VARA HUFFMAN COMPRESSION

Jose Alberto Nieto EA5HVK

Huffman encoding is an algorithm for the lossless compression based on the frequency of occurrence of a symbol in the string that is being compressed. The Huffman algorithm is based on statistical coding, which means that the probability of a symbol has a direct bearing on the length of its representation. The more probable the occurrence of a symbol is, the shorter will be its bit-size representation. In any transmission, certain characters are used more than others. Using binary representation, the number of bits required to represent each character depends upon the number of characters that have to be represented. Using one bit we can represent two characters, i.e., 0 represents the first character and 1 represents the second character. Using two bits we can represent four characters, and so on.

Unlike ASCII code, which is a fixed-length code using seven bits per character, Huffman compression is a variable-length coding system that assigns smaller codes for more frequently used characters and larger codes for less frequently used characters in order to reduce the size of files being compressed and transferred.

For example, in a input string with the following data:

XXXXXXYYYYZZ

The frequency of “X” is 6, the frequency of “Y” is 4, and the frequency of “Z” is 2. If each character is represented using a fixed-length code of two bits, then the number of bits required to store this file would be 24, i.e., (2 x 6) + (2x 4) + (2x 2) = 24.

If the above data were compressed using Huffman compression, the more frequently occurring numbers would be represented by smaller bits, such as:

X by the code 0 (1 bit)
Y by the code 10 (2 bits)
Z by the code 11 (2 bits)

therefore the size of the file becomes 18, i.e., (1x 6) + (2 x 4) + (2 x 2) = 18.

In the above example, more frequently occurring characters are assigned smaller codes, resulting in a smaller number of bits in the final compressed file.

