A case study in Test-Driven Development

The Huffman coding

M.Ganaj, L.Jubica

1

Agenda

- The Huffman code
- Red/Green/Refactor
- Implementation
- Statistics
- Conclusions

2

The Huffman code

- The encoding has 4 steps
  1. Count the characters' occurrences
  2. Build the Huffman tree
  3. Get the new binary representation for each character
  4. Represent the text with the new codes

3

The Huffman code (Step 1)

- Our plain text is "**test_driven_development**"

t	e	s	d	r	i	v	n	_	l	o	m	p
3	5	1	2	1	1	2	2	2	1	1	1	1

4

The Huffman code (Step 2)

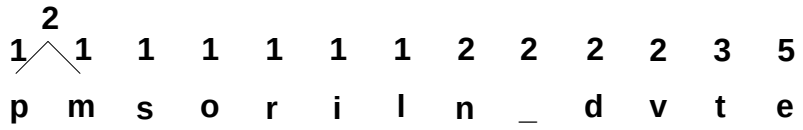
- Build the Huffman tree
  - Sort the table
  - Join the last two items in a node
  - Sort again

p	m	s	o	r	i	l	n	_	d	v	t	e
1	1	1	1	1	1	1	2	2	2	2	3	5

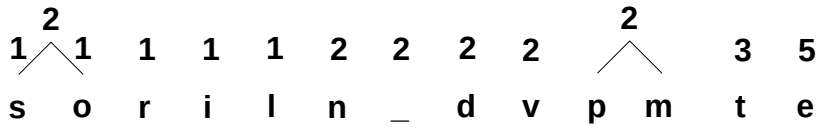
5

The Huffman code (Step 2)

- Join the last two items in a node



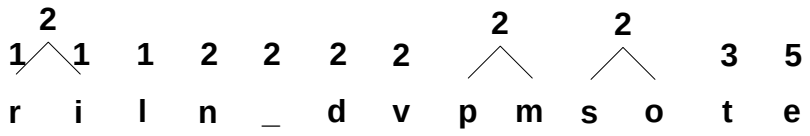
- Sort again
- Join the last two items in a node



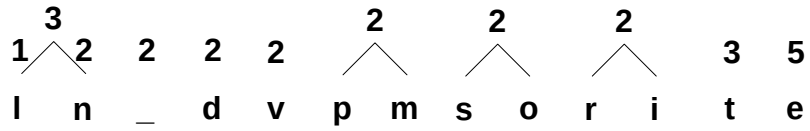
6

The Huffman code (Step 2)

- Sort again
- Join the last two items in a node

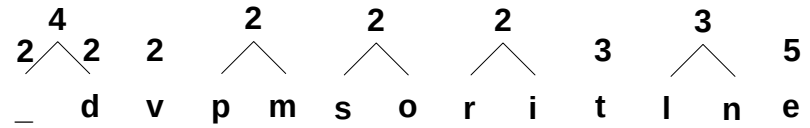


- Sort again
- Join the last two items in a node

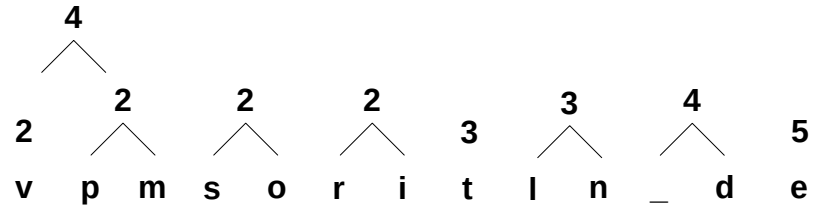


The Huffman code (Step 2)

