TrýSort – stromové řazení

Kamil Landa

http://github.com/KamilLanda/TreeSort

v **LibreOffice** http://libreoffice.org 12/2022

TrýSort (stromové řazení, z angl. Tree Sort) je algoritmus pro řazení, odstranění duplikátů a nalezení unikátů – vše v jednom. Lze rozšířit i pro rychlé vyhledávání stejně jako pro rychlé vyhledávání nezávislé na velikosti písmen či diakritice.

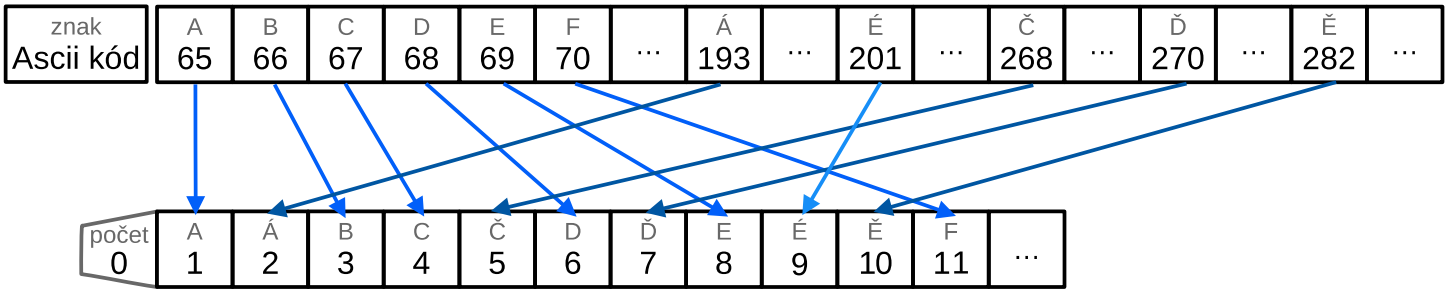
V algoritmu není žádné porovnávání řetězců ani jejich prohazování. V prvním průběhu zjistí jaké znaky jsou v položkách použity, ve druhém sestaví statistiku pro znaky položek v podobě stromu, a jedno přečtení onoho stromu poskytne výsledek.

Největším problémem může být velikost potřebné RAMky (hlavně pokud budou ve vstupních datech víceméně unikáty a budou jich milióny a více), ale nezkoumal jsem jak ji předem odhadnout či spočítat. Pro skrblení RAMky může být dobrá Komplexní varianta, ale ukázky v LibreOffice Basicu jsou hotové pro Jednoduchou a Rozšířenou variantu, a pro Jednoduchou v Pythonu.

Jednoduchá varianta

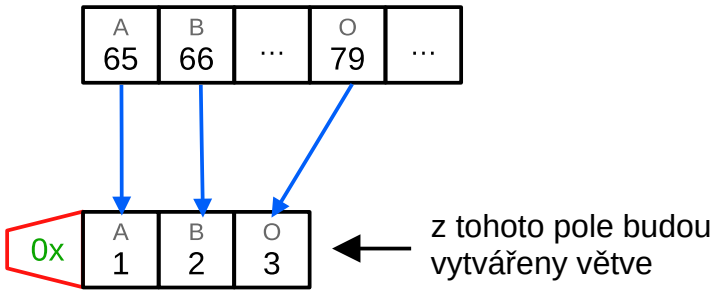
Jednoduchá varianta neznamená méněcenná, ale nejjednodušší na naprogramování a zároveň nejrychlejší. Počítá pouze s jednoznakovými znaky, není v ní možné řadit třeba dle českého písmene **ch**.

Každý znak má svůj Ascii kód a je potřeba vytvořit pole, ve kterém budou všechny použité znaky seřazené. Nebudou v něm však jako znaky, ale jako číselné indexy.

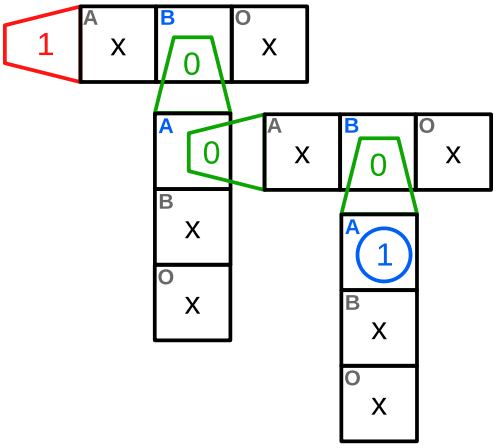


Strom poté bude z větví tvořených tím indexovým polem – dle toho jak budou načítány jednotlivé položky (slova). Pro každé další písmeno slova bude vytvořená další podvětev.

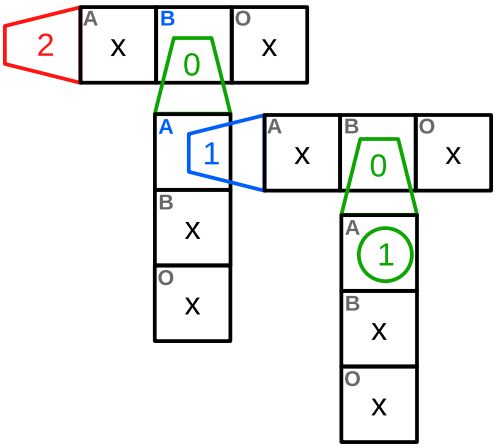
Příklad pro slova: BABA, BA, OBA, BAOBAB, OB, BOB, BOA. Bude se vystavovat pole o čtyřech prvcích, nultý index udává kolikrát je dané slovo obsaženo, zbytek jsou indexy pro jednotlivá písmena. Nejprve se vždy zjistí pro který Ascii znak je jaký index, a pak se staví větve z pole indexů.



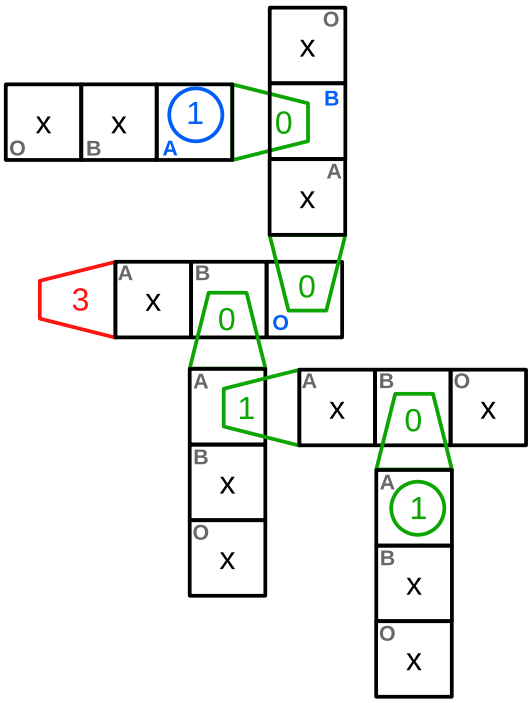
Pro první slovo **BABA** bude strom vypadat následovně. Červeně je nultý index stromu který udává celkový počet slov, každá další podvětev je pro jedno písmeno. Modře je označeno kde slovo končí a kolikrát se ono slovo vyskytuje. **x** znamená prázdnou buňku.