SOURCE CODE ROUTINES

```
Option Explicit

Private Const PROGRESS_CALCFREQUENCY = 7
Private Const PROGRESS_CALCCRC = 5
Private Const PROGRESS_ENCODING = 88
Private Const PROGRESS_DECODING = 89
Private Const PROGRESS_CHECKCRC = 11

Event Progress(Procent As Integer)

Private Type HUFFMANTREE
  ParentNode As Integer
  RightNode As Integer
  LeftNode As Integer
  Value As Integer
  Weight As Long
End Type

Private Type ByteArray
  Count As Byte
  Data() As Byte
End Type

Private Declare Sub CopyMem Lib "kernel32" Alias "RtlMoveMemory"
(Destination As Any, Source As Any, ByVal Length As Long)

'Create Huffman tree
Private Sub CreateTree(Nodes() As HUFFMANTREE, NodesCount As Long,
Char As Long, Bytes As ByteArray)
    Dim a As Integer, NodeIndex As Long

    NodeIndex = 0
    For a = 0 To (Bytes.Count - 1)
        If (Bytes.Data(a) = 0) Then
            If (Nodes(NodeIndex).LeftNode = -1) Then
                Nodes(NodeIndex).LeftNode = NodesCount
                Nodes(NodesCount).ParentNode = NodeIndex
                Nodes(NodesCount).LeftNode = -1
                Nodes(NodesCount).RightNode = -1
                Nodes(NodesCount).Value = -1
                NodesCount = NodesCount + 1
            End If
            NodeIndex = Nodes(NodeIndex).LeftNode
        ElseIf (Bytes.Data(a) = 1) Then
            If (Nodes(NodeIndex).RightNode = -1) Then
                Nodes(NodeIndex).RightNode = NodesCount
                Nodes(NodesCount).ParentNode = NodeIndex
                Nodes(NodesCount).LeftNode = -1
                Nodes(NodesCount).RightNode = -1
                Nodes(NodesCount).Value = -1
                NodesCount = NodesCount + 1
            End If
            NodeIndex = Nodes(NodeIndex).RightNode
        Else
            Stop
        End If
    Next
    Nodes(NodeIndex).Value = Char
```

```
End Sub


'Encode routine
Public Sub EncodeByte(ByteArray() As Byte, ByteLen As Long)
    Dim i As Long, j As Long, Char As Byte, BitPos As Byte, lNode1 As
Long
    Dim lNode2 As Long, lNodes As Long, lLength As Long, Count As
Integer
    Dim lWeight1 As Long, lWeight2 As Long, Result() As Byte,
ByteValue As Byte
    Dim ResultLen As Long, Bytes As ByteArray, NodesCount As Integer,
NewProgress As Integer
    Dim CurrProgress As Integer, BitValue(0 To 7) As Byte, CharCount(0
To 255) As Long
    Dim Nodes(0 To 511) As HUFFMANTREE, CharValue(0 To 255) As
ByteArray

    If (ByteLen = 0) Then
        ReDim Preserve ByteArray(0 To ByteLen + 3)
        If (ByteLen > 0) Then Call CopyMem(ByteArray(4), ByteArray(0),
ByteLen)
        ByteArray(0) = 72
        ByteArray(1) = 69
        ByteArray(2) = 48
        ByteArray(3) = 13
        Exit Sub
    End If

    ReDim Result(0 To 522)
    Result(0) = 72
    Result(1) = 69
    Result(2) = 51
    Result(3) = 13
    ResultLen = 4

    For i = 0 To (ByteLen - 1)
        CharCount(ByteArray(i)) = CharCount(ByteArray(i)) + 1
        If (i Mod 1000 = 0) Then
            NewProgress = i / ByteLen * PROGRESS_CALCFREQUENCY
            If (NewProgress <> CurrProgress) Then
                CurrProgress = NewProgress
                RaiseEvent Progress(CurrProgress)
            End If
        End If
    Next
    For i = 0 To 255
        If (CharCount(i) > 0) Then
            With Nodes(NodesCount)
                .Weight = CharCount(i)
                .Value = i
                .LeftNode = -1
                .RightNode = -1
                .ParentNode = -1
            End With
            NodesCount = NodesCount + 1
        End If
    Next

    For lNodes = NodesCount To 2 Step -1
        lNode1 = -1: lNode2 = -1
```

```
            For i = 0 To (NodesCount - 1)
                If (Nodes(i).ParentNode = -1) Then
                    If (lNode1 = -1) Then
                        lWeight1 = Nodes(i).Weight
                        lNode1 = i
                    ElseIf (lNode2 = -1) Then
                        lWeight2 = Nodes(i).Weight
                        lNode2 = i
                    ElseIf (Nodes(i).Weight < lWeight1) Then
                        If (Nodes(i).Weight < lWeight2) Then
                            If (lWeight1 < lWeight2) Then
                                lWeight2 = Nodes(i).Weight
                                lNode2 = i
                            Else
                                lWeight1 = Nodes(i).Weight
                                lNode1 = i
                            End If
                        Else
                            lWeight1 = Nodes(i).Weight
                            lNode1 = i
                        End If
                    ElseIf (Nodes(i).Weight < lWeight2) Then
                        lWeight2 = Nodes(i).Weight
                        lNode2 = i
                    End If
                End If
            Next

