(); //creat object from A star class //O(1)
                        A.A_Star(start, "manhattan");//Choose the hamming algo to solve
the Manhattan //O(E Log V)
                    }
                    else //O(1)
                    {
                        Console.WriteLine(" - No Feasible Solution For The Given Board
");
                        Console.WriteLine();
                        Main(null);
                    }

                }
                else if (ch == '3')//Bouns
Section!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //O(N^2)
                {
                    if (check_solvable(arr1d, N, Rowofzero)) //O(N^2)
                    {
                        Console.WriteLine("Solving .........");
                        Console.WriteLine();
                        Bfs.readAndCalc(target_test);//solving using BFS //O(N^3)
                        Main(null);
                    }

                    else
                    {
                        Console.WriteLine(" - No Feasible Solution For The Given Board
");
                        Console.WriteLine();
                        Main(null);
                    }

                }
            }
        }
    }
}
```

AS.CS :

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

namespace Puzzel
{
    class AS
    {
        private List<puzzel> Solution_Route;// To save every Step in the solution route
        private List<puzzel> Based_List;// To store every node that i deleted from the
main
        private PriorityQueue Main_List;// To store every in it to set the min hur in the
top and eleted it
```

```
        private Dictionary<string, puzzel> Main_Map;//to search fast
        private Dictionary<string, puzzel> Based_Map;//to search fast
        public AS()
        {//to intialize structures
            Based_List = new List<puzzel>();//O(1) Intialize The list that contain the
deleted nodes
            Solution_Route = new List<puzzel>();//O(1) Intialize the dict the declare the
status of each node
            Main_Map = new Dictionary<string, puzzel>();//O(1) Intialize the main dect
            Based_Map = new Dictionary<string, puzzel>();//O(1) intialize the base dict
            Main_List = new PriorityQueue();//O(1) intialize the pq
        }
        //*********************************************************
        public void A_Star(puzzel start, string kind)//Total complexity O(E*log(v)) num
of itr=E * Body Log(v)
        {
            Main_List.Enqueue(start);//O(Log(V)) Insert the first node in the priority
queue
            Main_Map.Add(start.key, start);//O(1) to insert the node key string in the
dictionary
            while (!Main_List.is_empty())//O(E))//to check that the queue empty or not
            {
                puzzel temp = Main_List.Dequeue();//O(1)  Delete the
                puzzel current = new puzzel(temp, 0);//O(1) Create the puzzle of child
                if (!check_in_Based(current))//O(1) Check if it found in
                {
                    Based_List.Add(current);//O(1) insert it in the based
                    Based_Map.Add(current.key, current);//O(1) insert it in the dict
                    Find_Neighboors(current, kind);//O(1) find the neigboors of the child
t
                }
            }
        }
        //*********************************************************

  public void Find_Neighboors(puzzel p, string kind)//O(1) To generate the child
        {
            if (p.Check("Left"))//O(1) Check if I can move it to Left
            {
                puzzel ch = new puzzel(p);//O(1) Create new node
                ch.Move("Left");//O(1) // Moving the node to left
                if (kind == "manhattan")//check if user Choose Manhattan
                {
                    ch.Generate_Man();//O(N^2) Generate the manhattan of the node
                    ch.Calc_Min_Cost_Man();//O(1) Calculate the Cost of the node
                    if (ch.Manhattan_rech_goal())//O(1) check if i Reach the Goal
                    {
                        ch.direction = "GOAL";//O(1) give it direction
                        // Add Goal Board
                        Solution_Route.Add(ch);//O(1)
                        Get_Route(ch);//O(1)
                    }
                }
                else
                {
                    ch.Generate_Hamming();//O(N^2)
                    ch.Calc_Min_Cost_Hamm();//O(1)
                    if (ch.Hamming_Rech_Goal())//O(1)
```

```
            {
                ch.direction = "Goal";//O(1)
                // Add Goal Board
                Solution_Route.Add(ch);//O(1)
                Get_Route(ch);//O(1)
            }
        }
        ch.direction = "Left";//O(1)

        //check if child in Main list or not
        bool flag;//O(1)
        flag = check_in_Main(ch);//O(1)
        if (flag == false)
        {//O(1)
            Main_List.Enqueue(ch);//O(Log(v)
            Main_Map.Add(ch.key, ch);//O(1)
