Morf code

Kamil Landa

made in **LibreOffice** http://libreoffice.org 5/2022

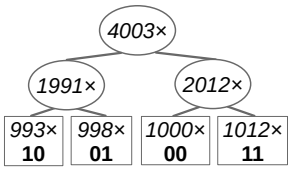
Morf code is a draft of 2-pass compress algorithm inspired by Huffman&Vitter codes. The binary tree needn't be static *(Huffman)* nor be only upsized *(Vitter)*, but could be dynamically up/down-sized.

My English isn't so good, so please be kind to me and my mistakes :-).

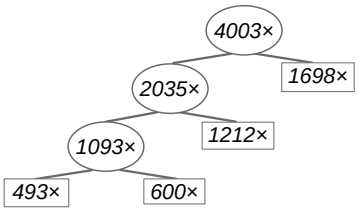
The algorithm uses other severance of sequence of bits than a standard severance to bytes, 2-bytes, 4-bytes etc. The characters with the fixed width have very uniform Counts in the compressed file *(like AVI, ZIP etc.)*, so the tree is upsized to full size very quickly at start, and effect of downsizing is very small at the end.

*The example with "byte" characters, for simplification only for 2-bits characters. Their counts in compressed file are for example: **00**: 1000×, **01**: 998×, **10**: 993×, **11**: 1012×; and the Huffman tree is symetric. I tried to find the characters with the unequal Counts to get an unsymetric binary tree, maybe there is the bigger chance the up/down-sizing will scrimp more bits.*

symetric tree



unsymetric tree

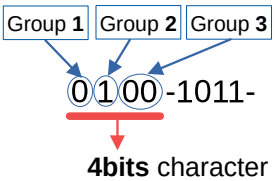


The Character is given by 3 Counts of bits from 3 Groups with same bits; the Group is only the Count of same bits. *Probably this definition isn't so luculent but I didn't discover easier one; but I believe the example will elucidate it.*

For example 5-bytes word **KaMiL**:

	K	**a**	**M**	**i**	**L**
Ascii Dec	75	97	77	105	76
Ascii Hex	#4B	#61	#4D	#69	#4C
Binary	0100 1011	0110 0001	0100 1101	0110 1001	0100 1100

whole sequence 0100101101100001010011010110100101001100



The Characters are 3 Counts of 3 same-bits Groups:

0100-1011-0110000-101-00110-1011-0100-101-001100

0100 = **1-1-2**

0100	**1011**	**0110000**	**101**	**00110**	**1011**	**0100**	**101**	**001100**	–
1-1-2	**1-1-2**	**1-2-4**	**1-1-1**	**2-2-1**	**1-1-2**	**1-1-2**	**1-1-1**	**2-2-2**	leftover

Group 1
Group 2
Group 3

Because there is the periodic alternating of zeroes and ones, the **zeroes-ones-zeroes** are the same characters as **ones-zeroes-ones**. For example **010** is same as **101** → both is the character **1-1-1**. *It is needy only to remember 1st bit of whole sequence and then only changed the **zeroes-ones-zeroes/ones-zeroes-ones** for odd\even characters :-).*

There is very many types of these characters in the compressed file, I counted some 200MB AVI and some fewMB ZIPs and there were about 12 000 types of characters in the AVI; of course all few-bits characters (**3bits**: **1-1-1**; **4bits**: **1-1-2**, **1-2-1**, **2-1-1**; **5bits**: **1-1-3**, **1-3-1**, **3-1-1**, **1-2-2**, **2-1-2**, **2-2-1**; **6bits** **1-1-4**, …), but also very long characters like 12-307-1251, 125-345-57, 1012-2-708 etc. → but these long characters were only once or twice. So it isn't needful to put these alone/twins characters to the tree(s) if you know the positions of first/last occurences of characters in the sequence.

The formula for count of types **N**-bits characters probably is:

$$\textbf{Count of Nbits} = \frac{(N-2)\cdot(N-1)}{2}$$

3bits 1×, **4bits** 3, **5bits** 6, **6bits** 10, **7bits** 15, **8bits** 21 etc.