            With Nodes(NodesCount)
                .Weight = lWeight1 + lWeight2
                .LeftNode = lNode1
                .RightNode = lNode2
                .ParentNode = -1
                .Value = -1
            End With

            Nodes(lNode1).ParentNode = NodesCount
            Nodes(lNode2).ParentNode = NodesCount
            NodesCount = NodesCount + 1
        Next

        ReDim Bytes.Data(0 To 255)
        Call CreateBitSequences(Nodes(), NodesCount - 1, Bytes, CharValue)

        For i = 0 To 255
            If (CharCount(i) > 0) Then lLength = lLength +
    CharValue(i).Count * CharCount(i)
        Next
        lLength = IIf(lLength Mod 8 = 0, lLength \ 8, lLength \ 8 + 1)

        If ((lLength = 0) Or (lLength > ByteLen)) Then
            ReDim Preserve ByteArray(0 To ByteLen + 3)
            Call CopyMem(ByteArray(4), ByteArray(0), ByteLen)
            ByteArray(0) = 72
            ByteArray(1) = 69
            ByteArray(2) = 48
            ByteArray(3) = 13
            Exit Sub
        End If

        Char = 0
```

```
    For i = 0 To (ByteLen - 1)
        Char = Char Xor ByteArray(i)
        If (i Mod 10000 = 0) Then
            NewProgress = i / ByteLen * PROGRESS_CALCCRC +
PROGRESS_CALCFREQUENCY
            If (NewProgress <> CurrProgress) Then
                CurrProgress = NewProgress
                RaiseEvent Progress(CurrProgress)
            End If
        End If
    Next
    Result(ResultLen) = Char
    ResultLen = ResultLen + 1
    Call CopyMem(Result(ResultLen), ByteLen, 4)
    ResultLen = ResultLen + 4
    BitValue(0) = 2 ^ 0
    BitValue(1) = 2 ^ 1
    BitValue(2) = 2 ^ 2
    BitValue(3) = 2 ^ 3
    BitValue(4) = 2 ^ 4
    BitValue(5) = 2 ^ 5
    BitValue(6) = 2 ^ 6
    BitValue(7) = 2 ^ 7
    Count = 0
    For i = 0 To 255
        If (CharValue(i).Count > 0) Then Count = Count + 1
    Next
    Call CopyMem(Result(ResultLen), Count, 2)
    ResultLen = ResultLen + 2
    Count = 0
    For i = 0 To 255
        If (CharValue(i).Count > 0) Then
            Result(ResultLen) = i
            ResultLen = ResultLen + 1
            Result(ResultLen) = CharValue(i).Count
            ResultLen = ResultLen + 1
            Count = Count + 16 + CharValue(i).Count
        End If
    Next

    ReDim Preserve Result(0 To ResultLen + Count \ 8)

    BitPos = 0
    ByteValue = 0
    For i = 0 To 255
        With CharValue(i)
            If (.Count > 0) Then
                For j = 0 To (.Count - 1)
                    If (.Data(j)) Then ByteValue = ByteValue +
BitValue(BitPos)
                    BitPos = BitPos + 1
                    If (BitPos = 8) Then
                        Result(ResultLen) = ByteValue
                        ResultLen = ResultLen + 1
                        ByteValue = 0
                        BitPos = 0
                    End If
                Next
            End If
        End With
    Next
```

```
    If (BitPos > 0) Then
        Result(ResultLen) = ByteValue
        ResultLen = ResultLen + 1
    End If

    ReDim Preserve Result(0 To ResultLen - 1 + lLength)

    Char = 0
    BitPos = 0
    For i = 0 To (ByteLen - 1)
        With CharValue(ByteArray(i))
            For j = 0 To (.Count - 1)
                If (.Data(j) = 1) Then Char = Char + BitValue(BitPos)
                BitPos = BitPos + 1
                If (BitPos = 8) Then
                    Result(ResultLen) = Char
                    ResultLen = ResultLen + 1
                    BitPos = 0
                    Char = 0
                End If
            Next
        End With
        If (i Mod 10000 = 0) Then
            NewProgress = i / ByteLen * PROGRESS_ENCODING +
PROGRESS_CALCCRC + PROGRESS_CALCFREQUENCY
            If (NewProgress <> CurrProgress) Then
                CurrProgress = NewProgress
                RaiseEvent Progress(CurrProgress)
            End If
        End If
    Next

    If (BitPos > 0) Then
        Result(ResultLen) = Char
        ResultLen = ResultLen + 1
    End If
    ReDim ByteArray(0 To ResultLen - 1)
    Call CopyMem(ByteArray(0), Result(0), ResultLen)
    If (CurrProgress <> 100) Then RaiseEvent Progress(100)
End Sub
Public Function DecodeString(Text As String) As String
    Dim ByteArray() As Byte
    ByteArray() = StrConv(Text, vbFromUnicode)
    Call DecodeByte(ByteArray, Len(Text))
    DecodeString = StrConv(ByteArray(), vbUnicode)
End Function
Public Function EncodeString(Text As String) As String
    Dim ByteArray() As Byte
    ByteArray() = StrConv(Text, vbFromUnicode)
    Call EncodeByte(ByteArray, Len(Text))
    EncodeString = StrConv(ByteArray(), vbUnicode)
End Function


'Decode routine
Public Sub DecodeByte(ByteArray() As Byte, ByteLen As Long)
    Dim i As Long, j As Long, Pos As Long, Char As Byte, CurrPos As
Long
