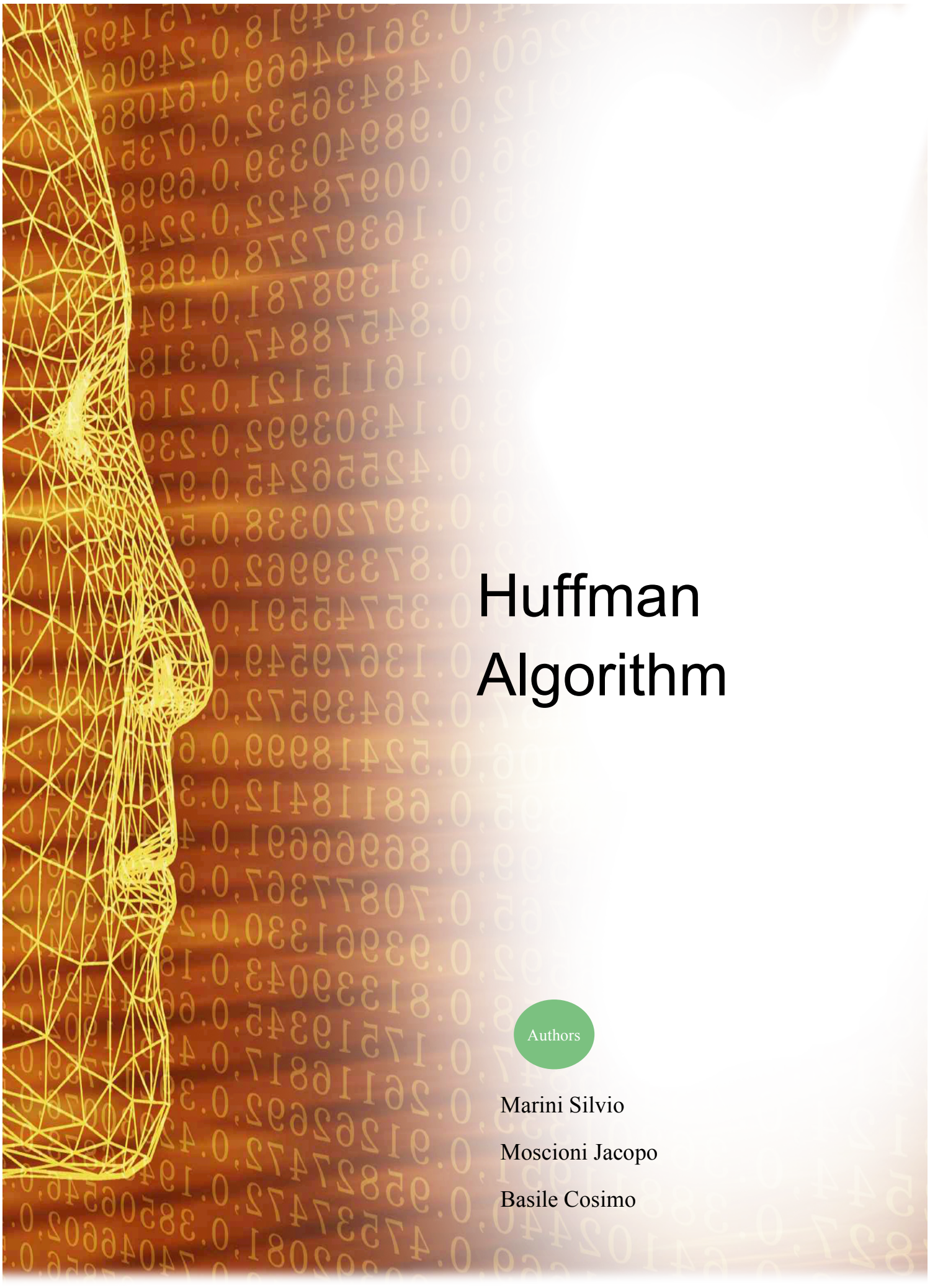


Huffman Algorithm

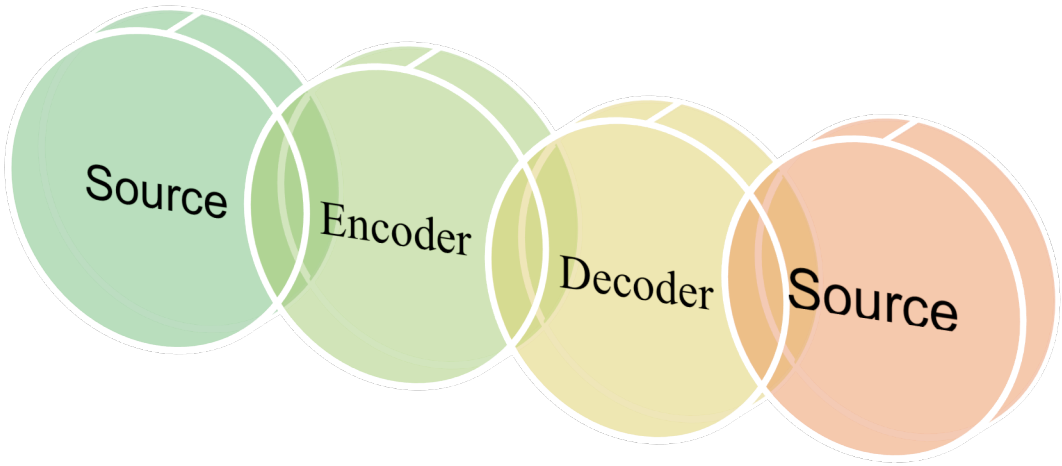
Authors

Marini Silvio

Moscioni Jacopo

Basile Cosimo

SUMMARY



COMPRESSION ALGORITHM

Data compression is a method to represent a source data using fewer bit than the original input. Compression can be lossless or lossy, with lossless compression algorithm after encoding and decoding of a source data, the output will be the same of the input data, so no information is lost in lossless compression. The output data of lossy algorithm instead, can be different from input but acceptable. Neither is better, lossy and lossless algorithms can be better for different contexts.

Why is it useful?

Data compression is usefull to reduces the usage of resources, as storage space of data or trasmission capacity. Compression method can be used to compress all type of data, such as text data, image data, speech data, audio data or video data.

How to compress?

There are several compression algorithm, in Lossless algorith there are Run length encoding , move-to-front encoding , Huffman Algorithm , 7z , Lempel–Ziv–Markov chain algorithm and so on. In Lossy algorithm there are JPEG, MPEG-4, AAC-LD and so on. We talk about the Huffman Algorithm and its application.

HUFFMAN ALGORITHM

Huffman algorithm is a lossless compression algorithm created by Hufffman in 1952 while he was PhD student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes". Huffman coding uses a specific method to choose the representation for each symbol. The encoding obtained depends by the frequency of each symbol. Symbols with high frequency will be rappresented with less charatters than less common symbols. Therefor, the Huffman algorithm works better when the input has not uniform probability distribution.