- Sort again
- Join the last two items in a node

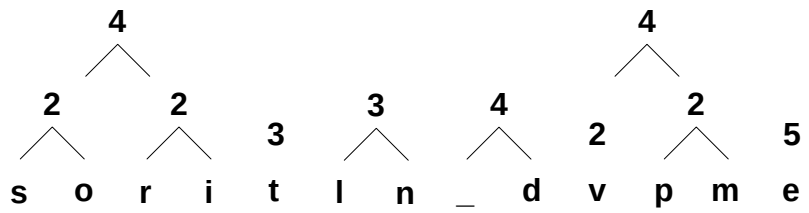


- Sort again
- Join the last two items in a node



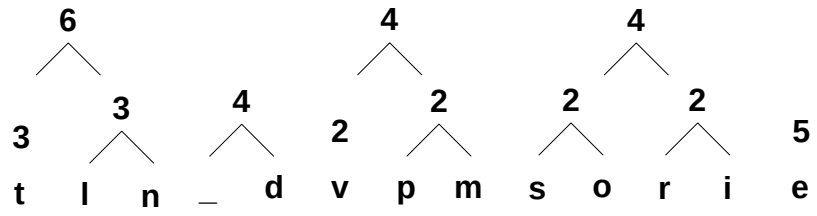
The Huffman code (Step 2)

- Sort again
- Join the last two items in a node



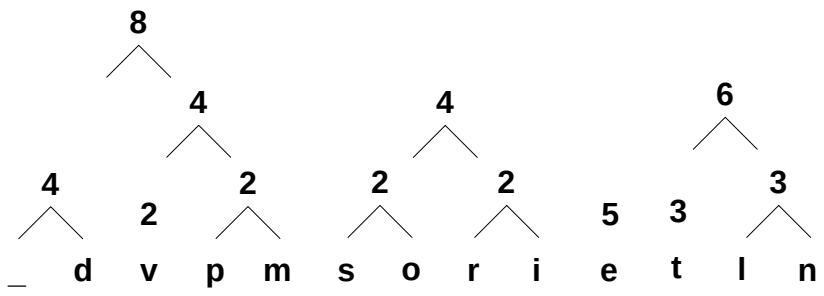
The Huffman code (Step 2)

- Sort again
- Join the last two items in a node



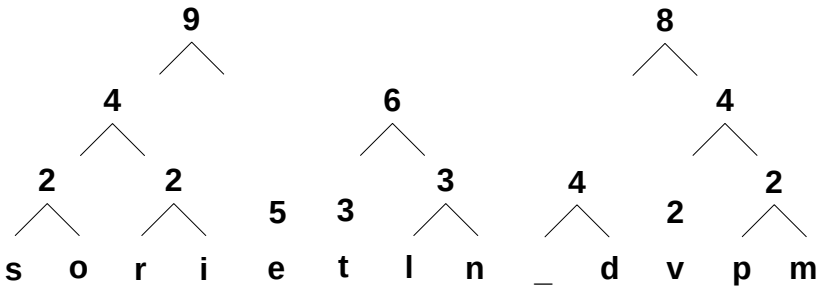
The Huffman code (Step 2)

- Sort again
- Join the last two items in a node



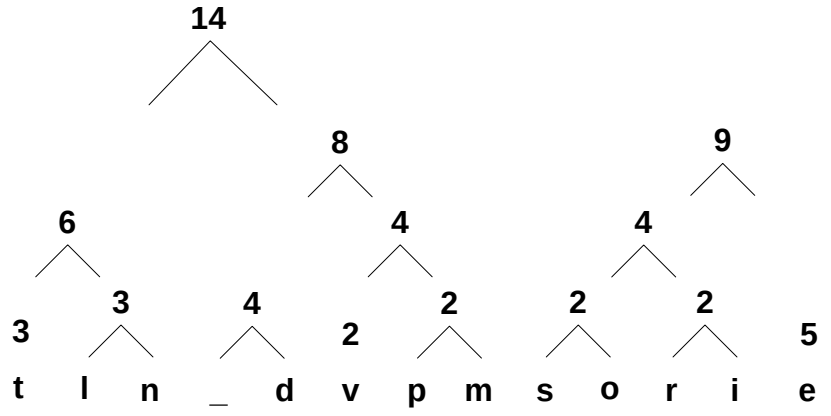
The Huffman code (Step 2)

- Sort again
- Join the last two items in a node



The Huffman code (Step 2)