    Dim Count As Integer, CheckSum As Byte, Result() As Byte, BitPos
As Integer
```

```
    Dim NodeIndex As Long, ByteValue As Byte, ResultLen As Long,
NodesCount As Long
    Dim lResultLen As Long, NewProgress As Integer, CurrProgress As
Integer, BitValue(0 To 7) As Byte
    Dim Nodes(0 To 511) As HUFFMANTREE, CharValue(0 To 255) As
ByteArray

    If (ByteArray(0) <> 72) Or (ByteArray(1) <> 69) Or (ByteArray(3)
<> 13) Then
    ElseIf (ByteArray(2) = 48) Then
        Call CopyMem(ByteArray(0), ByteArray(4), ByteLen - 4)
        ReDim Preserve ByteArray(0 To ByteLen - 5)
        Exit Sub
    ElseIf (ByteArray(2) <> 51) Then
        Err.Raise vbObjectError, "HuffmanDecode()", "The data either
was not compressed with HE3 or is corrupt (identification string not
found)"
        Exit Sub
    End If

    CurrPos = 5
    CheckSum = ByteArray(CurrPos - 1)
    CurrPos = CurrPos + 1

    Call CopyMem(ResultLen, ByteArray(CurrPos - 1), 4)
    CurrPos = CurrPos + 4
    lResultLen = ResultLen
    If (ResultLen = 0) Then Exit Sub
    ReDim Result(0 To ResultLen - 1)
    Call CopyMem(Count, ByteArray(CurrPos - 1), 2)
    CurrPos = CurrPos + 2

    For i = 1 To Count
        With CharValue(ByteArray(CurrPos - 1))
            CurrPos = CurrPos + 1
            .Count = ByteArray(CurrPos - 1)
            CurrPos = CurrPos + 1
            ReDim .Data(0 To .Count - 1)
        End With
    Next

    BitValue(0) = 2 ^ 0
    BitValue(1) = 2 ^ 1
    BitValue(2) = 2 ^ 2
    BitValue(3) = 2 ^ 3
    BitValue(4) = 2 ^ 4
    BitValue(5) = 2 ^ 5
    BitValue(6) = 2 ^ 6
    BitValue(7) = 2 ^ 7

    ByteValue = ByteArray(CurrPos - 1)
    CurrPos = CurrPos + 1
    BitPos = 0

    For i = 0 To 255
        With CharValue(i)
            If (.Count > 0) Then
                For j = 0 To (.Count - 1)
                    If (ByteValue And BitValue(BitPos)) Then .Data(j)
= 1
                    BitPos = BitPos + 1
```

```
                    If (BitPos = 8) Then
                        ByteValue = ByteArray(CurrPos - 1)
                        CurrPos = CurrPos + 1
                        BitPos = 0
                    End If
                Next
            End If
        End With
    Next

    If (BitPos = 0) Then CurrPos = CurrPos - 1

    NodesCount = 1
    Nodes(0).LeftNode = -1
    Nodes(0).RightNode = -1
    Nodes(0).ParentNode = -1
    Nodes(0).Value = -1

    For i = 0 To 255
        Call CreateTree(Nodes(), NodesCount, i, CharValue(i))
    Next

    ResultLen = 0
    For CurrPos = CurrPos To ByteLen
        ByteValue = ByteArray(CurrPos - 1)
        For BitPos = 0 To 7
            If (ByteValue And BitValue(BitPos)) Then NodeIndex =
Nodes(NodeIndex).RightNode Else NodeIndex = Nodes(NodeIndex).LeftNode
            If (Nodes(NodeIndex).Value > -1) Then
                Result(ResultLen) = Nodes(NodeIndex).Value
                ResultLen = ResultLen + 1
                If (ResultLen = lResultLen) Then GoTo DecodeFinished
                NodeIndex = 0
            End If
        Next
        If (CurrPos Mod 10000 = 0) Then
            NewProgress = CurrPos / ByteLen * PROGRESS_DECODING
            If (NewProgress <> CurrProgress) Then
                CurrProgress = NewProgress
                RaiseEvent Progress(CurrProgress)
            End If
        End If
    Next

DecodeFinished:
    Char = 0
    For i = 0 To (ResultLen - 1)
        Char = Char Xor Result(i)
        If (i Mod 10000 = 0) Then
            NewProgress = i / ResultLen * PROGRESS_CHECKCRC +
PROGRESS_DECODING
            If (NewProgress <> CurrProgress) Then
                CurrProgress = NewProgress
                RaiseEvent Progress(CurrProgress)
            End If
        End If
    Next
    If (Char <> CheckSum) Then Err.Raise vbObjectError,
"clsHuffman.Decode()", "The data might be corrupted (checksum did not
match expected value)"
    ReDim ByteArray(0 To ResultLen - 1)
```

```
    Call CopyMem(ByteArray(0), Result(0), ResultLen)
    If (CurrProgress <> 100) Then RaiseEvent Progress(100)
End Sub

Private Sub CreateBitSequences(Nodes() As HUFFMANTREE, ByVal NodeIndex
As Integer, Bytes As ByteArray, CharValue() As ByteArray)
    Dim NewBytes As ByteArray
    If (Nodes(NodeIndex).Value > -1) Then
        CharValue(Nodes(NodeIndex).Value) = Bytes
        Exit Sub
    End If
    If (Nodes(NodeIndex).LeftNode > -1) Then
        NewBytes = Bytes
        NewBytes.Data(NewBytes.Count) = 0
        NewBytes.Count = NewBytes.Count + 1
        Call CreateBitSequences(Nodes(), Nodes(NodeIndex).LeftNode,
NewBytes, CharValue)
    End If
    If (Nodes(NodeIndex).RightNode > -1) Then
        NewBytes = Bytes
        NewBytes.Data(NewBytes.Count) = 1
        NewBytes.Count = NewBytes.Count + 1
        Call CreateBitSequences(Nodes(), Nodes(NodeIndex).RightNode,
NewBytes, CharValue)
    End If
End Sub
```