I named these metamorphose characters 'morphs' but easier in Czech is: **morfs** :-).

> *When I programed it the main problem was, how to get the morfs with their counts really quickly :-). The characters have different bit-lengths and I used (in "inline Asm in C") the Shift instructions for shifting to CF and JC/JNC to Count the bits, because I knew it from x86-Asm for PC 286/386 from high school from the end of last century, and I wasn't able to explore newer instructions for x86-64, it was complicated for my English :-). Maybe you know some faster process.*

Up/down-sizing of tree: in contrast to Vitter tree, there isn't the Null character for unknown morf, so maybe the whole tree will be rebuilded sometimes, probably similarly for downsizing. And it seems minimally two trees will be needy *(one small and more static than dynamic, and one big and more dynamic than static)*, because if only one tree is used, then one from 4bits morfs will be in 5bits floor, one of 5bits morfs will be in 6bits floor etc. → *it is the same situation like in Huffman or Vitter, some characters should be shortened, but some lengthened, but the lengthening of morfs isn't cheering.*

If the started/ended positions of morfs are known, then these last morfs aren't in the tree, so it means the morf is removed from a tree at his penultimate occurence. And next possibility is to write more positions to the Head informations, for example with 2nd positions of morfs → in that case the morfs should be in the tree from their 2nd occurrences.

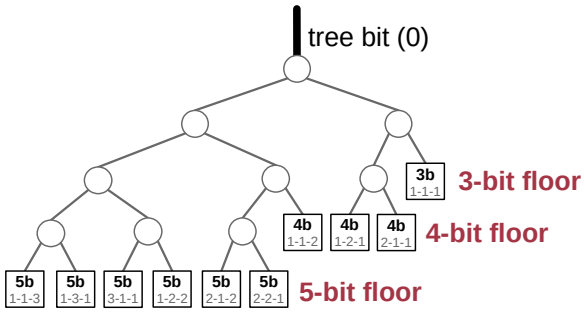
The positions&offsets *(like in dictionary methods)* can be attractive, but the rebus is, how to create the smallest Head for all informations :-), because the long-morfs could take many bytes in Head only to write their names → 1245-8-147, 12-1986-13, 1878-1936-254 etc., and these long morfs are very irregulared :-).

So use the offsets is meaningful because it downsizes the tree, but if the information about offsets in the Head will be shorter than offsetted morf. And how to write the positions&offsets to the Head? At least with a Huffman code, or try a recursion with a Morf code?

Two count-trees

Next reasoning is for the static trees, for the dynamic trees it is unpredictabled.

If the formula for **Count of N-bits** is well then it seems: With two count-trees some [primarily short] morfs will have the same lengths, but some long-morfs will be shortened, and there will not any elongated morf.

