Morf kód

Kamil Landa

vytvořeno **LibreOffice** http://libreoffice.org 6/2022

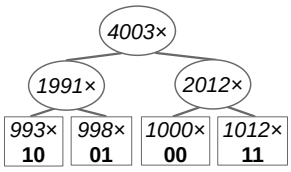
Morf kód je návrh dvouprůchodového kompresního algoritmu inspirovaný Huffmanovým&Vitterovým kódováním. Strom však nemusí být statický *(Huffman)* ani pouze zvětšovaný *(Vitter)*, ale může být dynamicky zvětšo/zmenšo-ván.

Česká verze je trochu pestřejší než anglická kterou jsem překládal sám, neboť anglicky moc dobře neumím a stejně tak nehodlám češtinu nijak primitivizovat. Dopředu upozorňuji, že nevím jestli algoritmus může vylepšit kompresi :-(. Buďte prosím laskaví ke mně i případně mým chybám :-).

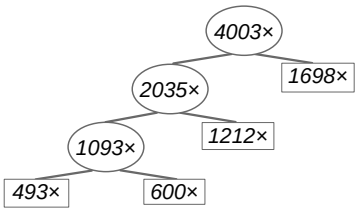
Algoritmus využívá jiné rozdělení řady bitů než standartně na bajty, dvojbajty, čtyřbajty atd. Znaky s pevnou šířkou mají v již komprimovaných souborech *(jako AVI, ZIP atd.)* v podstatě stejný počet výskytů, tudíž strom by byl velmi rychle zvětšen na plnou velikost hned zpočátku, a efekt zmenšování stromu byl velmi malý v podstatě až na konci.

> *Příklad s "bajtovými" znaky, pro zjednodušení pouze s dvoubitovými znaky. Jejich počet je ve zkomprimovaných souborech v podstatě stejný, třeba následující: **00**: 1000×, **01**: 998×, **10**: 993×, **11**: 1012×; a Huffmanův strom pak symetrický. Zkusil jsem najít znaky s nerovnoměrným zastoupením a dostat tak asymetrický strom, v němž by možná mohla být větší šance že zvětšo-zmenšování vyskrblí více bitů.*

symetrický strom



asymetrický strom

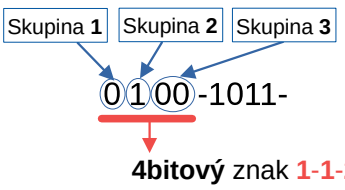


Znak je dán trojpočtem bitů ze tří skupin stejnobitů; skupina je prostě jen počet stejných bitů za sebou. *Tato definice toho asi moc neobjasní, ale na jednodušší jsem nepřišel; věřím však že příklad to vyjasní.*

Např. 5-ti bajtové slovo **KaMiL**:

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

celá sekvence 0100101101100001010011010110100101001100



Znak jsou 3 počty tří stejno-bitových skupin:

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

Skupina 1
Skupina 2
Skupina 3

Jelikož je tam pravidelné střídání nul a jedniček, tak **jedničky-nuly-jedničky** jsou stejné znaky jako **nuly-jedničky-nuly**. Např. **010** je stejné jako **101** → obé je znak **1-1-1**. *Je potřeba si zapamatovat akorát první bit celé řady a pak pouze střídat **jedničky-nuly-jedničky/nuly-jedničky-nuly** pro liché a sudé znaky :-).*

Ve zkomprimovaných souborech je mnoho druhů těchto znaků, zkoušel jsem to na 200MB Avíčku a párMB Zipech, přičemž v tom Avíčku byla asi 12 tisíc druhů znaků; samozřejmě všechny krátké (**3bitové**: **1-1-1**; **4bitové**: **1-1-2**, **1-2-1**, **2-1-1**; **5bitové**: **1-1-3**, **1-3-1**, **3-1-1**, **1-2-2**, **2-1-2**, **2-2-1**; **6bitové** **1-1-4**, …), ale také velmi dlouhé jako 12-307-1251, 125-345-57, 1012-2-708 atd. → ale tyto velmi dlouhé znaky tam byly např. jen jednou nebo dvakrát. A není potřeba tyto dlouhé znaky vkládat do strom-u(ů), jestliže jsou známé pozice jejich prvního a posledního výskytu.

