2018-CS-123

2018-CS-121

CS 311 Analysis of Algorithm

File Compression project

GUI of File Compression

In this document we will discuss about GUI of file compression we have different forms and in these different forms we have buttons and every button shows their unique functionality.

First form

In first form we have two buttons:

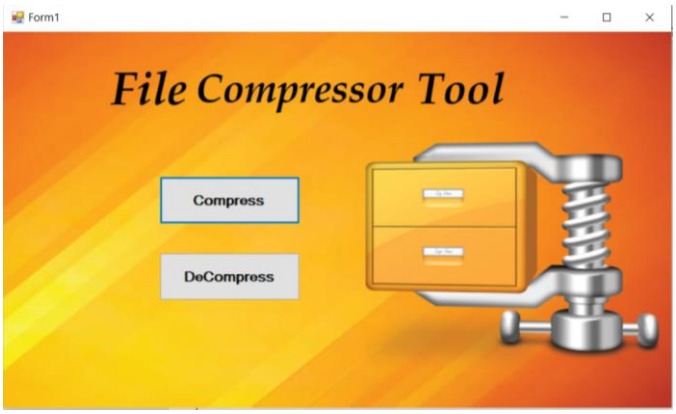
1) Compress
2) DeCompress

In first form1,the code behind on compress button:

```
var Form1 = new form2();
            this.Hide();
            new form2().Show();
```

In first form1,the code behind on DeCompress button

```
var Form1 = new DeCompress();
            this.Hide();
            new DeCompress().Show();
```



2nd form

In our first form, when we click compress button we jump into 2nd form which named as **COMPRESS** it opens a interface in which we have select a file and then click on browse button so it shows a message that our file is compressed successfully and if we want to go back then we just click on back button it backs us to our previous form which named as **form1.**

Commands to click the buttons:

Compress :

```
string path = textBox1.Text;

        if (path != "" && path != "Choose File")
        {
           File_compression_GUI.Huffman.Start(path);
           MessageBox.Show("Compression successful");
        }
        else
        {
           MessageBox.Show("Enter File paths");
        }
```

Browse:

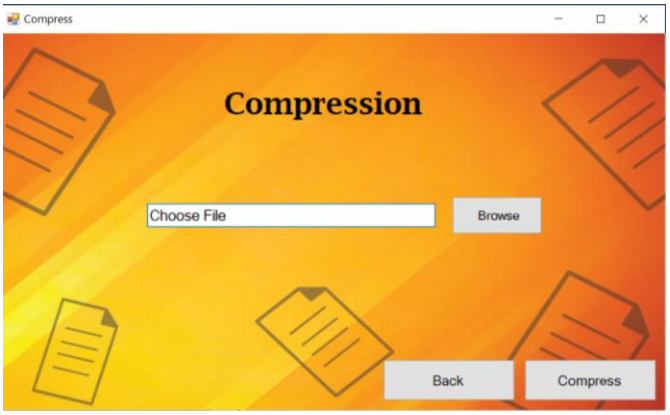
```
if(ofd.ShowDialog() == DialogResult.OK)
        {
           textBox1.Text = ofd.FileName;
        }
```

Textbox:

```
string path = textBox1.Text;
```

Back

```
var Form2 = new form2();
            this.Hide();
            new Form1().Show();
```



3rd form

Our 3rd form is about DeCompress

Again In 1st form , when we click DeCompress button we jump into 3rd form which named as **Decompress form** it opens a interface in which we have two files coded scheme file and compressed file, then click on browse button so it shows a message that our file is DeCompressed successfully and if we want to go back then we just click on back button it backs us to our previous form which named as **form1.**

Commands:

1) When we choose compressed file and then Browse it:

```
if (ofd.ShowDialog() == DialogResult.OK)
        {
           textBox1.Text = ofd.FileName;
        }
```

2) When we choose coded scheme file and then Browse it:

```
textBox3.Text = ofd.FileName;
```

DeCompress:

```
        string path1 = textBox1.Text;
        string path2 = textBox3.Text;

        if(path1 != "" && path2 != "" && path1 != "Choose Compressed File" && path2 != "Choose
Coded Schemme File")
        {
           MessageBox.Show("Decompression successful");
           File_compression_GUI.Huffman.Decoder(path1, path2);
        }
        else
```

```
        {
            MessageBox.Show("Enter File paths");
        }
```

Back

```
var Form2 = new form2();
            this.Hide();
            new Form1().Show();
```

Code for Compression and Decompression:

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


namespace File_compression_GUI
{
    public class Node
    {
        public char character { get; set; }
```

```
    public int frequency { get; set; }
    public Node left;
    public Node right;

}
// Class BST
public class BST
{
    public Node root;
    public BST()
    {
        root = null;
        root.left = null;
        root.right = null;
    }
}
class Huffman
{
    static Node head_addr;
    public static int l;