The running time of Huffman's method is fairly efficient, it takes $O(n \log n)$ operations to construct it .

How it works?

Huffman algorithm gets in input a text called Datavector $D=\{x_1,x_2,..x_n\}$ over the k-ary alphabet $A=\{ a_1,a_2,..a_k \}$. It counts the occurrences of each character and creates the vector of frequency $F=\{ f_1,f_2,..f_w \}$. Thus it makes a binary tree of nodes, each distinct character represents a leaf of the tree, then it creates a parent node for two children that have lowest occurrences. This node is the concatenation of the child nodes and assign a value equal to a sum of the child's value. Huffman algorithm repeats this process until it creates the root. Now the algorithm assigns a bit for each node and the leaves of tree have a code create by the root-leaf path, so it creates a code word for each symbol. The more

frequent symbols corresponds to a shorter code words. The code words obtained by Huffman algorithm create a Kraft vector. We can calculate the average cost for each symbol with

$$l = \sum_{i=1}^{k} p(a_i) l_i$$

Where l is the average length of symbol , $p(a_i)$ is the probability of the symbol a_i and l_i is the length of code word of a_i .

This formule allow us to declare that the Huffman coding is the best coding for the input Datavector D.

Variations

Until now we talk about of a specific variation of original Huffman algorithm, called Adaptive Huffman Coding. There are many other variations like n-ary Huffman Code, Huffman template algorithm, lenght-limited Huffman coding/minimum variance Huffman coding, Huffman coding with unequal letter costs and Optimal alphabetic binary trees.