Vzorec pro počet druhů N-bitových znaků je:

$$\textbf{Počet Nbitových} = \frac{(N-2)\cdot(N-1)}{2}$$

3bitový **1**×, **4bitové** **3**, **5bitové** **6**, **6bitové** **10**, **7bitové** **15**, **8bitové** **21** atd.

Dokument jsem nejprve psal v angličtině a pojmenoval tyto znaky 'morphy', ale v češtině je snažší: **morfy** :-).

> *Když jsem to kdysi poprvé programoval, tak hlavní problém byl jak rychle dostat morfy s jejich počty. Znaky mají různou bitovou délku a já používal (v Asm vkládaném do C) instrukce Posunu do příznakového bitu CF a skoky JC/JNC pro počítání bitů, protože jsem znal pouze x86-Asm pro PC 286/386 ze střední školy z konce minulého století a nebyl jsem schopen prozkoumat novější intrukce pro x86-64, bylo to na mou angličtinu moc kompli-kované. Možná znáš rychlejší postup.*

Zvětšozmenšování stromu: na rozdíl od Vitterova stromu zde není Nulový znak pro neznámý morf → tudíž je možné, že někdy bude muset být přestaven úplně celý strom, a stejně tak při zmenšování. A vypadá to, že budou potřeba alespoň dva stromy *(jeden malý a mnohem statičtější než dynamičtější, a jeden velký a mnohem dynamičtější než statičtější)*, neboť když je použit jen jeden strom, tak jeden z 4bitových znaků se dostane již na 5bitové patro, některý z 5bitových na 6bitové patro atd. → *je to stejná situace jako v Huffmanovi nebo Vitterovi, některé znaky by byly zkráceny ale některé prodlouženy, avšak prodlužování morfů není vítané.*

Jsou-li známy počáteční/koncové pozice morfů, pak poslední morfy nemusí být ve stromu, tudíž znak je odstraněn ze stromu při jeho předposledním výskytu. A další možností je zapsat do Hlavičky ještě více pozic, např. druhé výskyty morfů → v tom případě by morfy byly ve stromu až od jejich druhého výskytu.

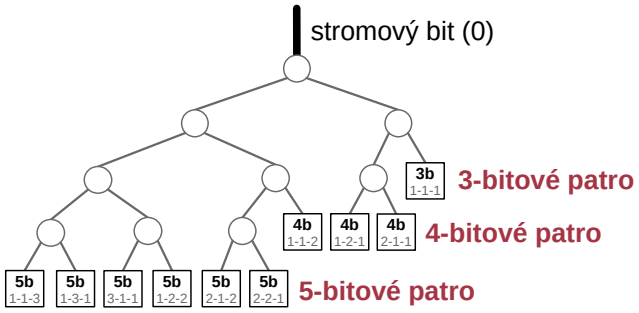
Pozice&vzdálenosti *(jako ve slovníkových metodách)* mohou být atraktivní, ale rébusem je, jak pro tyto informace vytvořit co nejmenší Hlavičku :-), protože dlouhé morfy mohou zabrat v hlavičce docela dost bajtů jen pro zápis jejich jmen → 1245-8-147, 12-1986-13, 1878-1936-254 atd., a tyto dlouhé morfy jsou velmi nepravidelné :-).

Využít vzdálenosti *(offsety)* je žádoucí neboť to zmenšuje strom, ovšem pokud bude vzdálenostní informace v Hlavičce kratší než zevzdálenostněný morf. A jak zapsat pozice&vzdálenosti do Hlavičky? Použít alespoň Huffmana, nebo zkusit rekurzi s Morf kódem?