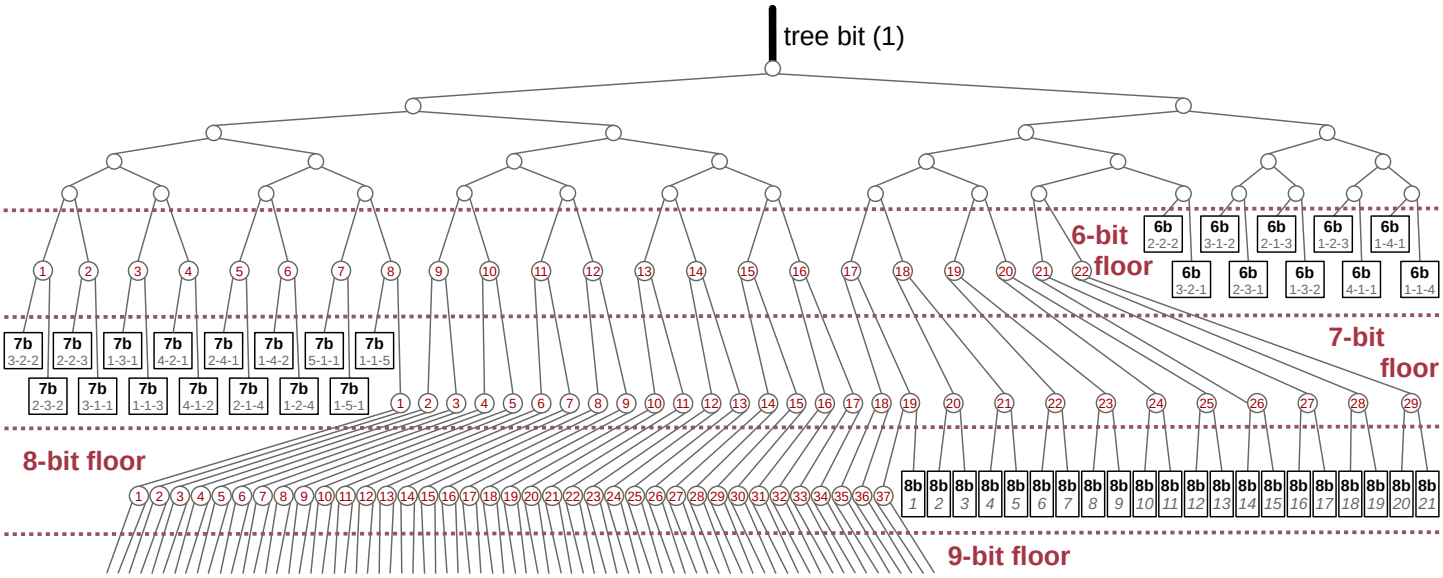


1st tree: the **3bits**, **4b** and **5b** morfs have the same bit-lengths. There isn't a space for other morfs in the tree, but the fact is, these morfs are the most times in compressed files.

2nd tree is a big tree, there are ≥**6bits** morfs without alone/twins morfs *(that are given by their start/end positions)*. The idea is no lengthening of morfs, so **6b** morfs must be maximally in **6-bit floor**, **7b** in **7-bit floor** etc., and it seems the tree allows it.

It was a problem to make so big tree (the space on a page isn't so big), so there isn't a Vitter's ordering for nodes and branches in the image.

And too much big tree probably will the disadvantage for the speed of processing :-(.



Nbits morf	6b	7b	8b	9b	10b	11b	12b	13b	14b	15b	16b
count	10	15	21	28	36	45	55	66	78	91	105

There are 32 branches in **6-bit floor**, 10 for **6b morfs** and 22 as nodes for branches in **7-bit floor**, so it means 22×2=44 branches in **7-bit floor**. From these 44 branches are 15 for **7b morfs**, and 29 as nodes for **8-bit floor**. 29×2=58 branches in **8-bit floor** therefrom 21 for **8b morfs** and 37 as nodes for **9-bit floor**.

6b
32-10=**22** **7b**
22×2= 44-15=**29** **8b**
29×2= 58-21=**37** **9b**
37×2= 74-28=**46** **10b**
46×2= 92-36=**56** **11b**
56×2= 112-45=**67** **12b**
67×2= 134-55=**79** **13b**
79×2= 158-66=**92** **14b**
92×2= 184-78=**106**

Probably the count of free branches in N^{th} floor = $1+\frac{N\cdot(N+1)}{2}$

There is enough space in every floor for the same-bits morfs like the floor-bits are, and there are always some free branches in last floor, so it means the morfs from last floor could be moved to lower floor so it truncates their lengths. And there will not all types of long-morfs in compressed files, so that the truncation of some long-morfs will bigger than 1 bit :-).

But be careful to add there extra "special character" like Null character in Vitter algorithm, or ofr example character means the repetition of previous character → if you add one furthermore character then there will not be enough of space in some next floors a some morfs would be lengthned.

Probably it will be favorlable to use only one tree from the start, but if some morf should be elongated, then it will be better to use two presented trees. And for example write to the Head for example the position from that the two trees are used.