The difference between basic Huffman code and adaptive variation is that in the adaptive Huffman algorithm it create a new prefix vector based to the occurrences of symbol in input text for each new input text, it is very rare in practice, because the cost of updating the tree makes it slower than the basic Huffman algorithm but that is more flexible and has a better compression.

TRANSFER PROTOCOL

The output of algorithm Is made by coding Symbol list + Specific sequence of characters + Huffman coding.

Coding symbol list Is the list of symbol that use 9 bits for esch symbol, 8bits for the ASCII code of the symbol and the ninth is a control bit for check if the symbol is finished or not. example : t is 01110100 and r is 01110010 , if we want to code symbol tr and symbol r, we write 011101001011100100011100100 where black bits are the ASCII code of symbol and brown bits are the control bits, 0 for stop the reading of the symbol and 1 to continue it.

Specific sequence of characters is a sequence of nine 0 : 000000000, when the algorithm read it know that the symbol list is finished and from that point will be the Huffman coding .

EXAMPLE

We consider as input text: “trentatre trentini entrarono in trento tutti e trentatre trotterellando” .

The datavector associated is: V={ t,r,e,n,t,a,t,r,e, ,t,r,e,n,t,i,n,i, ,e,n,t,r,a,r,o,n,o, ,i,n, ,t,r,e,n,t,o, ,t,u,t,t,i, ,e, ,t,r,e,n,t,a,t,r,e, ,t,r,o,t,t,e,r,e,l,l,a,n,d,o }

Over the alphabet : A={t,r,e,n,a, ,i,o,l,u,d}

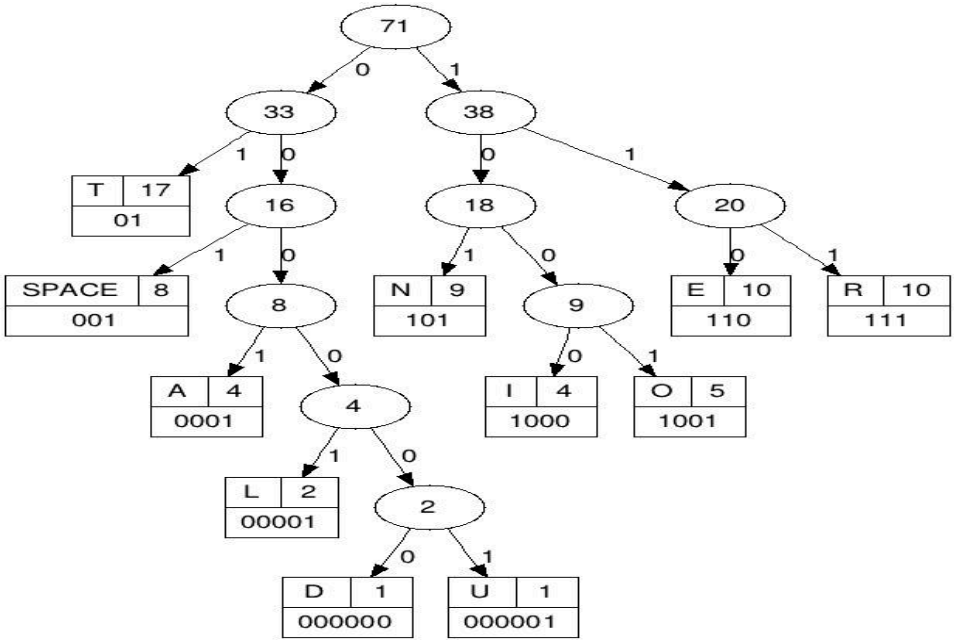
Count of occurrencies:

T:17 , :8 , N:9 , E:10 , R:10 , A:4 , I:4 , O:5 , L:2 , D:1 , U:1

The vector of frequences is: F={17,10,10,9,8,5,4,4,2,1,1}

And the vector probability: P={0.24 , 0.14 , 0.14 , 0.13 , 0.11 , 0.07 , 0.06 , 0.06 , 0.03 , 0.015 , 0.015}

Now we can create the tree of nodes with the symbol:



With tree of nodes we obtain code word of each symbol:

T : 01	_ : 001	N : 101	E : 110	R : 111	A : 0001
I : 1000	O : 1001	L : 00001	D : 000000	U : 000001	

The average length for this code is:

L = 0.24*2 + 0.14*3 + 0.14*3 + 0.13*3 + 0.11*3 + 0.07*4 + 0.06*4 + 0.06*4 + 0.03*5 + 0.015*6 + 0.015*6 = 3.13 bits/symbol

Length of source text: 71 symbol.