Dva stromy počtů

Následující úvaha je pro statické stromy, pro dynamické je to nepředvídatelné.

Je-li vzorec pro **Počet druhů N-bitových znaků** dobře, tak to vypadá, že se dvěma stromy počtů některé morfy [primárně ty krátké] zůstanou stejně dlouhé, avšak některé *dlouhé budou buď zkrácené, nebo alespoň neprodlužované.*

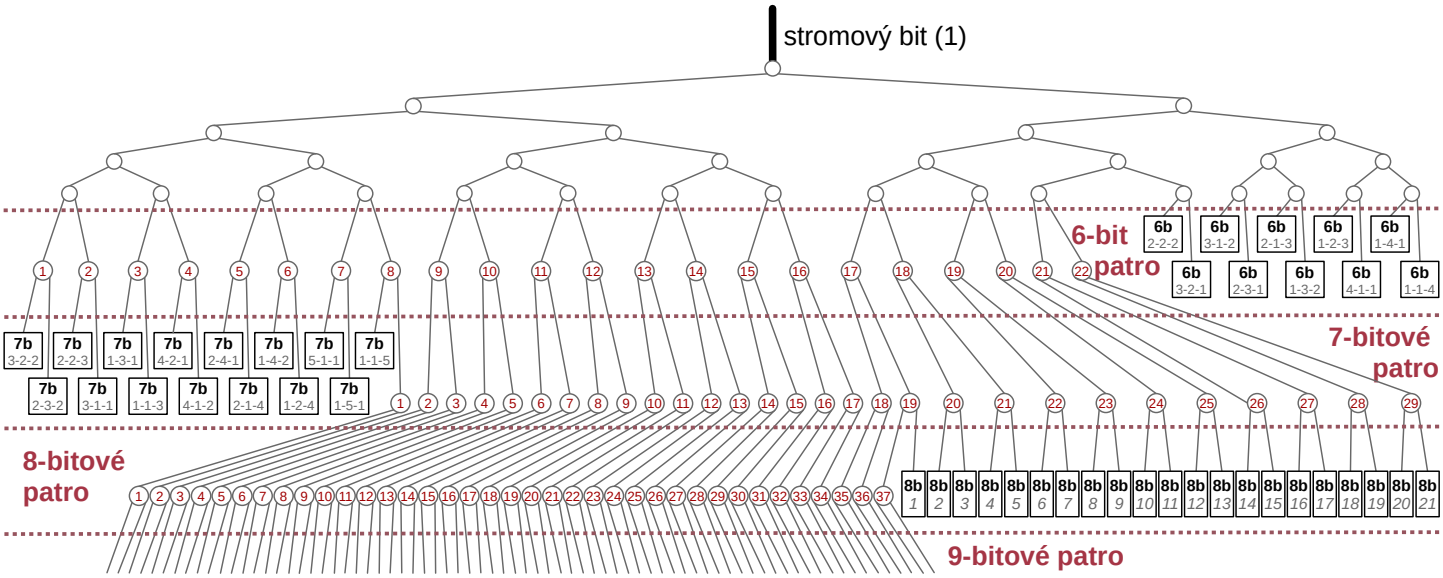


1. strom: **tří**, **čtyř** a **pětibitové** morfy mají zachovánu bitovou délku. Ve stromě sice není místo pro další morfy, ale je fakt, že tyto morfy jsou ve zkomprimovaných souborech nejčastěji.

2. strom je velký strom v němž jsou ≥**6bitové** morfy bez samotářů/dvojčat *(ty jsou dány jejich počátečními/koncovými pozicemi)*. Idea je neprodlužovat morfy, tudíž **6bitové** morfy musí být maximálně v **6-bitovém patře**, **7bitové** v **7-bitovém patře** atd., což vypadá že strom umožňuje.

Byl problém udělat tak velký strom (místa na stránce není tolik) tudíž na obrázku není strom podle Vitterova uspořádání uzlů/větví.

