// Course: CS6423

// Student name: Vinh Nguyen

// Student ID: 000200899

// Assignment #: #4

// Due Date: 11/9/2013

// Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

// (The signature means that the program is your own work)

// Score: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.Collections;

namespace HuffmanCode

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<Huffman\_Node> nodes = new List<Huffman\_Node>();

private Huffman\_Node Root = new Huffman\_Node();

// A string shows order of multiplication matrixes

public string str;

// An array to be used to store name of nodes

public string[] strOutputBST = new string[100];

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

{

// Reset all value output = ""

nodes.Clear();

if (richTextBox1.Text.ToString() != "")

{

richTextBox1.Text = "";

//richTextBox2.Text = "";

textBox2.Text = "";

textBox3.Text = "";

str = "";

}

// Process an input string to generate symbols and probabilities

if (textBox5.Text != "")

{

textBox1.Text = "";

textBox4.Text = "";

// read an input string

string strInput = textBox5.Text.ToString();

// Process to generate distinct symbols

ArrayList arrSym = new ArrayList();

char temp0 = strInput[0];

arrSym.Add(strInput[0]);

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

{

bool flag = true;

for (int j = 0; j < arrSym.Count; j++)

{

if (strInput[i] == (char)arrSym[j])

flag = false;

}

if (flag == true)

arrSym.Add(strInput[i]);

}

for (int j = 0; j < arrSym.Count; j++)

{

textBox4.Text += arrSym[j];

if (j < arrSym.Count - 1)

textBox4.Text += " ";

}

// Process to calculate probability of symbols

for (int j = 0; j < arrSym.Count; j++)

{

double count = 0.0;

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

{

if (strInput[i] == (char)arrSym[j])

count++;

}

textBox1.Text += Math.Round(count / strInput.Length,3);

if (j < arrSym.Count - 1)

textBox1.Text += " ";

}

}

// Read an input string that contains name of node

string input1 = textBox4.Text.ToString();

// Split the input string that contains name of node by white space

string[] arr1 = input1.Split(' ');

// a variable is assigned to store length of input

int n = arr1.Length;

// An array stores all name of nodes

for (int i = 0; i < n; i++)

{

strOutputBST[i + 1] = arr1[i];

}

// Read an input string that contains probabilities of nodes

string input = textBox1.Text.ToString();

// Split the input string that contains probabilities of nodes by white space

string[] arr = input.Split(' ');

// An array stores all probabilities of nodes

double[] p = new double[n];

for (int i = 0; i < n; i++)

{

p[i] = Double.Parse(arr[i]);

}

for (int i = 0; i < n; i++)

{

Huffman\_Node ptr = new Huffman\_Node(arr1[i], p[i]);

nodes.Add(ptr);

}

// Process to sort list by probability

List<Huffman\_Node> orderedDescending = nodes.OrderByDescending(node => node.probability).ToList<Huffman\_Node>();

//a1 a2 a3 a4 a5 a6

//a b c d e f

//0.1 0.4 0.06 0.1 0.04 0.3

//Process to output symbols and probabilities by orderedDescending

foreach (Huffman\_Node node in orderedDescending)

{

textBox2.Text += node.symbol + "\t";

textBox3.Text += node.probability + "\t";

//MessageBox.Show("symbol = " + node.symbol + "prob = " + node.probability);

}

// Huffman algorithm

while (nodes.Count > 1)

{

// Proccess to print step by step probabilities

List<Huffman\_Node> orderedTemp = nodes.OrderByDescending(node => node.probability).ToList<Huffman\_Node>();

foreach (Huffman\_Node node in orderedTemp)

{

richTextBox1.Text += node.probability +"\t";

}

richTextBox1.Text += "\n";

//Sort list by orderedAscending

List<Huffman\_Node> orderedNodes = nodes.OrderBy(node => node.probability).ToList<Huffman\_Node>();

if (orderedNodes.Count >= 2)

{

// Take first two items to construct new parent

List<Huffman\_Node> taken = orderedNodes.Take(2).ToList<Huffman\_Node>();

// Create a parent node by combining the frequencies

Huffman\_Node parent = new Huffman\_Node();

parent.probability = taken[0].probability + taken[1].probability;

taken[0].data = "1";

taken[1].data = "0";

parent.Left = taken[0];

parent.Right = taken[1];

nodes.Remove(taken[0]);

nodes.Remove(taken[1]);

nodes.Add(parent);

}

this.Root = nodes.FirstOrDefault();

}

string res = "";

for (int i = 0; i < n; i++)

{

Root.pathToNode(Root, arr1[i], res);

}

}

}

}

Output Screenshot:



    