Length of source (8bits*symbol) : 71 symbol * 8bits = 568 bits

Huffman code:
01111110101010001011111100010111111010101100010110000011101010111100011111001101100100110001010010111111010101100100101000001010110000011100010111111010101000101111110001011111001010111011111000001000010001101000000 01001

Length of Huffman code : 219bits

Compression ratio: $$r = \frac{n \log_2 |A|}{k}$$

r = (568 * log | 11 |) / 219 = 8,97 to 1

$$R = \frac{k}{n}$$

Compression rate:

R = 0.385 codebits per data sample.

Percentage of compression: 38,5%

MODIFICATION OF ALGORITHM

We propose this modification of algorithm:

1. Use the Huffman algorithm with characters and more frequently words. Create a list of words and its occurrences inside the text. Choose the more frequent words and try to add it in the symbol list and execute Huffman Algorithm to check if it is convenient. Repeat this for the most common words in the text.

2. Use the Huffman algorithm with characters and more frequently sequences of letters. To do this we scan the text character by character and create an Array key-value, that contains the sequence of characters and its frequencies. We will calculate the value of the new symbol based on the number of occurrences and on the characters that it cover, this value is called by us VoS (value of substitution). The symbol with the higher VoS will be added in the list of symbol and will be calculate again the new list of symbol with its occurrences. Now we can execute Huffman algorithm with newest symbol list and check if it execution is better than the last. We repeat this iteration until we find the first symbol list that have a worst cost of coding. At the end will have the best symbol list and can coding the text with it and concatenate the symbol list for permit the decompression.

Method 2 _ description

- Create the symbol list with the occurrences of each characters, execute Huffman algorithm and save the weight of generated code.
- Create a new symbol list for each couples of characters, find the symbol with best VoS and add it in the old symbol list.
- Execute Huffman algorithm and valuate if this is better than the older
- Repeat the last two points until the new coding is better than the older one.

Method 2 _ example

Source: trentatre trentini

Symbol list with only characters: t:5 , r:3 , e:3 , n:3 , I:2 , a:1 , _:1

Execute Huffman Algorithm: 0101101110011111001011011111101011011100111101110111110

Cost: 54bits (Huffman Code) + 7*9=63bits (Symbol list) = 117bits

New symbol list with couples of characters: tr:3 , re:3 , en:2 , nt:2 , ta:1 , at:1 , et:1 ,in:1 , ni:1 , ti:1 , e_:1 , _t:1

Calculate VoS for each couple: VoS(tr) = 60% + 100% / 2 = 80% , VoS(re) = 100% + 100% / 2 = 100% ….

Choose the best Vos and add it in the symbol list, new symbol list: t:5 , re:3 , n:3 , a:1 ,i:1 , _:1

Execute Huffman Algorithm: 0101100111001011111010110011110111011110

Cost: 39bits (Huffman Code) + 63bits (Symbol list) = 102bits

The new symbol list can use symbol “re” as a single character.

New symbol list with couples of characters: tre:3 , ren:2 , nt:2 , ta:1 , at:1 , re_:1 , _t:1 , ,in:1 , ni:1 , ti:1

Calculate VoS for each couple: VoS(tre) = 60% + 100% / 2 = 80% , VoS(ren) = 66% + 66% / 2 = 66% , VoS(nt) = 40% + 66% / 2 = 48%

Choose the best Vos and add it in the symbol list, new symbol list: tre:3 , n:3 , t:2 , a:1 , i:1 , _:1

Execute Huffman Algorithm: 0101101110011110010110111111011111

Cost: 34bits (Huffman Code) + 81bits (Symbol list) = 115bits

The algorithm ends because the newest symbol list is worse than the previous one.

We will implement this modifications and then we can calculate if that modifications are better than original algorithm or not.

