// Course: CS6423

// Student name: Vinh Nguyen

// Student ID: 000200899

// Assignment #: #3

// Due Date: 10/29/2013

// 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;

namespace OptimalBST

{

public partial class Form1 : Form

{

// An array 2 dimensions contains cost of matrixes

public double[,] m = new double[100, 100];

// An array 2 dimensions contains probabilityies of key nodes

public double[,] prob = new double[100, 100];

// An array 2 dimensions contains k value

public int[,] s = new int[100, 100];

// 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];

public Form1()

{

InitializeComponent();

}

//A function get string of subtrees

private string construct\_optimal\_subtree(int i, int j, int r1, string subtree)

{

int t;

// a temporary string to get string of subtrees

string str1 = "";

if (i <= j)

{

t = s[i,j];

str1 += strOutputBST[t] + " is " + subtree + " subtree of " + strOutputBST[r1] + "\n";

str1 += construct\_optimal\_subtree(i, t - 1, t, "left");

str1 += construct\_optimal\_subtree(t + 1, j, t, "right");

}

return str1;

}

// A function to contructs binary tree. get root string

private string construct\_BST(int n)

{

int r1;

r1 = s[1,n];

str += strOutputBST[r1] + " is the root \n";

str += construct\_optimal\_subtree(1, r1 - 1, r1, "left");

str += construct\_optimal\_subtree(r1 + 1, n, r1, "right");

return str;

}

// Optimal Binary Search Binary Tree Algorithm

private void optimalBST(double[] p, int n)

{

int j;

double q;

for (int i = 1; i <= n+1 ; i++)

{

m[i, i - 1] = 0;

prob[i, i - 1] = 0;

}

for (int l = 1; l <= n ; l++)

{

// richtextbox1 is used to show cost table step by step

richTextBox1.Text += "Step"+l+":\n";

// richtextbox1 is used to show k table step by step

richTextBox2.Text += "Step" + l + ":\n";

for (int i = 1; i <= n - l + 1; i++)

{

j = i + l - 1;

m[i, j] = Math.Pow(10, 5);

prob[i, j] = prob[i, j - 1] + p[j];

for (int k = i; k <= j; k++)

{

q = m[i, k - 1] + m[k + 1, j] + prob[i, j];

if (q < m[i, j])

{

m[i, j] = q;

s[i, j] = k;

}

}

}

// Show the cost table

for (int i = 1; i <= n ; i++)

{

for (int k = 1; k <= n ; k++)

{

richTextBox1.Text += m[i, k] + " \t";

richTextBox1.Text += " ";

}

richTextBox1.Text += "\n";

}

richTextBox1.Text += "\n";

// Show the k table

for (int i = 1; i <= n ; i++)

{

for (int k = 1; k <= n ; k++)

{

richTextBox2.Text += s[i, k] + " \t";

richTextBox2.Text += " ";

}

richTextBox2.Text += "\n";

}

richTextBox2.Text += "\n";

}

richTextBox1.Text += "\n\n";

richTextBox2.Text += "\n\n";

}

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

{

// Reset all value = ""

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

{

richTextBox1.Text = "";

richTextBox2.Text = "";

richTextBox3.Text = "";

richTextBox4.Text = "";

richTextBox5.Text = "";

textBox5.Text = "";

textBox6.Text = "";

str = "";

}

// 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 = 0;

for (int i = 0; i < arr1.Length; i++)

{

if (arr1[i] != "")

n++;

}

// 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(' ');

// a variable is assigned to store length of input

n = 0;

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

{

if (arr[i] != "")

n++;

}

// An array stores all probabilities of nodes

double[] p = new double[n+1];

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

{

p[i+1] = Double.Parse(arr[i]);

}

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

{

textBox5.Text += i + "\t";

textBox6.Text += i + "\t";

richTextBox3.Text += i + 1 + "\n";

richTextBox4.Text += i + 1 + "\n";

}

//Call optimal binary search tree function

optimalBST(p, n);

// Show binary search tree

string strOrder = construct\_BST(n);

richTextBox5.Text = strOrder;

// Show cost of binary search tree

textBox3.Text = m[1, n].ToString();

}

}

}

Screenshot:

