Data Compression

What is Compression?

The compression is the process of encoding data more efficiently to achieve a reduction in the data size.

Why?

- Storing or transmitting multimedia data requires large space or bandwidth – it reduces the necessary space or time.
- ***Examples***
    - 1 hour audio, CD quality, stereo: 44100 Hz x 2 B x 2 channels x 3600 s ≈ 635 MB → MP3 compression can reduce it about 10 times.
    - Image: True Colour (24 bit), 1000 x 500 pixels → 1,5 MB; JPEG compression can reduce it 10 to 20 times.
    - Size of 1 minute real-time, full size, color video clip at 640x480 pixels is 60 seconds x 30 frames x 640 x 480 x 3 B = 1.659 GB (2 hours would take about 200 GB). MPEG-4 can reduce it 20 to 200 times.

Figure 1: Another way of how to compress data (source: http://hughewilliams.files.wordpress.com/2014/03/datacompression.jpg?w=625)

Pros and Cons

+ When compressed, the quantity of bits used to store the information is reduced.
+ Files that are smaller in size will result in shorter transmission times when they are transferred on the Internet. Compressed files also take up less storage space.
+ File compression can zip up several small files into a single file for more convenient email transmission.

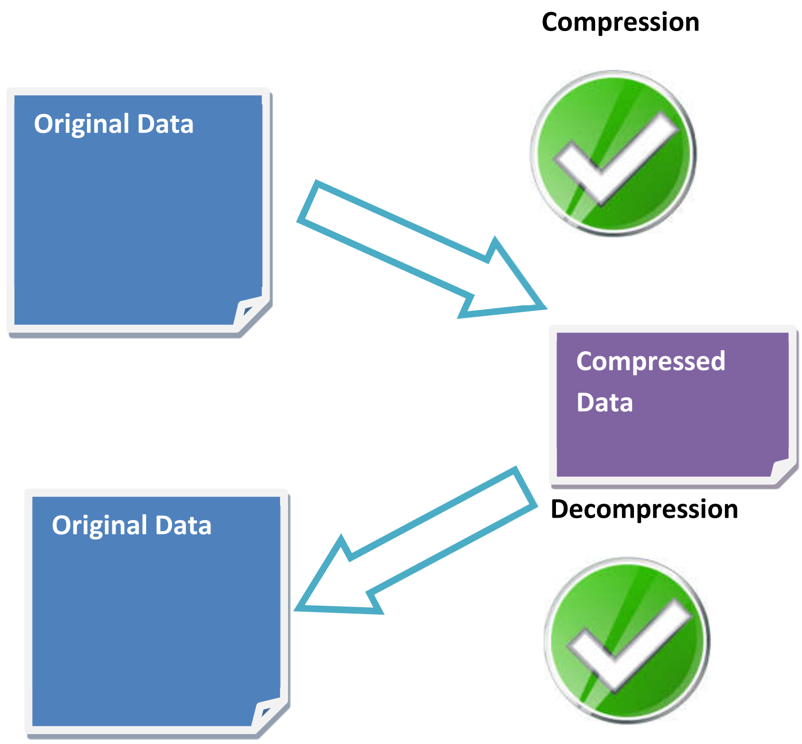
– Compression is a mathematically intense process, it may be a time consuming process, especially when there is a large number of files involved.

- Some compression algorithms also offer varying levels of compression, with the higher levels achieving a smaller file size but taking up an even longer amount of compression time. It is a system intensive process that takes up valuable resources that can sometimes result in "Out of Memory" errors.
- With so many compression algorithm variants, a user downloading a compressed file may not have the necessary program to un-compress it (e.g. some users experienced problems with 7z files).

Types of Data Compression

Lossless Compression

Losslessly compressed data can be decompressed to exactly its original value. This kind of compression is essential for files, where each bit is important for correct use – documents, setup files, database files etc.