It seems well if the long morfs will shorter, but will the information for this in Head also shorter? *Is it possible to solve it with some deep philosophies or huge mathematical theoretizations? I think 'no', but surely 'yes' with practical programming :-).*

How to count the morfs "quickly"

The shorter morfs can be the indexes in static array, like 1-2-2 for example in C: **arr[10011]++**. And longer morfs count in dynamic array like „associative array of array of array" with indexes as numbers from the morf. The boundary between short/long morfs can be for example 24 bits, so count the morfs longer than 24 bits to the dynamic array.

The **24bits** array **×4**bytes for longint variables takes c. 64MB RAM. It is jammy acceptable. Longer static array for example 30bit*4bytes=4GB RAM. It could be also acceptable for example if the normal computer has ≥16GB RAM.

The count for all morfs in group with maximum length (N) is: **Count= N·(N-1)·(N-2)/6**

The interest is, there will be only 2024 items in 24b array, or only 4060 items in 30b array, but direct addressing in memory isn't good under OS with other programs :-(.

Of course if the morf starts with zeroes, then there can be a shifting to next variable-with-ones and a negation.

111	1	1	1	1	1	1	1	1	←	**0**	**1**	**1**	**0**	**0**	…
111	1	1	1	**0**	**1**	**1**	**0**	**0**	Shift						
000	0	0	0	**1**	**0**	**0**	**1**	**1**	Neg						

How to write the type of morfs

It is possible to sort the morfs about their indexes, but there will start to absent some morf in some N-bit group. So to write only one number – the count of morf to this place.

> *For example I have all 3b morfs, all 4b morfs, but there **isn't** one of 5b morfs. The indexes are: 101, 1001, 1011, 1101, 10001, 10011, 10111, **11001**, 11011, 11101. So I can write 7 and it means there are all first seven morfs.*

For some next morfs it is possible to use the sequece of bits if the morf is or no.

> *For previous example the next sequence is: **0**11*

But for some very-long morfs probably be shorter that these morfs will be written in some classical numerical form like 1234-15-867 etc., prospectively there could be written their offsets from previous morf. But for optimal Head it seems it need some test for the optimal possibility for a concrete case.

Final

There could be for example 2-5 sequences of bits in output file. Standard Head and Body; or Head with Count of morfs, Head with offsets&positions for morfs, bits for type of tree (small/big tree), and the data bits for small tree and the bits for big tree. Maybe some of these sequences could be compressed again with standard methods, maybe also by recursion. But maybe, I really don't know.

But maybe you also discovered yet, that there is a big space for an ecryption. It is possible to mix the bits from 5 sequences according to some keys; counterhange the same-length morfs in the same floor by some key or makes a different trees *(by Vitter's ordering, FGK ordering etc.)*; put to the final file some random bits according to some other key; and of course some bits in final file XOR etc. by some next key.

I'm not a supporter of concealment, but with these possibilities it is possible to make really very strong and quality encryption, maybe without any chance to enucleate it with AI, because there could be so many combinations and so many „sham herons“ with the random and XORed bits.

> *When I studied a 'computer systems' at high school, we had also a manual workshops. There were a lathes in workshop and we turned a disks for a dumbells to school gymnasia. The lathe published a strange sounds during a turning. And after it, we had a programming, so we had said: „lathe“ to the HDD that published similar sounds for many read/write operations.*
>
> *The SSD and RAM are silent, but maybe there could be inaudible „sounds“ like pant or screams etc. from surprised metamorphoses, recursions, „sham herons“, XORs etc. in the sequence of bits :-). But it could be also a fact, that some “IQ”kepls [kepl = computer, AI; **e** in k**e**pl is speaked as '**e**' like in 't**e**n'] could get from the welter of a bits for example something like: “the meaning of universe is equal to the red flower that changed in purple car and flew to the heaven as submarine”, simply something like AI on LSD :-).*