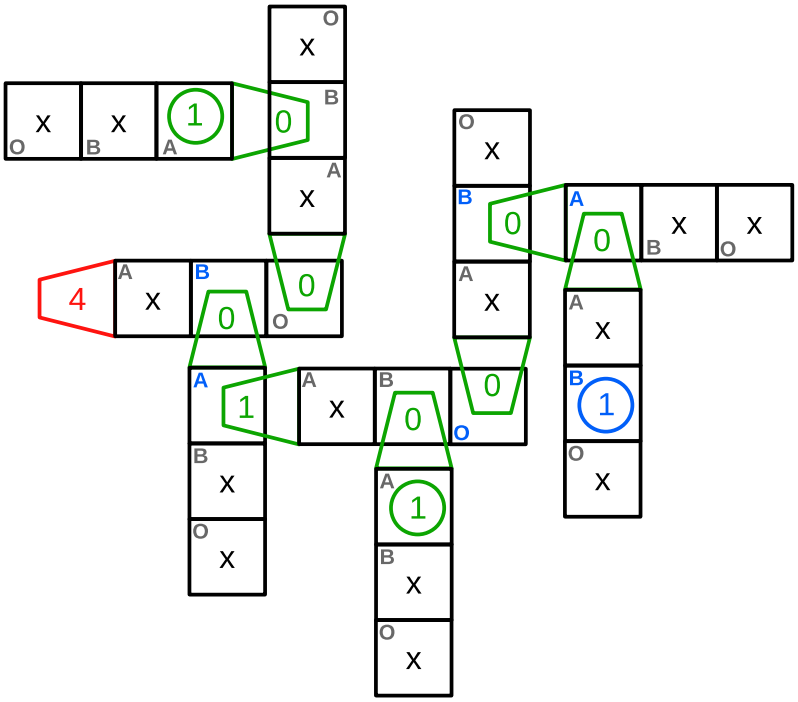
Další přibyde slovo **BA**. Nultý index v nějakém podpoli značí že tam slovo končí a kolikrát tam je. Zde jiná modrá označuje že je tam slovo **1x**.



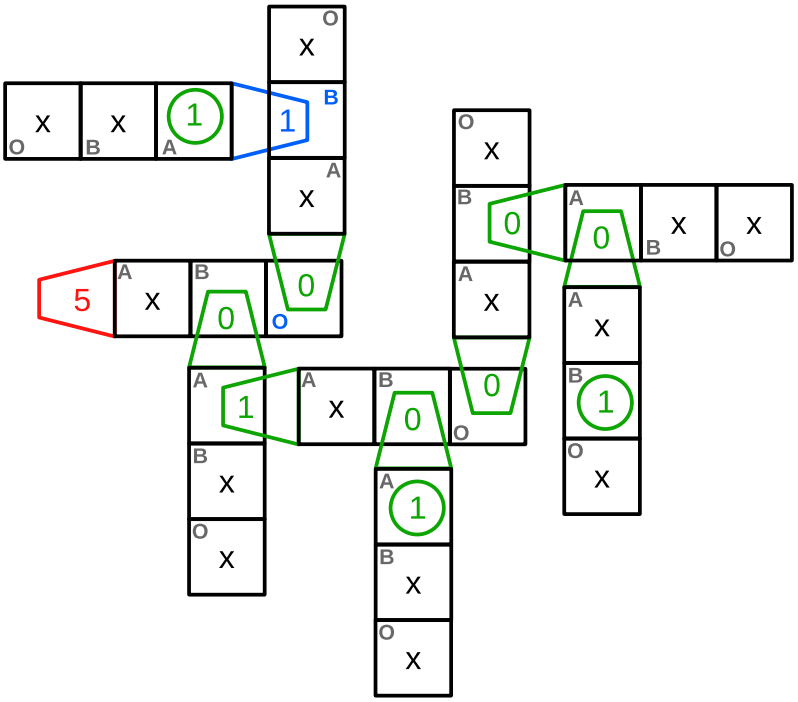
Následující je **OBA**.