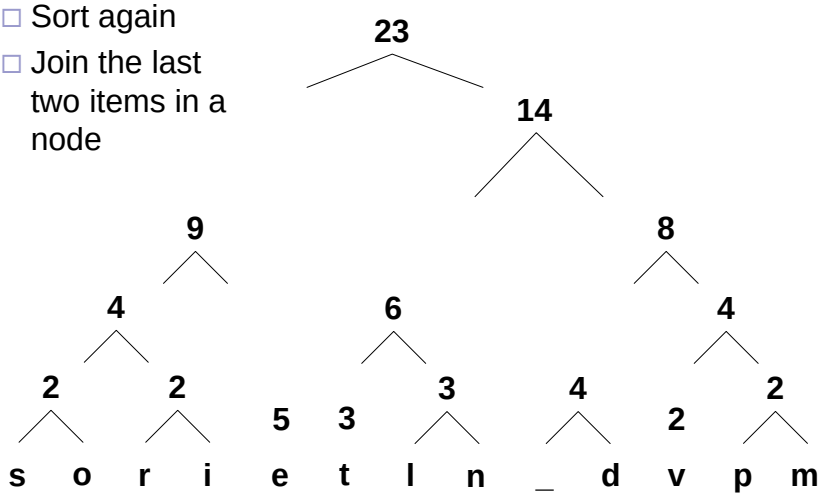
- Sort again
- Join the last two items in a node



13

The Huffman code (Step 2)

- Sort again
- Join the last two items in a node



14

Red/Green/Refactor

- Create a list of tests
- Implement a few just to see the tests fail (**Red**)
- Implement just to pass the test (**Green**)
- Refactor the code (**Refactor**)
- Run the tests to see you did not break anything

15

Implementation

- List of tests
  1. Create the Huffman class verify it can be created
  2. Set the plain text, verify the class returns it
  3. Verify the sum of frequences of all characters to be 23
  4. Verify the frequency table is :

t	e	s	d	r	i	v	n	_	l	o	m	p
3	5	1	2	1	1	2	2	2	1	1	1	1

16

Implementation (cont.)

Test 1

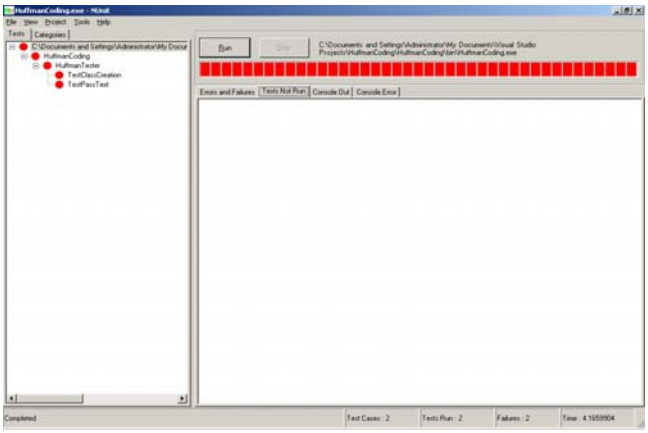
```
<Test()> Public Sub TestCreation ()
   Dim cHuffman as new HuffmanCode
   Assert.AreNotSame(Nothing, cHuffman)
End Sub
```

Test 2

```
<Test()> Public Sub TestPassText()
    Dim cHuffman as new HuffmanCode
     cHuffman.plainText="test_driven_development"
    Assert.AreEqual(cHuffman.plainText,"test_driven_development")

End Sub
```

17

Red



18

Implementation (cont.)

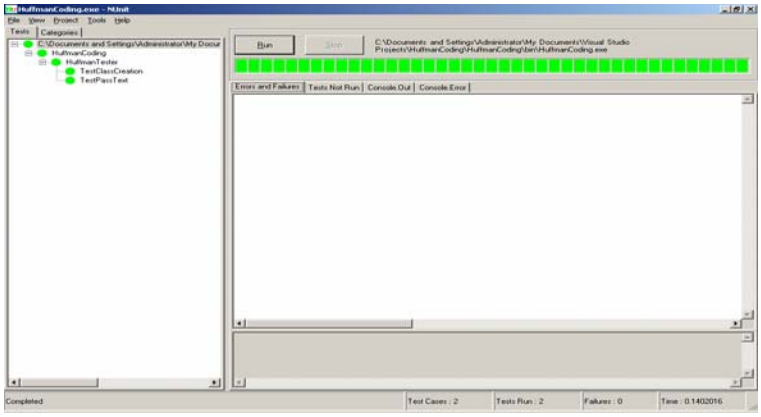
Implement Class

Public Class HuffmanCode
'Just pass the test
  Public Shared **plainText** As String