*There is a nice sentence in Czech that describes the inner “state of mind”: “**Nechce se mi do toho, protože + …**”, but the exact translation to English with respecting the thinking in English really isn't easy. The typical translations are: **I don't want to do it**; **I'm not interested in it**; **I don't care**. Now I tried to solve it by this way :-):*

***The most likely (/most preferably) I would not want to do it, because + your true reasons**.*

I don't know if these improvements can make better or repeated compression. I haven't a mood, willpower and knowledges, and also I don't feel that I ought to program it, but the strongest reason is the missing knowledges. And I know it would take a long time than I would learn a needful knowledges – I've made about 10 extensions for LibreOffice and I've been working on the most comprehensive one for about 6 years. And I don't want to stop this work only to learn for example C/Asm.

I was “discovering” this algorithm in last 12 years. I got some ideas about one in last few days. So if the programming of algorithm depends on me, it can calmly wait for some next years :-).

I tried to program some of it for the last time in 2015. It was really with a big ardour – I sit 4 days at old small 11“ laptop and 5th day I got a big vertigos, so I had to call an ambulance. Overstretched neck rachis with problems to these days (2022) :-(. So I can say: „I knew on my own body how not to do a programming“ :-).

The example with the Counts and Positions

for example sequence of bits: 0101010010110101101011001010011011010110101001010110001001010100101 01

severally:	010	101	0010	1101	0110	1011	0010	10011	0110	1011	010	1001	010	110001	0010	101	0010	101
morfs:	1-1-1	1-1-1	2-1-1	2-1-1	1-2-1	1-1-2	2-1-1	1-2-2	1-2-1	1-1-2	1-1-1	1-2-1	1-1-1	2-3-1	2-1-1	1-1-1	2-1-1	1-1-1

for a simplification, the morfs are substitued by the letters; **start position**, **singles/twins**, **end positions**, **deducibles**

1-1-1 **A**
2-1-1 **B**
1-2-1 **C**
1-1-2 **D**
1-2-2 **E**
2-3-1 **F**

	A	**A**	**B**	**B**	**C**	**D**	**B**	**E**	**C**	**D**	**A**	**C**	**A**	**F**	**B**	**A**	**B**	**A**
positions:	**1**	**2**	**3**	*4*	**5**	**6**	*7*	**8**	*9*	**10**	*11*	**12**	*13*	**14**	*15*	**16**	**17**	**18**

AABBCDBECDACAFBABA

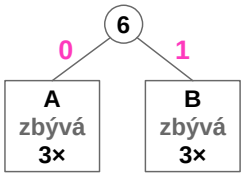
STATISTICS

character	COUNT	start position	end position
A	3	**1**	**18**
B	3	**3**	**17**
C	1	**5**	**12**
D		**6**	**10**
E		**8**	
F		**14**	

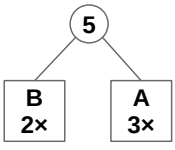
The **COUNT** means the count of characters included to the tree, there aren't **start** nor **end** positions, nor **deducibles** occurences.

There is known from Head the 1st character is **A** and next new one (**B**) starts on 3rd position, so 2nd charcter have to be **A**.
→ **AAB**

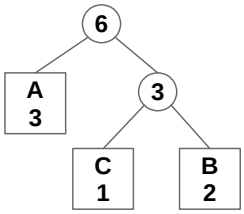
There are more than twice both characters and there isn't the penultimate occurrence of one of these morfs in these three positions, so it is needful to create the tree with these characters, because next new character (**C**) is in 5th position and the character at 4th position have to be **A** or **B**. The Count of characters in three means how many characters **remain** to the penultimate position, because end position is known from a Head.



4th character is **B** , so the tree must be recounted (for example the trees to left are **0**, to right **1**)
→ **AAB**B **:** 0 [**characters :** bits report]



5th character is **C** and there is more than 2×, so it is needful to add it to the tree *(there is a Vitter order for the tree example)*
→ **AABBC : 0**

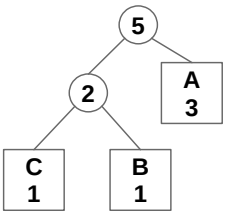


Next character is the 'position character' **D**, his Count is 2× so it measn in needn't be added to the tree.