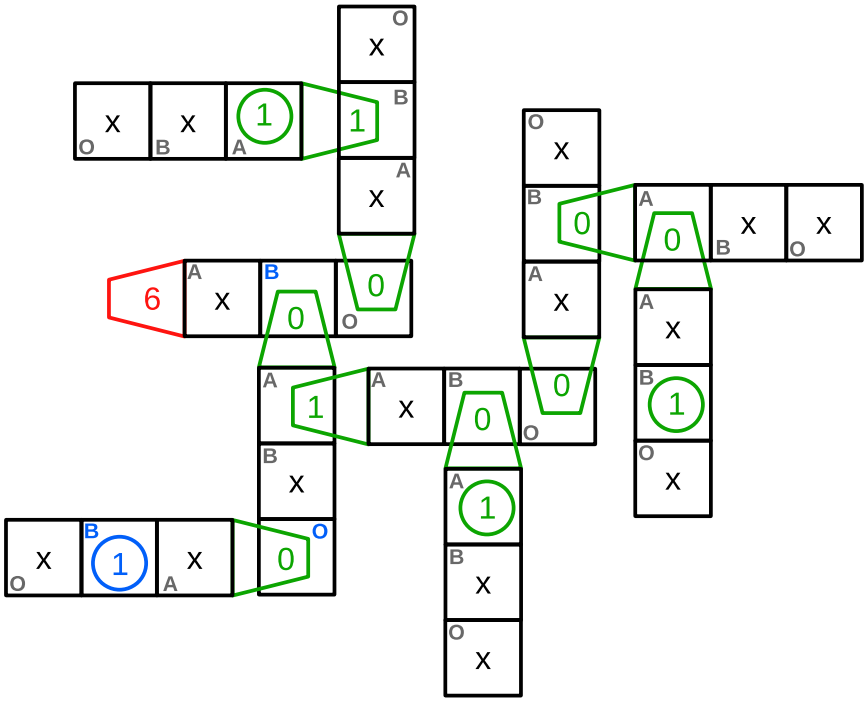
Poté **BAOBAB**.



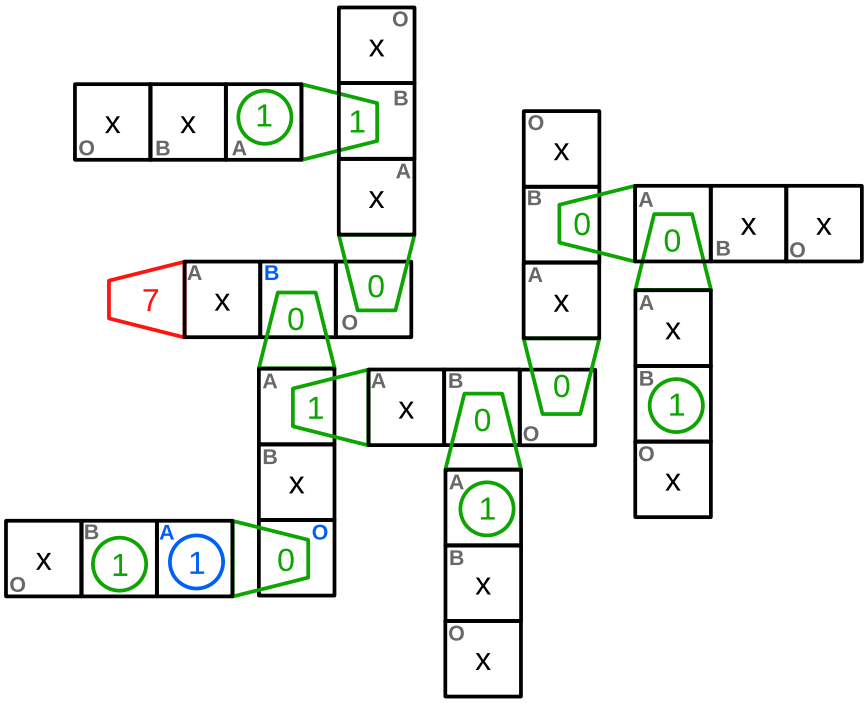
Následován **OB**.



Předposlední **BOB**.