A příliš velký strom bude zřejmě nevýhoda pro rychlost zpracování :-(.



Nbitové morfy	6b	7b	8b	9b	10b	11b	12b	13b	14b	15b	16b
počet	10	15	21	28	36	45	55	66	78	91	105

V **6-bitovém patře** je 32 větví, **10** pro **6bitové morfy** a **22** jako uzle pro větve v **7-bitovém patře**, což znamená 22×2=44 větví v **7-bitovém patře**. Z těchto 44 větví je **15** pro **7bitové morfy**, a **29** jako uzly pro **8-bitové patro**. 29×2=58 větví v **8-bitovém patru** přičemž **21** pro **8bitové morfy** a **37** jako uzly pro **9-bitové patro**.

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**

Nejspíš vzoreček pro počet volných větví v **N-bitovém patře** = $1+\frac{N\cdot(N+1)}{2}$

V každém patře je dost místa pro stejně-bitové morfy jako má patro, a v posledním patře navíc zbydou některé prázdné větve, což znamená že morfy z posledního patra mohou být přesunuty do nižšího což zkrátí jejich délku. A ve zkomprimovaných souborech nejsou všechny druhy dlouhých morfů, tudíž zkrácení některých bude větší než jen o bit :-).

Pozor však na to kdyby se do stromu měl přidat nějaký „speciální znak“ navíc. Např. něco jako Null znak ve Vitterově algoritmu nebo třeba znak značící opakování předchozího znaku → jeden znak navíc a během pár pater dojde k tomu, že nebude dostatek místa a některé morfy by se již začaly prodlužovat.

Zpočátku bude zřejmě výhodné používat jen jeden strom, ale jakmile by nějaký morf měl být ve stromu prodloužen, bude asi lepší přejít již na dva uvedené stromy. A např. zapsat do Hlavičky pozici odkdy jsou použité dva stromy.

Vypadá dobře když mohou být nějaké morfy kratší, ale bude informace pro zkracování v Hlavičce také kratší? *Je možné to vyřešit nějakým hlubokým filozofováním nebo mohutným matematickým teoretizováním? Myslím že ne; ale určitě 'ano' s praktickým programováním.*

Jak spočítat morfy „rychle“

Kratší morfy mohou být indexy ve statickém poli, např. 1-2-2 pro příklad v Céčku: **arr[10011]**++. A delší morfy mohou být počítány v dynamickém poli jako „asociativní pole polí polí“ s indexy z čísel morfu. Hranice mezi krátkými/dlouhými morfy může být např. 24 bitů, takže morfy delší než 24 bitů počítat v dynamickém poli.

24bitové pole **×4**bajty longintové proměnné zaberou cca 64MB RAM. To je snadno akceptovatelné. Delší statické pole např. 30bitové*4bajty=4GB RAM. Což může být také akceptovatelné, má-li běžný počítač alespoň 16GB RAM.

Počet všech morfů ve skupině s maximální bitovou délkou (N) je: **počet= N·(N-1)·(N-2)/6**

Zajímavostí je, že ve 24 bitovém poli bude pouze 2024 položek, nebo 4060 položek v 30 bitovém poli, ale přímé adresování paměti není moc dobré pod operačním systémem s jinými programy :-(.

Samozřejmě když morf začíná nulami, tak může být posunut do další vyjedničkované proměnné a znegován.

111	1	1	1	1	1	1	1	1	←	**0**	**1**	**1**	**0**	**0**	…
111	1	1	1	**0**	**1**	**1**	**0**	**0**	posun						
000	0	0	0	**1**	**0**	**0**	**1**	**1**	negace						

Jak zapsat druhy morfů

Je možné seřadit morfy dle jejich indexů, ale někde začne chybět nějaký morf v nějaké N-bitové skupině. Takže zapsat jedno číslo – počet morfů do tohoto místa.

