```
//Created by Hongwei Cao
//Created Date: November 12, 2013

//ACM International Collegiate Programming Contest

//Problem A: The Agency

//Following in the footsteps of a number of flight searching startups you want to create the first     ↙
    interplanetary
//travel website. Your first problem is to quickly find the cheapest way to travel between two
//planets. You have an advantage over your competitors because you have realized that all the planets
//and the flights between them have a special structure. Each planet is represented by a string of N bits
//and there is a flight between two planets if their N-bit strings differ in exactly one position.
//The cost of a flight is the cost of landing on the destination planet. If the ith character in a planet’s
//string is a 1 then the ith tax must be paid to land. The cost of landing on a planet is the sum of the
//applicable taxes.
//Given the starting planet, ending planet, and cost of the ith tax compute the cheapest set of flights to
//get from the starting planet to the ending planet.]

//Input

//Input for each test case will consist of two lines. The first line will have N (1 ≤ N ≤ 1,000), the     ↙
    number
//of bits representing a planet; S, a string of N zeroes and ones representing the starting planet; and E,
//a string representing the ending planet in the same format. The second line will contain N integers the
//ith of which is the cost of the ith tax. All costs will be between 1 and 1,000,000.

//Output

//For each test case output one number, the minimum cost to get from the starting planet to the ending
//planet, using the format given below.

//Sample Input
//3 110 011
//3 1 2

//5 00000 11111
//1 2 3 4 5

//4 1111 1000
//100 1 1 1

//30 000000000000000000000000000000 111111111111111111111111111111
//1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30


//Sample Output
//Case 1: 4
//Case 2: 35
//Case 3: 106
//Case 4: 4960



using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace the_cheapest_cost
```

```
{
    public partial class lowestCostForm : Form
    {
        //Declare the lists which will hold specific bits as the name mentioned.
        List<int> theSameBits = new List<int>();
        List<int> oneToZeroBits = new List<int>();
        List<int> zeroToOneBits = new List<int>();
        List<int> allThePossibleCost = new List<int>();
        //Declare an array to hold the taxes of each bit.
        int[] intTaxOfEachBit;
        //Declare a variable to hold the number of bit in the code.
        int numberOfBits;

        public lowestCostForm()
        {
            //initialize the components.
            InitializeComponent();
            startingPlanetTextBox.Enabled = false;
            endingPlanetTextBox.Enabled = false;
            taxTextBox.Enabled = false;
        }

        private void clearButton_Click(object sender, EventArgs e)
        {
            //reset the texts to empty and focus on the number of bits text box.
            numberOfBitsTextBox.Text = string.Empty;
            startingPlanetTextBox.Text = string.Empty;
            endingPlanetTextBox.Text = string.Empty;
            taxTextBox.Text = string.Empty;
            lowestCostTextBox.Text = string.Empty;
            numberOfBitsTextBox.Focus();
        }

        private void exitButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void calculateButton_Click(object sender, EventArgs e)
        {
            int cost = 0;
            //check whether the user input's validation.
            if (validationOfInput())
            {
                //initialize the lists
                initializeThreeList();
                allThePossibleCost.Clear();
                //calculate the possible optimal cost.
                cost = countTheSamebits() + countZeroToOne() + countOneToZero();
                //add the cost to the possible cost list.
                allThePossibleCost.Add(cost);
                //Calculate the possibe lower cost, when the same ONE bits list have some bits with high ↙
    tax.
                whenTheSameBiteWithBigTax();
                //initialize the lists
                initializeThreeList();
                //Calculate the possible lower cost, when the same ONE bits list and the One to Zero list ↙
    have some bits with high tax.
                swithBetweenTheSameBitsAndOneToZeroBits();
                //Show the lowest cost.