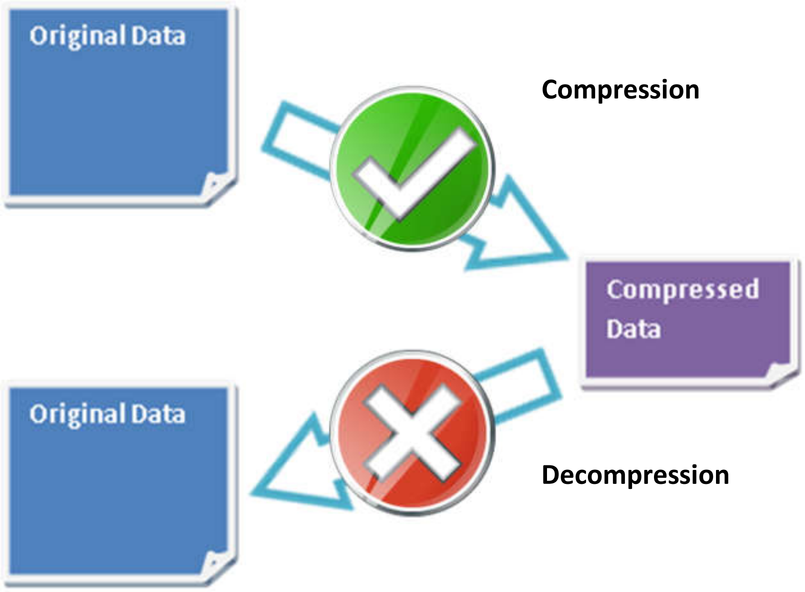
Lossless compression algorithms use statistic modeling techniques to reduce repetitive information in a file. Some of the methods may include removal of spacing characters, representing a string of repeated characters with a single character or replacing recurring characters with smaller bit sequences.

Examples

- General purpose: BZip2, DEFLATE (ZIP), LZMA (7-ZIP)
- Audio: FLAC
- Graphics: PNG, WebP

Lossy Compression

Lossy compression discards "unimportant" data, for example, details of an image or audio clip that are not perceptible to the eye or ear. Lossy compression is acceptable in the case of multimedia data, where the limitations of human senses provide opportunities to discard some details without being noticed.



Examples

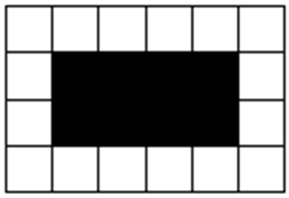
- Graphics: JPEG
- Music: MP3, Ogg Vorbis, WMA
- Video: MPEG-2, MPEG-4, H.264

There is no such thing as a "universal" compression algorithm that is guaranteed to compress any input, or even any input above a certain size. In particular, it is not possible to compress random data or compress recursively (that means, re-compress data, which were already compressed).

Classification of Compression Algorithms

Run-length Coding

- Very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.
- Example



  - A scanline of the binary image (8-bit, grey only) is
    11111111 11111111 11111111 11111111 11111111 11111111
    11111111 00000000 00000000 00000000 00000000 11111111
    11111111 00000000 00000000 00000000 00000000 11111111
    11111111 11111111 11111111 11111111 11111111 11111111
    Total of 192 bits
  - However, strings of consecutive 0's or 1's can be represented more efficiently: 1(56) 0(32) 1(16) 0(32) 1(56)
  - If the counts can be represented using 6 bits (max. count is 64), then we can reduce the amount of data to 5+5x6=35 bits. A compression ratio is about 80 %.

Huffman Coding

- The basic idea behind Huffman coding algorithm is to assign shorter code words to more frequently used symbols
- Example:

"***sent session message***"

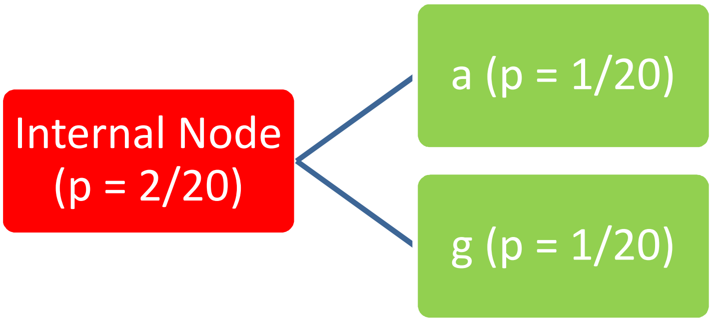
A frequency table is built:

Symbol	**Count**	**Probability**
s	6	6/20
e	4	4/20
n	2	2/20
space	2	2/20
t	1	1/20
i	1	1/20
o	1	1/20
m	1	1/20
a	1	1/20
g	1	1/20