> *Například mám všechny 3b morfy, všechny 4b morfy, ale **není** tam jeden 5bitový morf. Indexy jsou: 101, 1001, 1011, 1101, 10001, 10011, 10111, **11001**, 11011, 11101. Tudíž lze zapsat 7 což znamená že je tam prvních sedm morfů.*

Pro další morfy je možné použít sekvenci bitů zda-li morf je nebo ne.

> *Pro předchozí příklad je následná sekvence: **0**11*

Avšak pro velmi dlouhé morfy bude zřejmě kratší, když budou zapsané v nějaké klasické číselné formě jako 1234-15-867 atd., případně když budou zapsané nějaké jejich vzdálenosti od předchozího morfu. Ale pro optimální Hlavičku asi bude potřeba prostě zkusit která z možností bude v konkrétním případě zrovna vyhovovat.

Finále

Ve výstupním souboru může být např. 2-5 sekvencí bitů. Standartně Hlavička a Tělo; nebo Hlavička s počty morfů, Hlavička s pozicemi&vzdálenostmi; bity pro typ stromu (malý/velký strom), a datové bity pro malý strom a pro velký. Možná některé z těchto sekvencí by mohly být znovu zkomprimovány standartními metodami, nebo možná rekurzí.

Avšak možná jsi také již objevil, že je zde velký prostor pro šifrování. Je možné zmixovat bity z 5 sekvencí dle nějakého klíče; zaměňovat stejně dlouhé morfy v tom samém patře podle dalšího klíče nebo dělat různé stromy *(podle Vitterova nebo FGK uspořádání atd.)*; vložit do výstupního souboru nějaké náhodné bity podle nějakého dalšího klíče; střídat nějak bity pro směr větví pro různá patra; prohazovat větve a uzly se stejnými počty; a samozřejmě nějaké bity ve finále třeba zeXORovat podle dalšího klíče.

Nejsem podporovatel zatajování, ale s těmito možnostmi je možné vytvořit opravdu velmi silné a kvalitní šifrování, možná i bez šance to rozluštit s umělými inteligencemi, protože tam může být velmi mnoho kombinací a velmi mnoho „falešných volavek“ s náhodnými nebo zeXORovanými bity.

> *Když jsem studoval 'počítačové systémy' na střední škole, měli jsme i dílny. Byly tam soustruhy a soustružili jsme kotouče na činky do školní tělocvičny. Soustruhy vydávaly během soustružení divné zvuky. A poté jsme měli programování, takže jsem říkávali 'soustruh' pevnému disku HDD když vydával soustruhu podobné zvuky při mnoha čtení/zápisových operacích.*
>
> *SSD disky a RAMky jsou tiché, ale možná by tam mohly být neslyšitelné „zvuky“ jako funění či výkřiky z překvapivých metamorfóz, rekurzí, „falešných volavek“, XORů atd. v sekvenci bitů :-). Ale může být i fakt, že některé ÍKVÉkeply by z toho mohly dostat třeba to, „že smysl vesmíru je roven té červené kytičce jež se změnila ve fialové auto a odlétla do nebe jako ponorka“, prostě něco jako umělá inteligence na LSD :-).*

*V češtině je pěkná věta: “**Nechce se mi do toho, protože + …**”, ale přeložit jí do angličtiny a respektovat přitom myšlení v angličtině vůbec není snadné. Typické překlady do angličtiny jsou: **I don't want to do it = nechci to udělat**; **I'm not interested in it = dosl.: nejsem v tom zainteresován** (nezajímám se o to); **I don't care. = dosl.: nestarám se** (nezajímá mě to!) **– ale tohle je prý v En již silné odsouzení.** Já to zatím vyřešil takto: **Nejraději bych to nechtěl udělat, protože + uvést pravdivé důvody = The most likely (/most preferably) I would not want to do it, because + your true reasons.***