                lowestCostTextBox.Text = allThePossibleCost.Last().ToString();
            }

        }
        private Boolean validationOfInput()
        {
            string codeOfPlanet;
            string taxString;
            Boolean validationPassed;
            validationPassed = true;
            //Make sure the starting planet code text box accept "0" or "1"
            if (int.TryParse(numberOfBitsTextBox.Text, out numberOfBits))
            {
                codeOfPlanet=startingPlanetTextBox.Text;
                if (codeOfPlanet.Length == numberOfBits)
                {
                    for (int i = 0; i < codeOfPlanet.Length; i++)
                    {
                        if (codeOfPlanet[i] != '1' && codeOfPlanet[i] != '0')
                        {
                            MessageBox.Show("Please input valid code in starting planet.");
                            validationPassed = false;
                        }

                    }
                }
                else
                    MessageBox.Show("The length of the code is not met the requirement for the starting    ↙
    planet.");
                //Make sure the ending planet code text box accept "0" or "1"
                codeOfPlanet = endingPlanetTextBox.Text;
                if (codeOfPlanet.Length == numberOfBits)
                {
                    for (int i = 0; i < codeOfPlanet.Length; i++)
                    {
                        if (codeOfPlanet[i] != '1' && codeOfPlanet[i] != '0')
                        {
                            MessageBox.Show("Please input valid code in ending planet.");
                            validationPassed = false;
                        }
                    }
                }
                else
                {
                    MessageBox.Show("The length of the code is not match the requirement for the ending    ↙
    planet.");
                    validationPassed = false;
                }
            //Validate the tax text box, and initialize the tax array.
                taxString = taxTextBox.Text.Trim();

                intTaxOfEachBit = new int[numberOfBits];

                char[] delim = { ' ' };
                string[] taxesOfEachBit = taxString.Split(delim);
                if (taxesOfEachBit.Length == numberOfBits)
                {
                    for (int i = 0; i < numberOfBits; i++)
                    {
                        if( !int.TryParse(taxesOfEachBit[i],out intTaxOfEachBit[i]))
                        {
                            MessageBox.Show("You input invalid tax. Please check it again.");
                            validationPassed = false;
                        }
                    }
                }else
                {
                    MessageBox.Show("You missed some tax(es) of planet.");
                    validationPassed = false;
                }
            }
            else {
                MessageBox.Show("Please input the lenght of the code.");
                validationPassed = false;
                }
            //Return the status of validation result.
            return validationPassed;
        }

        private void initializeThreeList()
        {
            theSameBits.Clear();
            oneToZeroBits.Clear();
            zeroToOneBits.Clear();

            string codeOfStartingPlanet;
            string codeOfEndingPlanet;

            codeOfStartingPlanet = startingPlanetTextBox.Text;
            codeOfEndingPlanet = endingPlanetTextBox.Text;


            for (int i = 0; i < numberOfBits; i++)
            {
                if (codeOfStartingPlanet[i] == codeOfEndingPlanet[i])
                {
                    if (codeOfStartingPlanet[i] == '1')
                    {

                        theSameBits.Add(intTaxOfEachBit[i]);
                    }
                }
                else if (codeOfStartingPlanet[i] == '1')
                {
                    oneToZeroBits.Add(intTaxOfEachBit[i]);
                }
                else
                    zeroToOneBits.Add(intTaxOfEachBit[i]);
            }

            oneToZeroBits.Sort();
            zeroToOneBits.Sort();
            theSameBits.Sort();
        }
        private void whenTheSameBiteWithBigTax()
        {
            int total;
            int subtotal;
            subtotal = 0;

            for (int i = 0; i < theSameBits.Count; i++)
            {
                total = 0;

                zeroToOneBits.Add(theSameBits.Last());
                theSameBits.RemoveAt(theSameBits.Count - 1);
                subtotal += countAllOneBitTax();

                zeroToOneBits.Sort();
                total += subtotal + countTheSamebits() + countZeroToOne() + countOneToZero();

                if (total > allThePossibleCost.Last())
                {
                    break;
                }
                else
                {
                    allThePossibleCost.Add(total);
                    total = 0;
                }
            }

        }
        private void swithBetweenTheSameBitsAndOneToZeroBits()
        {
            int total;
            int subtotal;
            int subtotalInner;

            for (int index = 0; index < oneToZeroBits.Count; index++)
            {

                subtotalInner = 0;
                subtotal = 0;
                total = 0;
                initializeThreeList();
                for (int j = 0; j <= index; j++)
                {
                    oneToZeroBits.RemoveAt(oneToZeroBits.Count - 1 - j);
                    subtotal += countAllOneBitTax();
                }


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

                    zeroToOneBits.Add(theSameBits.Last());
                    theSameBits.RemoveAt(theSameBits.Count - 1);
                    zeroToOneBits.Sort();
                    subtotalInner += countAllOneBitTax();

                    total += subtotal + subtotalInner + countTheSamebits() + countZeroToOne() +     ↙
    countOneToZero();

                    if (total > allThePossibleCost.Last())
                        break;
                    else
                        //The last element of the list "allThePossibleCost" will hold the currently lowest ↙
    cost.