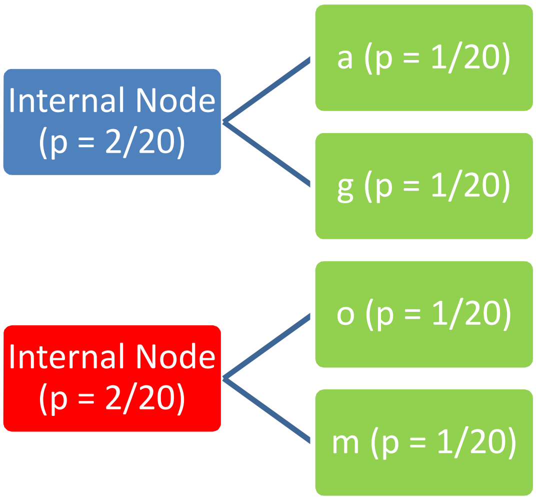
Next, make a binary tree using the following rules:

1. Create a leaf node for each symbol and add it to a queue.
2. While there is more than one node in the queue:
    1. Remove two nodes of the lowest probability from the queue
    2. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    3. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

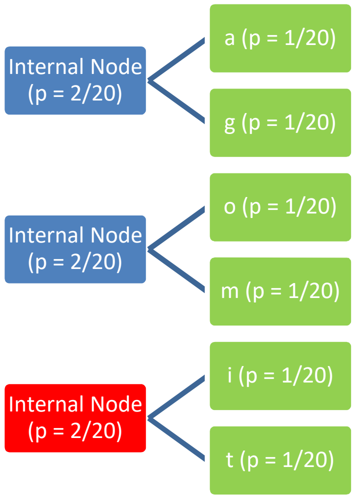
In our case:



Then:

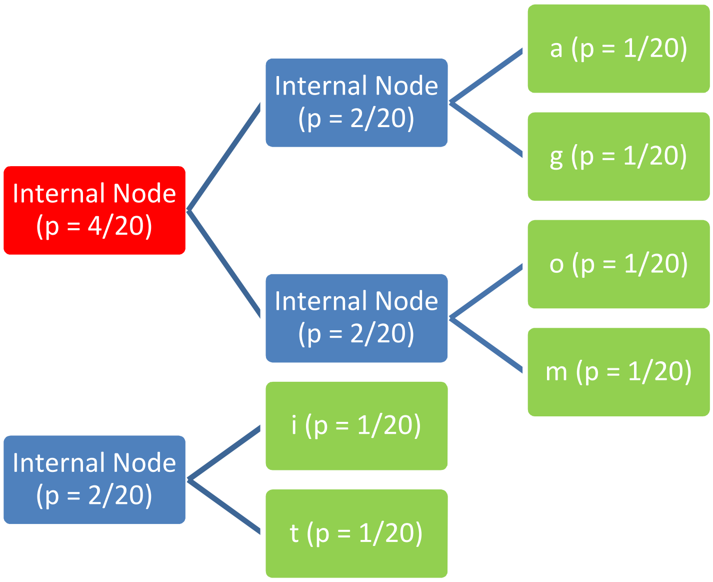


Then:



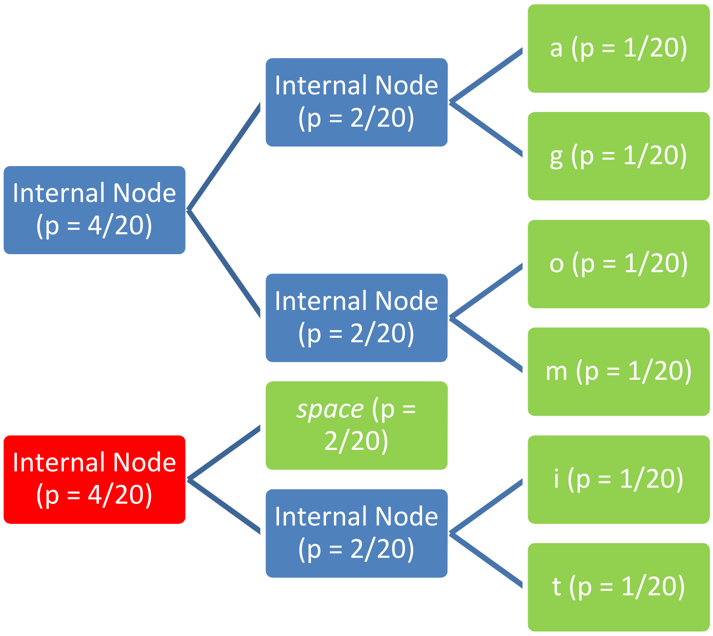
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

Then:



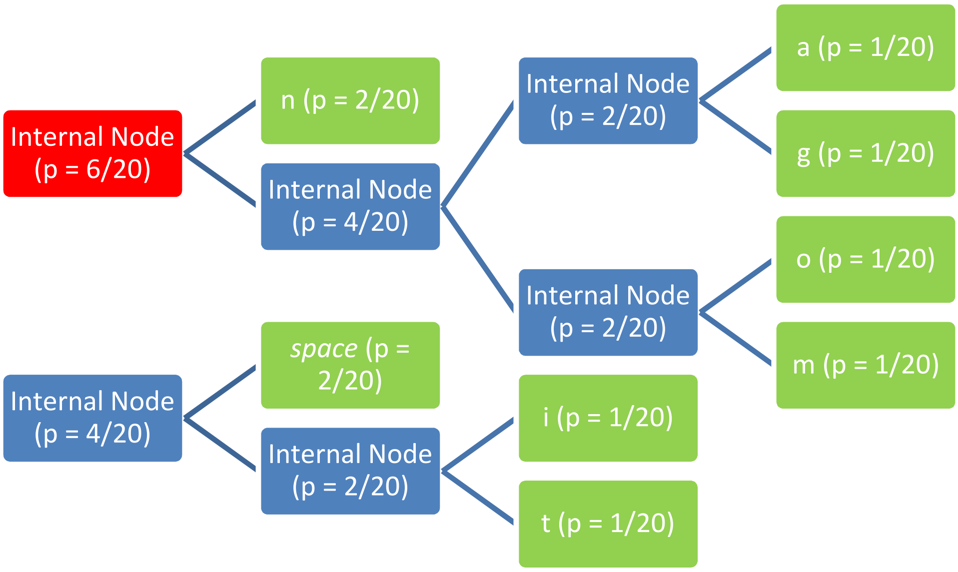
Internal Node (p = 4/20)
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

Then:



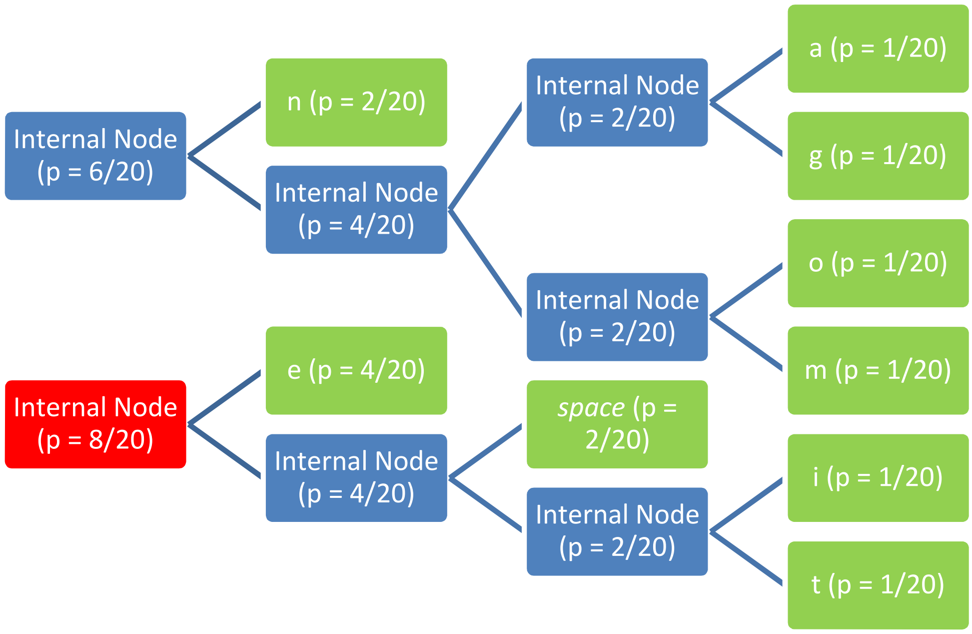
Internal Node (p = 4/20)
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 4/20)
space (p = 2/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

Then:

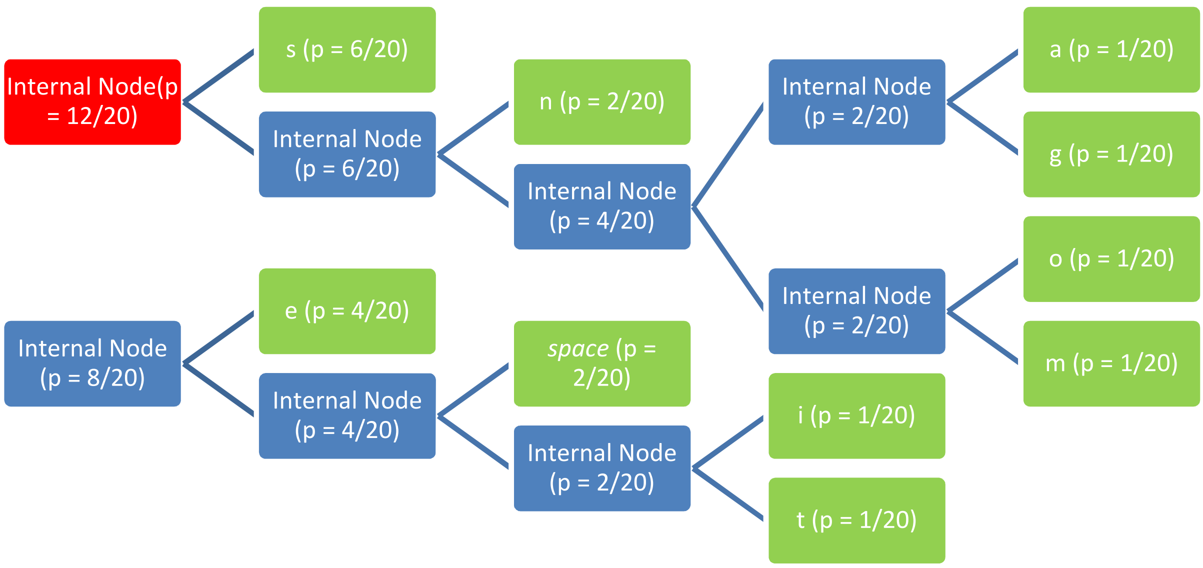


Internal Node (p = 6/20)
n (p = 2/20)
Internal Node (p = 4/20)
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 4/20)
space (p = 2/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

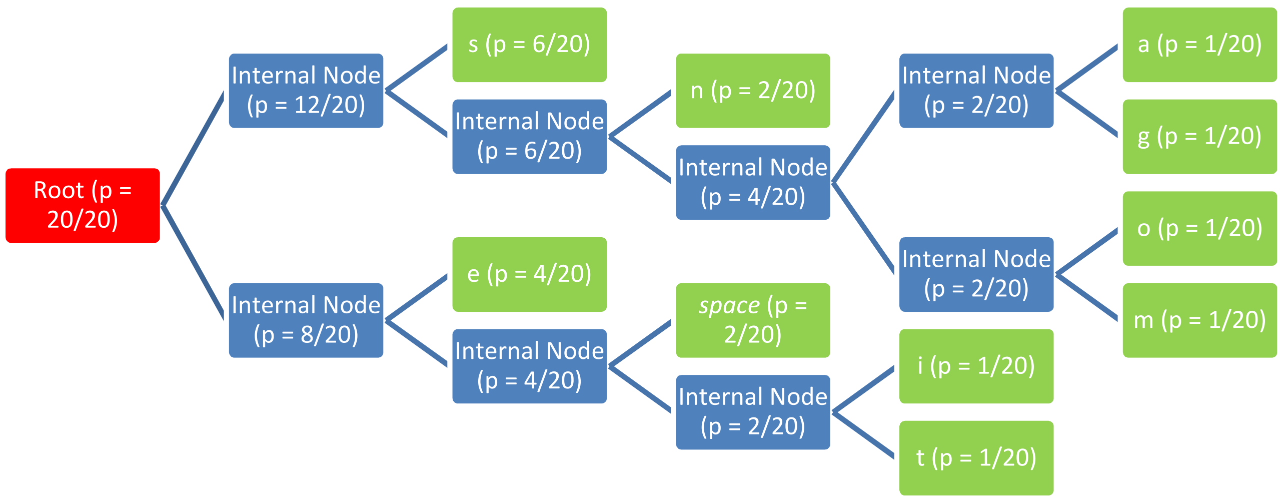
Then:


Internal Node (p = 6/20)
n (p = 2/20)
Internal Node (p = 4/20)
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 8/20)
e (p = 4/20)
Internal Node (p = 4/20)
space (p = 2/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