Nevím jestli tato vylepšení mohou udělat lepší nebo opakovatelnou kompresi. Nechce se mi to programovat, nemám na to náladu, vůli ani znalosti, a ani necítím že bych nějak měl, přičemž nejsilnější důvod proč se mi nechce je ten, že prostě v Céčku/Asm moc neumím a bylo by to na moc dlouho než bych se potřebné naučil – mám rozdělaných asi 10 rozšíření pro LibreOffice přičemž na nejobsáhlejším dělám už asi 6 let. A nechci tyto projekty na několik měsíců přerušit jen z důvodu že bych se učil C/Asm. Algoritmus jsem „objevoval“ během uplynulých 12-ti let a poslední nápady jsem dostal v několika posledních dnech. Takže je-li programování algoritmu na mně, tak to klidně může ještě nějaký rok počkat :-).

Naposledy jsem něco zkoušel naprogramovat někdy v roce 2015. A bylo to se skutečně velkým zápalem. 4 dny jsem proseděl u malého 11-ti palcového laptopu a pátý den jsem dostal tak velké závratě, že jsem si musel volat sanitku. Natažená krční páteř s problémy dodnes (2022) :-(. Čili mohu říkat: „Na vlastním těle jsem poznal jak programování nedělat“ :-).

Ukázka s počty a pozicemi

např. řada bitů: 010101001011010110101100101001101101011010100101011000100101010010101

jednotlivě:	010	101	0010	1101	0110	1011	0010	10011	0110	1011	010	1001	010	110001	0010	101	0010	101
morfy:	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

pro zjednodušení jsou morfy nahrazeny písmeny; **počáteční pozice**, **samotáři/dvojčata**, **koncové pozice**, **odvoditelné**

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**
pozice:	**1**	**2**	**3**	*4*	**5**	**6**	*7*	**8**	*9*	**10**	*11*	**12**	*13*	**14**	*15*	**16**	**17**	**18**

AABBCDBECDACAFBABA

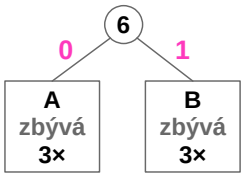
STATISTIKA

znak	POČET	počáteční pozice	koncové pozice
A	3	**1**	**18**
B	3	**3**	**17**
C	1	**5**	**12**
D		**6**	**10**
E		**8**	
F		**14**	

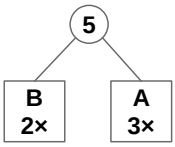
POČET udává počet znaků zahrnutých do stromu, **nepočítají se** do něj **počáteční** ani **koncové** pozice, ani logicky **odvoditelné** výskyty.

Z hlavičky je jasné že první znak je **A** a další nový znak (**B**) je až na třetí pozici, tudíž druhý znak musí být též **A**.
→ **AAB**

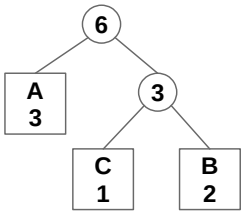
Oba znaky jsou však obsaženy více než dvakrát a v oněch třech pozicích nedošlo na předposlední výskyt žádného z nich, je tedy potřeba s nimi založit strom, neboť další nový znak (**C**) je až na páté pozici a čtvrtý znak tedy musí být buď **A** nebo **B**. Ve stromu je udáno kolik znaků **zbývá** bez poslední pozice (tedy do předposlední), neboť koncová pozice je známa z tabulky.



Čtvrtý znak je **B** přičemž dojde na úpravu počtu **B** ve stromu (větve směrem vlevo jsou třeba **0**, směrem vpravo **1**)
→ **AAB**B **: 0** [**znakový : bitový** zápis]



Pátý znak je **C** a jelikož je víc než 2×, je třeba ho přidat do stromu *(v ukázce je strom uspořádán podle Vittera)*
→ **AABBC : 0**

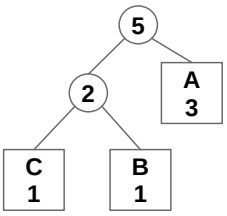


Další znak je opět poziční a to **D**, je pouze 2× tudíž není potřeba ho zahrnovat do stromu