    //Huffman Tree, this functions builds tree and then genrates coded scheme file
    public static void Huffman_Tree(int[] freq, char[] ch, int length)
    {

        int sum = 0;
        Node s, tptr, pptr;
        Node head = new Node();
        head.left = null; head.right = null;
```

```
pptr = s = head;
head.frequency = sum;
head.character = '\0';

for (int j = 0; j < length; j++)
{
   Node new_node = new Node();
   new_node.left = null; new_node.right = null;
   tptr = new_node;
   tptr.frequency = freq[j];
   tptr.character = ch[j];
   sum = sum + freq[j];
   if (length == 1)            /* Added condition */
   {
      Node new2 = new Node();
      new2.frequency = sum;
      new2.character = '\0';
      new2.right = tptr;
      new2.left = null;
      s = new2;
      head = s;
   }
   else
   {
      if (s.left == null || s.right == null)
      {
         if (tptr.frequency > s.frequency && s.right == null)
         {
            s.frequency = sum;
            s.right = tptr;
```

```
            head = s;
        }
        else if (s.left == null)
        {
            s.frequency = sum;
            s.left = tptr;
            head = s;
        }

    }
    else
    {
        Node new_node_2 = new Node();
        new_node_2.right = null; new_node_2.left = null;
        new_node_2.frequency = sum; new_node_2.character = '\0';
        s = new_node_2;
        if (pptr.frequency < s.frequency)
        {
            s.left = tptr;
            s.right = pptr;
            pptr = s;
            head = s;
        }
        else
        {
            s.right = tptr;
            // head = s;
        }
    }
}
```

```
    }
    head_addr = head;
    l = length;
}
public static void Code_generator(int length, char[] ch, string path, Node head)
{
    //this part of code is to traverse left side of tree from head
    File.WriteAllText("coded_scheme_file.txt", "");
    int i = 0;
    string[] temp = new string[length];
    int[] array = new int[length * 5];
    for (int m = 0; m < length * 5; m++)
    {
        array[m] = -1;
    }
    File.WriteAllText("coded_scheme_file.txt", "");
    Node t, p;
    if (head == null || (head.left == null && head.right == null))
    {
        Console.WriteLine("Tree is empty");
    }
    else
    {
        t = head;
        if (t.left != null)
        {
            t = t.left;
            array[i] = 0;
            while (t.left != null && t.right != null)
            {
```

```
    if (t.left != null)
    {
        i++;
        t = t.left;
        array[i] = 0;
    }
    else
    {
        i++;
        t = t.right;
        array[i] = 1;
    }

}
//write code to file
int k = 0;
string a = "";
while (array[k] != -1)
{
    a += Convert.ToString(array[k]);
    k++;
}
a += "`" + Convert.ToString(t.frequency) + t.character + "\n";
File.WriteAllText("coded_scheme_file.txt", a);

// this part of code is to traverse right side of tree from head
i = 0; t = head;
a = "";
if (t.right != null)
{
```

```
t = t.right;
array[i] = 1;
if (t.right == null && t.left == null)
{
   k = 0;
   while (array[k] != -1)
   {
      a += Convert.ToString(array[k]);
      k++;
   }
   a += "`" + Convert.ToString(t.frequency) + t.character + "\n";
   File.AppendAllText("coded_scheme_file.txt", a);
   a = "";
}
while (t.left != null && t.right != null)
{
   if (t.left.left == null && t.left.right == null)
   {
      p = t;
      p = p.left;
      i++;
      array[i] = 0;
      //copy character, freq, array to file
      k = 0;
      while (array[k] != -1)
      {
         a += Convert.ToString(array[k]);
         k++;
      }
      a += "`" + Convert.ToString(p.frequency) + p.character + "\n";
```

```
            File.AppendAllText("coded_scheme_file.txt", a);
            a = "";
            t = t.right;
            //i++;
            array[i] = 1;
        }
        else if (t.right.left == null && t.right.right == null)
        {
            i++;
            array[i] = 1;
            //copy code character and frequency to file
            k = 0;
            while (array[k] != -1)
            {
                a += Convert.ToString(array[k]);
                k++;
            }
            a += "`" + Convert.ToString(t.frequency) + t.character + "\n";
            t = t.left;
        }
    }
    if (t.left == null && t.right == null)
    {
        k = 0;
        while (array[k] != -1)
        {
            a += Convert.ToString(array[k]);
            k++;
        }
        a += "`" + Convert.ToString(t.frequency) + t.character + "\n";
```

```
            }
            File.AppendAllText("coded_scheme_file.txt", a);
            Convert_to_compressed(path);
        }
    }
    else if (t.left == null && t.right != null && t.right.right == null && t.right.left == null)
    {
        int k = 0; string a = "";
        t = t.right;
        array[i] = 1;
        while (array[k] != -1)
        {
            a += Convert.ToString(array[k]);
            k++;
        }
        a += "`" + Convert.ToString(t.frequency) + t.character + "\n";
        File.AppendAllText("coded_scheme_file.txt", a);
        Convert_to_compressed(path);
    }
  }
}
//this function is to convert text file to compressed file using coded scheme file
public static void Convert_to_compressed(string path)
{
  string text = "";
  File.WriteAllText("Compressed_text_file.txt", "");
  string lines1 = File.ReadAllText(path);
  string[] lines = File.ReadAllLines("coded_scheme_file.txt");
  int n = lines.Length;
  foreach (char ch in lines1)
```