End Class

19

Green



20

Refactoring

Implement Class

```
Public Class HuffmanCode
'Just pass the test
   Private Shared plainText As String

    Public Shared Sub setPlainText(ByVal Text As String)
       plainText = Text
   End Sub
   Public Shared Function getPlainText() As String
       getPlainText = plainText
   End Function
End Class
```

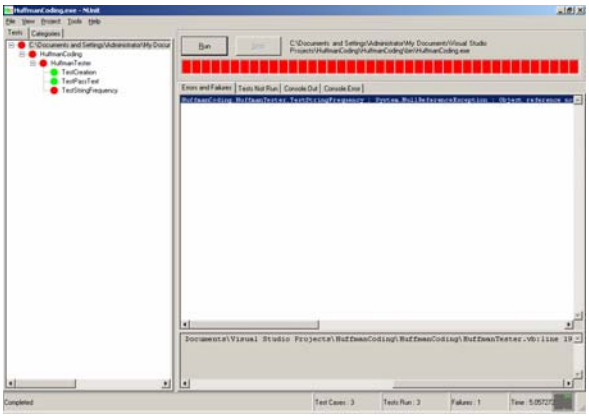
21

Test 3

```
<Test()> Public Sub TestSumFrequency()

      Dim sumFreq As Integer
      Dim i As Integer

      HuffmanCode.generateFrequenceTable()

      sumFreq = 0
      For i = 0 To HuffmanCode.frequencyTable.Length - 1
         sumFreq = sumFreq + HuffmanCode.frequencyTable(i).Frequency
      Next
      Assert.AreEqual(HuffmanCode.getPlainText.Length, sumFreq)

   End Sub
```

22

Oh Red Again!!



23

```
Public Class treeNode
    Public Frequency As Integer
End class

Public Class HuffmanCode
'Just pass the test
  Private Shared plainText As String
  Public Shared frequencyTable() As treeNode

   Public Shared Sub setPlainText(ByVal Text As String)
      plainText = Text
   End Sub

   Public Shared Function getPlainText() As String
      getPlainText = plainText
   End Function


    Public Shared Sub generateFrequenceTable()
    'applied to the plaintext
    end sub

 End Class
```

Focus on this method to pass the test

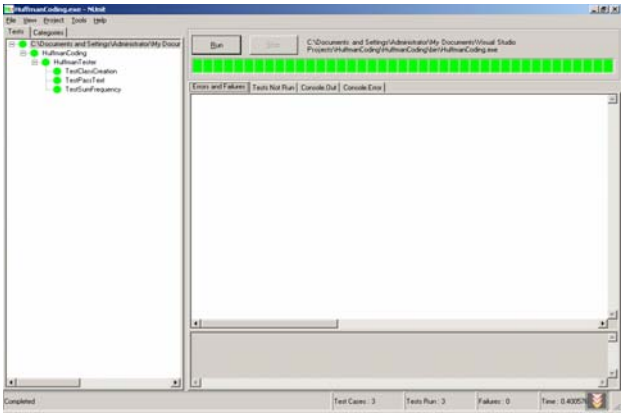
24

```
Public Shared Sub generateFrequenceTable()
    Dim tempresult(-1) As treeNode


    Dim i As Integer
    Dim j As Integer
    Dim found As Boolean
    For i = 0 To plainText.Length - 1
      found = False
      For j = 0 To tempresult.Length - 1
        If Not IsNothing(tempresult(j)) Then
          If plainText.Chars(i).ToString = tempresult(j).Character Then
            found = True
            tempresult(j).Frequency += 1
          End If
        End If
      Next
      If Not found Then
        ReDim Preserve tempresult(tempresult.Length)
        tempresult(tempresult.Length - 1) = New treeNode(plainText.Chars(i).ToString, 1)

      End If
    Next

   frequencyTable = tempresult

End Sub
```

```
Public Class treeNode
Public Character As String
Public Frequency As Integer
End class
```

```
Public Class treeNode
Public Sub New(ByVal Character As String,
ByVal Frequency As Integer)
    Me.Character = character
    Me.Frequency = Frequency
End Sub
```