→ **AABBCD : 0**

Následuje **B**, zapsat a upravit strom

→ **AABBCDB : 011**

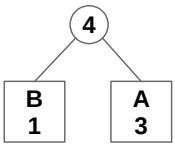


Poziční **E** není ve stromu, takže pouze zapsat

→ **AABBCDBE : 011**

Dále zapsat **C** a snížit jeho počet ve stromu což bude nula takže jej odstranit ze stromu *(jeho poslední výskyt bude dán koncovou pozicí)*

→ **AABBCDBEC : 01100**

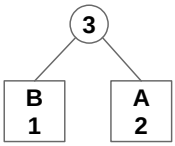


Jsem na 10. znaku o němž je z Hlavičky známo že jde o koncovou pozici znaku **D**, tudíž jen zapsat

→ **AABBCDBECD : 01100**

Následuje znak A s upravením počtu ve stromu

→ **AABBCDBECDA : 011001**

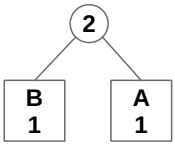


12. znak je koncovou pozicí **C**

→ **AABBCDBECDAC : 011001**

Další je **A**, zapsat a upravit strom

→ **AABBCDBECDACA : 0110011**



Následuje samotář **F** čili bez úpravy stromu

→ **AABBCDBECDACAF : 0110011**

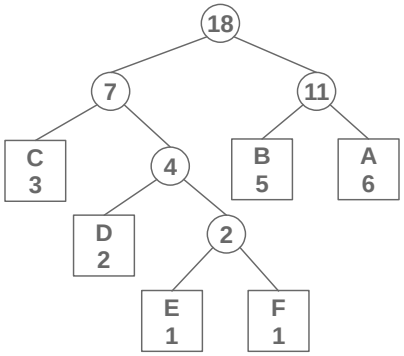
Další je **B**, čili zapsat a vyhodit ze stromu, šlo o předposlední pozici

→ **AABBCDBECDACAFB : 01100110**

Ve stromu by zbýval jen list **A**, čili bude na místě které neodpovídá koncové pozici posledního **A** ani **B**, do bitové sekvence již není potřeba nic přidávat.

→ **AABBCDBECDACAFB + ABA : 01100110 délka 8bitů**

Pro ukázku, Huffmanův strom (uspořádán dle Vittera) by vypadal takto:



a bitový zápis by byl:

AABBCDBECDACAFBABA ≈ *11110100001010110000101100111011110111011* délka 42 bitů

Samozřejmě že výsledná komprese o délce 8 bitů je mnohem přívětivější než standartní statický výsledek 42 bitů, ale je třeba nezapomínat na to, že jde jen o krátkou ukázku a že Hlavička je pro dynamický strom větší.

Dekomprese se provádí se stejnou logikou, nejdříve přenastavit strom a pak odečítat datové bity ze zkomprimované řady.

Že je možné odstraňovat znaky ze stromu mě napadlo, když jsem se učil Vitterův algoritmus a zkoušel dělat strom pro morfy na ruském počítači (= papír a tužka). Při překreslování znaků před uzly (aby platilo že znak se stejným počtem je před uzlem se stejným počtem) mě snad po třetím překreslení napadlo „kurva proč bych se měl pořád srát s těma znakama který tam jsou jen jednou!“ a začal to zkoušet s odstraňováním znaků :-).

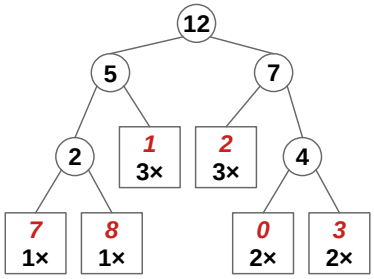
Ukázka i se vzdálenostmi

*Teorie ze které to vychází je **komplet zevzdálenostněné** – nebrat v úvahu počty znaků, ale jen vzdálenosti stejných znaků od sebe, čili Vzdálenost znamená kolik jiných znaků se nachází mezi dvěma stejnými znaky.*

0 8 1 2 1
AABBCDBECDACAFBABA

znak	VZDÁLENOSTI				
A	0	8	1	2	1
B	0	2	7	1	
C	3	2			
D	3				
E					
F					

Počáteční strom by vypadal třeba takto a samozřejmě s každou vzdáleností by se snížil počet u oné vzdálenosti a strom se přestavěl, čímž by se v podstatě neustále zmenšoval.