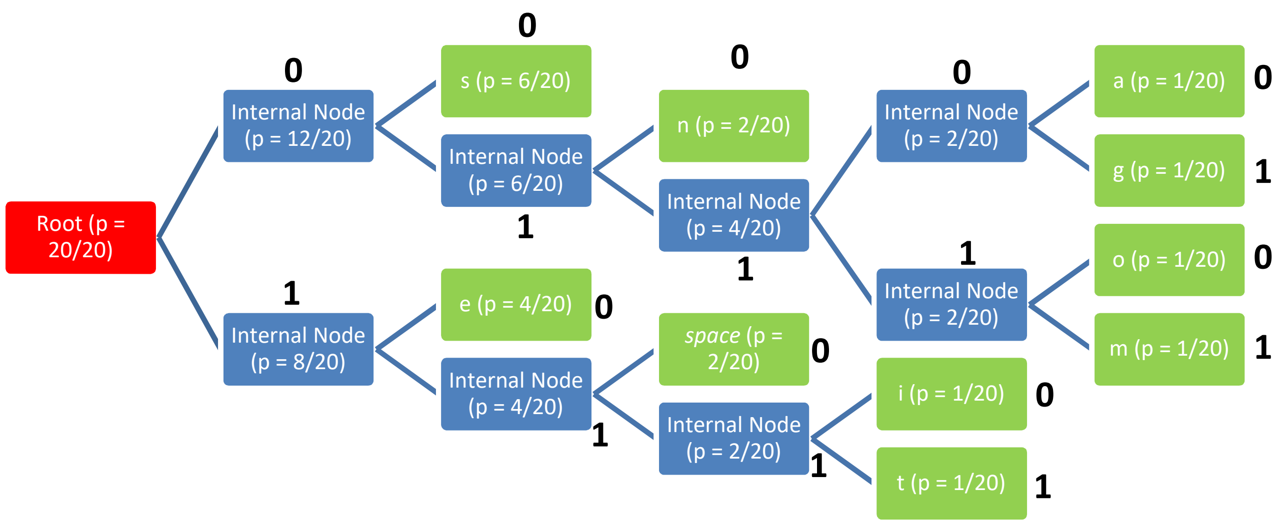
Then:


Internal Node(p = 12/20)
s (p = 6/20)
Internal Node (p = 6/20)
n (p = 2/20)
Internal Node (p = 4/20)
Internal Node (p = 2/20)
a (p = 1/20)
g (p = 1/20)
Internal Node (p = 2/20)
o (p = 1/20)
m (p = 1/20)
Internal Node (p = 8/20)
e (p = 4/20)
Internal Node (p = 4/20)
space (p = 2/20)
Internal Node (p = 2/20)
i (p = 1/20)
t (p = 1/20)

And, finally:



The last phase – starting from the root label the children – first is 0, the second is 1. Repeat until all nodes are labeled.



The final coding table is made of 0s and 1s read from the root towards the given symbol:

Symbol	**Code**
s	00
e	10
n	010
space	110
t	1111
i	1110
o	01110
m	01111
a	01100
g	01101

The code assembled from the tree satisfies one mandatory condition for any code: there is no valid code word in the system that is a prefix (start) of any other valid code word in the set.

"sent session message"

Huffman: 00 10 010 1111 110 00 10 00 00 1110 01110 010 110 01111 10 00 00 01100 01101 10 = 60 bits

ASCII: 20 symbols x 8 bits = 160 bits

Exercises

1. Create the compressed version of this text (ASCII 8-bit):

    01010100 01101000 01100101 01100010 01100001 01110011 01101001 01100011
    01101001 01100100 01100101 01100001 01100010 01100101 01101000 01101001
    01101110 01100100 01001000 01110101 01100110 01100110 01101101 01100001
    01101110 01100011 01101111 01100100 01101001 01101110 01100111 01100001
    01101100 01100111 01101111 01110010 01101001 01110100 01101000 01101101
    01101001 01110011 01110100 01101111 01100001 01110011 01110011 01101001
    01100111 01101110 01110011 01101000 01101111 01110010 01110100 01100101
    01110010 01100011 01101111 01100100 01100101 01110111 01101111 01110010
    01100100 01110011 01110100 01101111 01101101 01101111 01110010 01100101
    01100110 01110010 01100101 01110001 01110101 01100101 01101110 01110100
    01101100 01111001 01110101 01110011 01100101 01100100 01110011 01111001
    01101101 01100010 01101111 01101100 01110011

    Devise a version for run-length compression as well as for the Huffman coding.

Wordstock

character	znak
printable	tlačiteľný
grapheme	grafémа
glyph	glyf