→ **AABBCD : 0**

Next is the **B**, write and recalculate the tree

→ **AABBCDB : 011**

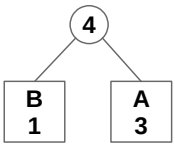


Single **E** isn't in the tree so only write

→ **AABBCDBE : 011**

Next is writing the **C** and decrease its Count in the tree → it will be 0, so remove it from the tree *(the last occurrence is known from the last position from Head)*

→ **AABBCDBEC : 01100**

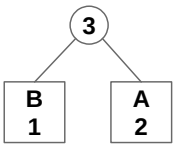


10th character and the end position of **D**, so only to write

→ **AABBCDBECD : 01100**

Next is **A** with recalculating the Count in the tree

→ **AABBCDBECDA : 011001**

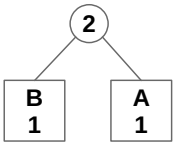


12th character is end position for the **C**

→ **AABBCDBECDAC : 011001**

Next is **A**, write and recalculate the tree

→ **AABBCDBECDACA : 0110011**



Single **F** so without the edit of tree

→ **AABBCDBECDACAF : 0110011**

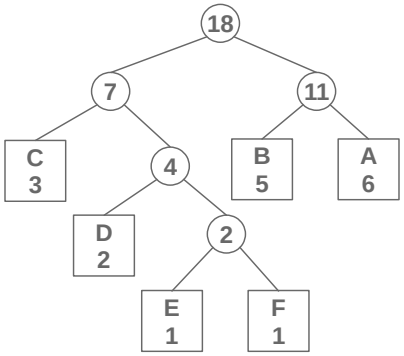
Next is **B**, write and remove from the tree (it was a penultimate position for the B)

→ **AABBCDBECDACAFB : 01100110**

There is only one leaf **A** in the tree, so it will be at the position that isn't the end position of last **A** nor last **B**, so thre aren't next more bits in bits-report.

→ **AABBCDBECDACAFB + ABA : 01100110 length 8bits**

For example, the classical Huffman tree (with the Vitter's ordering):



and the sequence of bits from one:

AABBCDBECDACAFBABA ≈ ***11111010000101010110000101110011011110111011*** *length 42 bits*

Of course it seems better the final compression with only the lenght=8bits than standard result with 42bits, but don't forget it is only short example and the Head is bigger for the Dynamic tree.

The decompression is with the same logic, reconfigure the tree and then read the bits from the compressed sequence.

> *I got the idea for removing the characters from the tree when I learned the Vitter algorithm and I tried to make a tree for the morfs on the Russian Computer! (= paper & pencil) :-) :-). When I redrawed the tree, so maybe after 3rd redrawing I said "Fuck, why to always strain with the redrawing of characters that has only one occurrence?!" a I started to try it with the removing of characters from the tree :-).*

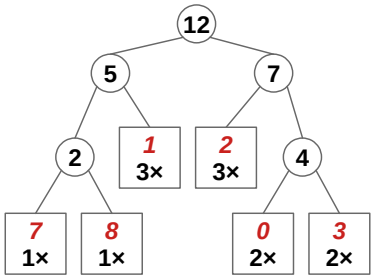
Example with the offsets (distances)

> *The initial thoery is to use it as **completely offseted** – don't look the Counts of characters, but only the offets among the characters like in the dictionary methods.*

0 8 1 2 1
AABBCDBECDACAFBABA

char	OFFSETS				
A	0	8	1	2	1
B	0	2	7	1	
C	3	2			
D	3				
E					
F					

> *Initial tree could look like this and it would be downsized after every offset, so the tree would be downsized for all time.*



> *So it seems, it would be very large tree at start so there would be very many offsets which would mean that some offsets would have lot of bits, and I don't believe this large start-tree would make a good compressions with the donwsizing to its end disassembly → and perhaps it would be a success if it remained at least the same size after compression and not larger.*

The detection of morfs and offsets should be feasible in one pass, and in the 2nd pass, the writting with tree dowsizing. However, it would probably not be fast :-(.