V praxi by to zřejmě bylo tak, že strom by byl zpočátku hrozně velký, což by znamenalo že nějaké vzdálenosti by zabíraly strašně moc bitů, přičemž nějak nevěřím tomu, že by se toto počáteční roztažení stromu do jeho rozebrání alespoň vyrovnalo → a možná by bylo úspěchem, kdyby to po kompresi zůstalo alespoň stejně velké a nikoliv větší.

Zjištění morfů i vzdáleností by však mělo by realizovatelné v jednom průchodu, a ve druhém zápis se změnšováním stromu. Ovšem nebylo by to asi nic rychlého :-(.

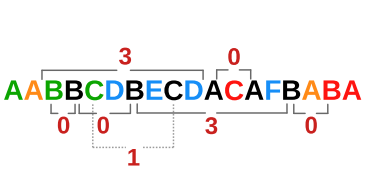
ČÁSTEČNÉ ZEVZDÁLENOSTNĚNÍ

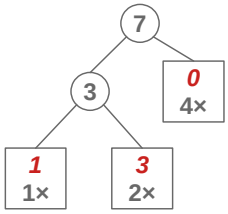
Do vzdáleností se nemusí započítávat znaky které jsou známy z pozicování a odvozování, čímž by se mělo dostat o něco menší počet druhů vzdáleností. Na příkladu se tedy do vzdáleností počítají jen **černé** znaky.

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** (**BBC**). Pro **B** a zahrnutí ho do stromu od **B** (a ne od **B**) by platila vzdálenost **0**; a **C** by díky vzdálenosti **1** vůbec nebylo ve stromu.

Tudíž např. znak **C** *má 3 výskyty – 1. daný počáteční pozicí* (C)*; 2. daný vzdáleností* (**C**)*; a 3. daný koncovou pozicí* (C)*, takže nepotřebuje být dán do stromu.*

Na tomto malém příkladě by ***komplet zevzdálenostnění*** *s vynecháním odvoditelných znaků sice znamenalo mnohem menší startovní strom, ale skutečně předpokládám, že v reálu by to dopadlo tak, že by se to dalo popsat stylem: „sice lepší než nic, ale i tak v podstatě k ničemu“.*





Ale kolik použít vzdáleností aby se ušetřilo místo tím že znaky díky nim nebudou ve stromech? A zkusit na zápis vzdáleností aplikovat také třeba dva stromy? Nebo ještě přidat možnosti s tím, že pro různé znaky je brán různý počet vzdáleností, a to třeba ještě od různých pozic znaků? *Např. tedy že pro A by byly 3 vzdálenosti, ale až od 5. výskytu A, pro B třeba 2 vzdálenosti od prvního neodvoditelného výskytu apod.?*

Strom by šel vzdálenostmi zkrátit i kdyby se braly odzadu, tudíž např. při jedné vzdálenosti odzadu by znak byl ze stromu odstraněn již při předpředposledním výskytu.

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 :-).*

Tudíž teoreticky je toho možné sesmolit asi hodně, ale asi by se slušelo alespoň popřát případnému programátorovi, aby si přitom třeba nemusel připadat jen nějaká ta umělá inteligence na drogách :-).

Anglickou verzi jsem překládal podle sebe, přičemž 2 poslední stránky už na mě byly poněkud moc a dal jsem to s DeepL. Kdybyste se někdo třeba nudil a přeložil to celé správně, dal bych na Github klidně i váš lepší překlad :-).