        }

    }
    if (p.Check("Right"))//O(1)
    {
        puzzel c = new puzzel(p);//O(1)
        c.Move("Right");//O(1)
        if (kind == "manhattan")//O(1)
        {
            c.Generate_Man();//O(N^2)
            c.Calc_Min_Cost_Man();//O(1)
            if (c.Manhattan_rech_goal())//O(1)
            {
                c.direction = "Goal";//O(1)
                // Add Goal Board
                Solution_Route.Add(c);//O(1)
                Get_Route(c);//O(1)
            }
        }
        else
        {
            c.Generate_Hamming();//O(n^2)
            c.Calc_Min_Cost_Hamm();//O(1)
            if (c.Hamming_Rech_Goal())//O(1)
            {
                c.direction = "Goal";//O(1)
                // Add Goal Board
                Solution_Route.Add(c);//O(1)
                Get_Route(c);//O(1)
            }
        }
        c.direction = "Right";//O(1)
        //check if child in open list or not
        bool flag;//O(1)
        flag = check_in_Main(c);//O(1)
        if (flag == false)//O(1)
        {
            Main_List.Enqueue(c);////O(Log(V))
            Main_Map.Add(c.key, c);//O(1)
        }
    }
    if (p.Check("Up"))//O(1)
```

```
{
    puzzel c = new puzzel(p);//O(1)
    c.Move("Up");//O(1)
    if (kind == "manhattan")//O(1)
    {
        c.Generate_Man();//O(N^2)
        c.Calc_Min_Cost_Man();//O(1)
        if (c.Manhattan_rech_goal())//O(1)
        {
            c.direction = "Goal";//O(1)
            Solution_Route.Add(c);//O(1)
            Get_Route(c);//O(1)
        }
    }
    else
    {
        c.Generate_Hamming();//O(N^2)
        c.Calc_Min_Cost_Hamm();//O(1)
        if (c.Hamming_Rech_Goal())//O(1)
        {
            c.direction = "Goal";//O(1)
            Solution_Route.Add(c);//O(1)
            Get_Route(c);//O(1)
        }
    }
    c.direction = "Up";//O(1)
    bool flag;//O(1)
    flag = check_in_Main(c);//O(1)
    if (flag == false)
    {
        Main_List.Enqueue(c); //O(Log(V))
        Main_Map.Add(c.key, c);////O(1)
    }

}
if (p.Check("Down"))//O(1)
{
    puzzel c = new puzzel(p);//O(1)
    c.Move("Down");//O(1)
    if (kind == "manhattan")//O(1)
    {
        c.Generate_Man();//O(1)
        c.Calc_Min_Cost_Man();//O(1)
        if (c.Manhattan_rech_goal())//O(1)
        {
            c.direction = "Goal";//O(1)
            Solution_Route.Add(c);//O(1)
            Get_Route(c);//O(1)
        }
    }
    else
    {
        c.Generate_Hamming();//O(N^2)
        c.Calc_Min_Cost_Hamm();//O(1)
        if (c.Hamming_Rech_Goal())//O(1)
        {
            c.direction = "Goal";//O(1)
            Solution_Route.Add(c);//O(1)
```

```
                    Get_Route(c);//O(1)
                }
            }
            c.direction = "Down";//O(1)
            //check if child in open list or not
            bool flag;//O(1)
            flag = check_in_Main(c);//O(1)
            if (flag == false)
            {//O(1)
                Main_List.Enqueue(c);//O(log(V)
                Main_Map.Add(c.key, c);//O(1)
            }
        }
    }
    //*********************************************

public bool check_in_Based(puzzel c)//O(1) Check if it found in the based or not
    {
        if (Based_Map.ContainsKey(c.key))//O(1)
        {
            puzzel _key = Based_Map[c.key];//O(1)
            if (_key.cost < c.cost)
            {//O(1)
                Main_List.Enqueue(_key);//O(1)
                Main_Map.Add(_key.key, _key);//O(1)
            }
            return true;//O(1)
        }
        return false;//O(1)
    }
    //*********************************************
    public bool check_in_Main(puzzel c)
    {

        return Main_Map.ContainsKey(c.key);//O(1)
    }

    //*************************************************
    public void display_Solution_Route()//O(1)
    {
        int n = Solution_Route.Count();//O(1)
        int Steps = n - 1;//O(1)
        for (int i = Steps; i >= 0; i--)//O(1)
        {
            Solution_Route[i].display();
        }
        Console.WriteLine(" - Num of Moves = " + Steps);////O(1)
        Console.WriteLine();//O(1)
        if (Steps == Program.moves)//O(1)
        {
            Console.WriteLine("Congratulation !!!!!");//O(1)
            Console.WriteLine();//O(1)
        }
        else Console.WriteLine("Wrong answer Expected " + Steps + " Recived " +
Program.moves);//O(1)
        Program.Main(null);
    }
    //*************************************************
```

```
        public void Get_Route(puzzel goal)//O(1)
        {
            // Get Path
            puzzel p = goal.parent;//O(1)
            while (p.parent != null)//O(1)
            {

                Solution_Route.Add(p);//O(1)
                p = p.parent;//O(1)
            }
            // Add Start Board
            Solution_Route.Add(p);//O(1)
            // call Display Solution_Route
            display_Solution_Route();//O(1)
        }
        //*****************************************************
    }
}
```

|===|