PARTIAL DISTANCES (OFFSETS)

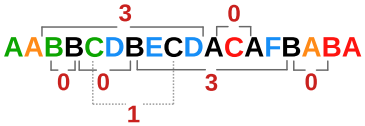
The characters that are known from the positions&deducibles need't be included to the offsets, which shold mean the smaller number of the offsets. In the example, only the **black** characters are counted in the offsets.

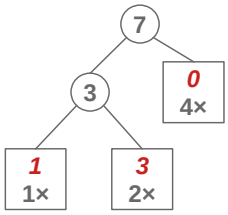
Takže kdyby byly zapamatovaná třeba jedna vzdálenost pro znak **A** (aby **A** bylo zahrnuto do stromu až od pozice **A**), měla by hodnotu **3**, neboť neodvoditelné by mezi A a **A** by byly jen znaky **BBC**. Pro **B** a zahrnutí ho do stromu od **B** by platila vzdálenost **0**; a **C** by díky vzdálenosti **1** vůbec nebylo ve stromu.

So if, say, one offset were remembered for character **A** (so that **A** would be included in the tree only from position **A**), it would have a value of **3**, because between A and **A** there are only **BBC** characters would be non-derivable (**BBC**). For **B** and including it to the tree from **B** (and not from B), an offset 0 would apply; and **C** would not be in the tree, because it has a distance and no next occurrence.

So for example, the character **C** *has 3 occurrences – 1st given by start position (C); 2nd given by offset (C); and 3rd given by end position (C), so it is needn't put it to the tree.*

In this small example, a ***complete distancing*** *with the omission of deducibled characters would indeed mean a much smaller start tree, but I really suspect that in reality it would end up being described as "better than nothing, but still good-for-nothing".*





But how many distances to use to scrimp the space the characters will not be in the trees? And try to apply, say, two trees to write offsets? Or add more options with different distances taken for different characters, maybe even from different character positions? *E.g. so that for A there would be 3 distances, but only from the 5th occurrence of A; for B maybe 2 distances from 1st non-deducible occurrence, etc.?*

The tree could be shortened by the distances even if they were taken from behind, so e.g. at one distance from behind the character would be removed from the tree at the penultimate occurrence.

Variant se na to dá možná vymyslet ještě více, a asi nejlepší by bylo zjistit všechny vzdálenosti při prvním průchodu (ovšem zřejmě na úkor rychlosti zpracování) a pro kompresi pak dělat nějaké různé porovnávací výpočty které by zjišťovaly která varianta by ušetřila více místa → což by možná znamenalo nejen větší pomalost, ale možná i různé zaplácávání RAMky všelijakými dočasnými pokusnými řadami bitů. *Ovšem různí paranojci, vyvolení, poznamenaní, agentící i údajně šéfující by s každou další kombinací mohli spatřovat další a další možnost pro šifrování :-).*

There might be even more variations on this, and probably the best would be to get all the distances on the first pass (at the expense of processing speed, of course) and then do some various comparison calculations for compression to see which variation would save more space → which might mean not only more slowness, but perhaps also various RAM cluttering with some sorts of temporary experimental bit strings. *Of course, various paranoids, the chosen, the jinxed, the agents and the supposedly bosses might see more and more possibilities for encryption with each new combination :-).*

Thus, theoretically, it is possible to make a lot of combinations, but it would be decent to at least wish the potential programmer not to feel like some AI on drugs :-).

Last 2 pages were very hard to translate for me, so I used DeepL. If you don't understand some things from the English document, firstly please try to translate the Czech version in the e-translators, maybe you get a better translation than this English version from me :-).