                        allThePossibleCost.Add(total);
                 }

            }


        }

        private int countAllOneBitTax()
        {
            int total = 0;
            for (int i = 0; i < theSameBits.Count; i++)
            {
                total += theSameBits[i];
            }
            for (int i = 0; i < oneToZeroBits.Count; i++)
            {
                total += oneToZeroBits[i];
            }
            return total;

        }
        private int countTheSamebits()
        {
            int total = 0;
            for (int i = 0; i < theSameBits.Count; i++)
            {
                total += theSameBits[i] * (oneToZeroBits.Count + zeroToOneBits.Count);
            }
            return total;
        }
        private int countOneToZero()
        {
            int total = 0;
            for (int i = 1; i < oneToZeroBits.Count; i++)
            {
                total += oneToZeroBits[oneToZeroBits.Count - 1 - i] * i;
            }
            return total;
        }
        private int countZeroToOne()
        {
            int total = 0;
            for (int i = 0; i < zeroToOneBits.Count; i++)
            {
                total += zeroToOneBits[i] * (zeroToOneBits.Count - i);
            }
            return total;
        }

        private void numberOfBitsTextBox_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!(char.IsNumber(e.KeyChar)) && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
            else {
                startingPlanetTextBox.Enabled = true;
                endingPlanetTextBox.Enabled = true;
                taxTextBox.Enabled = true;
            }
        }

        private void startingPlanetTextBox_KeyPress(object sender, KeyPressEventArgs e)
        {
            int length;
            length = 0;
            if(e.KeyChar != (char)Keys.D0 && e.KeyChar!= (char)Keys.D1 && e.KeyChar != (char)Keys.Back)
            {
                e.Handled=true;
            }
            else if (numberOfBitsTextBox.Text != "")
            {
                if (int.TryParse(numberOfBitsTextBox.Text, out length) && startingPlanetTextBox.Text.Length ↙
     >= length && e.KeyChar != (char)Keys.Back)
                {
                    e.Handled = true;
                    MessageBox.Show("the code length is " + length);
                }
            }
            else {
                MessageBox.Show("You haven't input the length of the code.");
                e.Handled = true;
            }

        }

        private void endingPlanetTextBox_KeyPress(object sender, KeyPressEventArgs e)
        {
            int length;
            length = 0;
            if (e.KeyChar != (char)Keys.D0 && e.KeyChar != (char)Keys.D1 && e.KeyChar != (char)Keys.Back)
            {
                e.Handled = true;
            }
            else if (numberOfBitsTextBox.Text != "")
            {
                if (int.TryParse(numberOfBitsTextBox.Text, out length) && endingPlanetTextBox.Text.Length >↙
    = length && e.KeyChar != (char)Keys.Back)
                {
                    e.Handled = true;
                    MessageBox.Show("the code length is " + length);
                }
            }
            else
            {
                MessageBox.Show("You haven't input the length of the code.");
                e.Handled = true;
            }
        }

        private void taxTextBox_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!(char.IsNumber(e.KeyChar)) && e.KeyChar != (char)Keys.Space && e.KeyChar != (char)Keys.   ↙
    Back)
            {
                e.Handled = true;
            }
        }
    }
}
```