Tests passed, next test…



Test list

- Create the Huffman class verify it can be created……done
- Set the plain text, verify the class returns it…….done
- Verify the sum of frequences of all characters to be 23……..done
- Verify the frequency table is

t	e	s	d	r	i	v	n	_	l	o	m	p
3	5	1	2	1	1	2	2	2	1	1	1	1

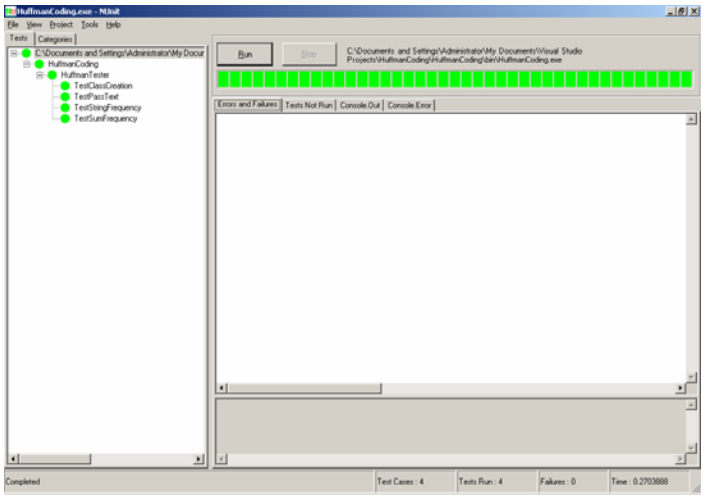
27

Test 4

```
<Test()> Public Sub TestStringFrequency()
    Dim i As Integer
    Dim j As Integer
    Dim FrequencyTable() As treeNode
    FrequencyTable = cHuffman.frequencyTable
    Assert.AreEqual(VerifyTable.Length, FrequencyTable.Length)
    For i = 0 To VerifyTable.Length - 1
      For j = 0 To FrequencyTable.Length - 1
        If VerifyTable(i).Character = FrequencyTable(j).Character Then
          Assert.AreEqual(FrequencyTable(j).Frequency,VerifyTable(i).Frequency)
        End If
      Next
    Next
  End Sub
```

28

Green



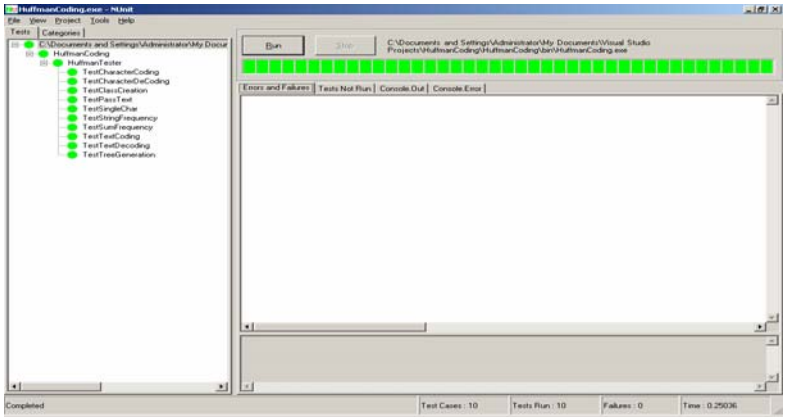
29

Some more tests

5. Verify the tree created is not nothing
6. Verify the character with 100% frequency is coded with 1 bit
7. Verify the coding of “p” is 10011
8. Verify the decoding of “10011” is “p”
9. Verify the text coding of ”test_driven_development” is the known value
10. Verify the text decoding of the known value is ”test_driven_development”

30

All Green



31

Statistics

- 9 test cases
- 81 lines of code for the test
- 141 lines for the implementation
- 4 hours for the whole process

32

Conclusions

- A little bit strange at the beginning
- What to test?? Which first?? How to test?
- May be the TDD must be introduced early in the programmer education
- Anyway at the end you have a very good feeling about your software

33

To whom it may concern:

THANK YOU !

34