ALGORITHM’S IMPLEMENTATION

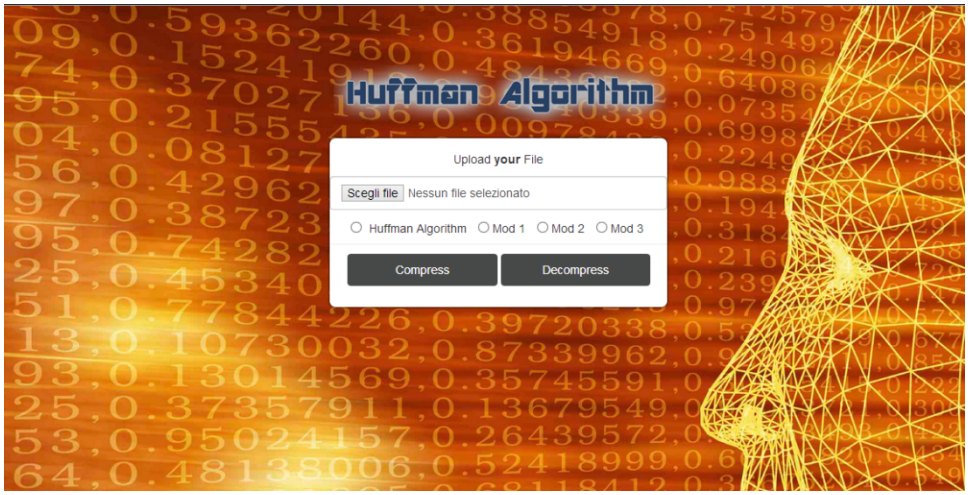
1. Get in input the source file
2. count the occurence for each charatters found in the sentence and make the tree in according to the Huffman’s algorithms, from datavector to vector of occurences.
3. applying huffman algortim to create an Huffman Code.
4. Create a prefix set from Huffman Code
5. Coding datavector with the prefix set
6. Calculate the Compression Ratio and Compression rate of coding

Modifty of Algorithm:

1. Apply Huffman algorithm also with the occurrences of the most frequencies words in the text
2. Calculate Compression ratio and check if the compression is better
3. Apply Huffman algorithm also with the the most frequencies sequences words in the text
4. Calculate Compression ratio and check if the compression is better

We decide to use PHP and create a web application.

The web application is based to modern language like HTML5 and CSS3, we create a form where the user can upload a XML file and choose if to apply our modification to original Huffman algorithm or not. Then the web application will create a compressed code and show it to the user with information about compression ratio and compression rate.



1 readFile(inputFile.xml)

Php has a function to open, read and write a stored file, it's called "fopen()" and it accepted two parameters: the first one is for the name of input file and the second one is for the access method (read only, read and write, and so on).

2_ countOccurences(fileContent)

We can process the content of input file to count the occurrences of all charatters with the php function called count_chars(), it Counts the number of occurrences of every byte-value (0..255) in string and returns it in various ways. After that we can create the vector of occurences.

3_ createArray(occurence,value)

repeat

Merge the first and second minimum value of occurence and update value sequence

Until (the array has only one cell)

Take Huffman Code by the content of remain cell, that contains the array with our huffman code

We have to create Array of Array, that must be ordered in descendent mode and each array must have inside the value of occurence and a sequence of values. Find the two minimum value of occorunce and merge it for create a new array with the sum of occurence in the first cell and a concatenate sequence of value increased by 1 in other cells. Repeat this method until remain only one cell. The remaining array in first cell except the occurences value will be the Huffman Vector.

4_ create prefix set from Huffman Vector

One method to produce a prefix vector is a priority queue. At the beginning create a leaf node for each symbol and add it to priority queue. While there is more than one node, remove the two nodes with lowest probability and create a new parent node with probability equal of the sum of the probability of the nodes just removed. Add the new node in the queue, and continue the iteration until the condition is true. At the end the remaining node is the root and tree is complete. Now the code word for each symbol is the root-leaf path.

5_ save new vector with the coding text, replacing each symbol with its code word.

6_ calculate Compression ratio and compression rate

$$r = \frac{n \log_2 |A|}{k} \qquad R = \frac{k}{n}$$

SITOGRAPHY

- http://www.mini.pw.edu.pl/~herbir/www/data/Notes%201%202012.pdf
- http://www.mini.pw.edu.pl/~herbir/www/data/notes%202.pdf
- http://www.mini.pw.edu.pl/~herbir/www/data/notes%203%202012.pdf
- http://en.wikipedia.org/wiki/Lossless_compression
- http://en.wikipedia.org/wiki/Huffman_coding