```
{
    for (int i = 0; i < n; i++)
    {
        if (lines[i].Length == 1)
        {
            if (Char.Equals(lines[i], ch) == true)
            {
                int j = 0;
                while (lines[i][j] != '`')
                {
                    text += lines[i][j];
                    j++;
                }
            }

        }
        else if (lines[i].Length == 0)
        {
            if (ch == '\n')
            {
                int j = 0;
                while (lines[i - 1][j] != '`')
                {
                    text += lines[i - 1][j];
                    j++;
                }
            }
            else
                continue;
        }
```

```
            else if (Char.Equals(lines[i][lines[i].Length - 1], ch) == true)
            {
                int j = 0;
                while (lines[i][j] != '`')
                {
                    text += lines[i][j];
                    j++;
                }
            }
        }
    }
    File.WriteAllText("Compressed_text_file.txt", text);
}
public static void Build_decode_tree(string path1, string path2)
{

    string f = "";
    string[] read_lines = File.ReadAllLines(path2);
    int n = read_lines.Length; int index = 0;
    int[] arr_f = new int[n];
    char[] arr_c = new char[n];

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

        string data = read_lines[i];
        if (data.Length == 0)
            continue;
        else if (!(i + 1 == n) && read_lines[i + 1].Length == 0)
        {
```

```
        int k = data.Length - 1; string r = "";
        while (data[k] != '`')
            k--;
        r = data.Substring(k + 1);
        arr_f[index] = Convert.ToInt32(r);
        arr_c[index++] = '\n';
        arr_f[index] = Convert.ToInt32(r);
        arr_c[index++] = '\r';
        i += 2;
    }

    else
    {
        char ch = data[data.Length - 1];
        string frq = "";
        int counter = data.Length - 1;
        while (data[counter] != '`')
            counter--;
        frq = data.Substring(counter + 1, (data.Length - 1) - counter - 1);

        arr_f[index] = Convert.ToInt32(frq);
        arr_c[index++] = ch;
    }
}
Array.Reverse(arr_f);
Array.Reverse(arr_c);
Huffman_Tree(arr_f, arr_c, n);
File.WriteAllText("new_file.txt", "");
string r1 = File.ReadAllText(path1);
Node tptr = head_addr;
```

```
Node pptr = tptr;
char character;
if (head_addr != null)
{
   for (int i = 0; i < r1.Length - 1; i++)
   {
      char code_char = r1[i];
      if (code_char == '0')
      {
         tptr = tptr.left;
         character = tptr.character;
         File.AppendAllText("new_file.txt", Convert.ToString(character));
         tptr = head_addr;
      }
      else if (code_char == '1')
      {
         if (tptr.right.left == null && tptr.right.right == null)
         {
            tptr = tptr.right;
            character = tptr.character;
            string a = File.ReadAllText("new_file.txt");
            if (a.Length == 0)
               File.WriteAllText("new_file.txt", Convert.ToString(character));
            else
               File.AppendAllText("new_file.txt", Convert.ToString(character));

            tptr = head_addr;
         }
         else
         {
```

```
                tptr = tptr.right;
            }
        }

    }
  }

}
```

//this function sorts the frequency array in increasing order and character array as well with frequency array

```
public static void sort(int[] arr, char[] arr2, int n)
{

  for (int i = 1; i < n; ++i)
  {
    int key = arr[i];
    char c = arr2[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > key)
    {
      arr[j + 1] = arr[j];
      arr2[j + 1] = arr2[j];
      j = j - 1;
    }
    arr[j + 1] = key;
    arr2[j + 1] = c;
  }
}
```

//Main fumction

```
public static void Start(string path)
{

    int n = 256;
    int[] freq = new int[n];
    char[] ch = new char[n];
    string line;

    int ind = 0;

    line = File.ReadAllText(path);
    int index = -1;
    foreach (char c in line)
    {
        index = -1;
        for (int i = 0; i < 256; i++)
        {
            if (c == ch[i])
                index = i;

        }
        if (index >= 0)
        {
            freq[index] = freq[index] + 1;
            ch[index] = c;

        }
        else
        {
            freq[ind] = 1;
```

```
                ch[ind] = c;
                ind++;
            }

        }

        sort(freq, ch, ind);   //calling to sort array


        Huffman_Tree(freq, ch, ind); //calling huffman to buid tree and generate code.
        Code_generator(l, ch, path, head_addr);
       // Build_decode_tree();
        //Console.WriteLine("done");


        Console.Read();
    }
    public static void Decoder(string path1, string path2)
    {
        Build_decode_tree(path1, path2);
    }
  }
}
```