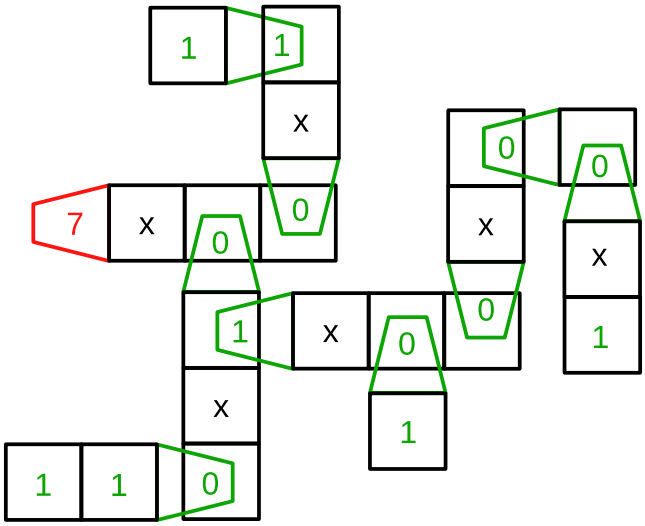


A poslední **BOA**.



Systematické přečtení stromu (využil jsem rekurzi) poté dá výsledek → seřazená slova s počtem kolikrát se vyskytují. Unikáty jsou pouze jednou; bez duplicit je výpis bez počtů; a pro seřazený seznam je potřeba vypsat každé slovo tolikrát kolikrát se vyskytuje.

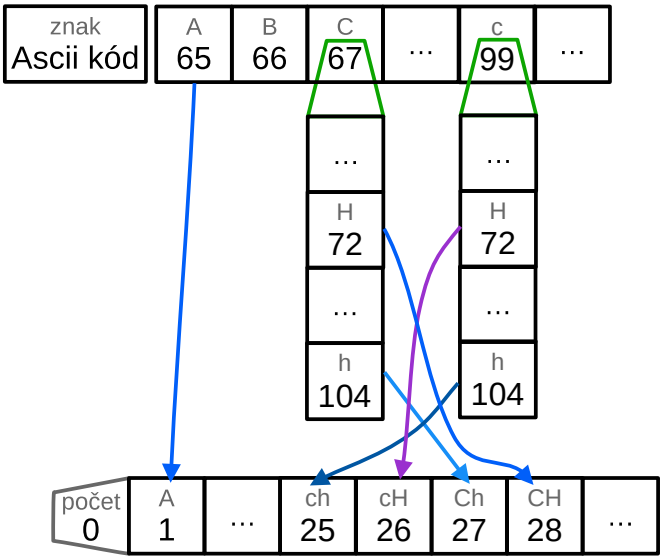
Je vidět že nějaké přečnívající větve by šly uříznout, čímž by se ušetřila nějaká ta RAMka. Strom by pak vypadal v podstatě následovně – uvedeno bez pomocných písmen neb vše je jen dle indexů.



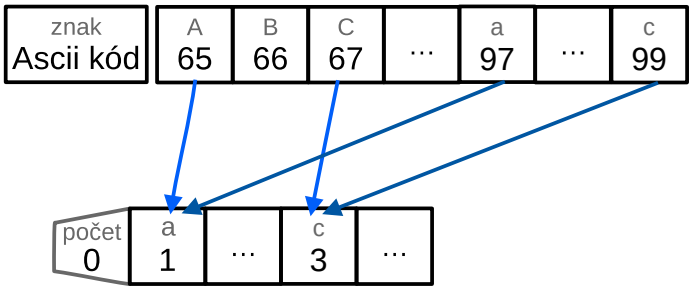
Rozšířená varianta

Zde je počítáno s tím že se bude řadit i dle dvojpísmen, tedy např. i dle českého **ch**. S detekcí i neznámých (nepředpokládaných) znaků v datech které jsou řazeny na konci dle jejich Ascii kódů. Do ukázky je přidána i možnost pro otesané řazení s ignorováním velikosti písmen – což znamená že vše bude natvrdo převedeno pouze na malá nebo jen na velká písmena.

Strom je pak stavěn úplně stejným způsobem, tedy dle pole indexů. Při načítání znaků se ale musí kontrolovat je-li znak částí dvoupísmene, pro což je využívána funkce **IsArray()**. Je-li v tabulce Ascii znaků na dané pozici proměnná typu pole, znamená to, že má podpole s druhými možnými písmeny.

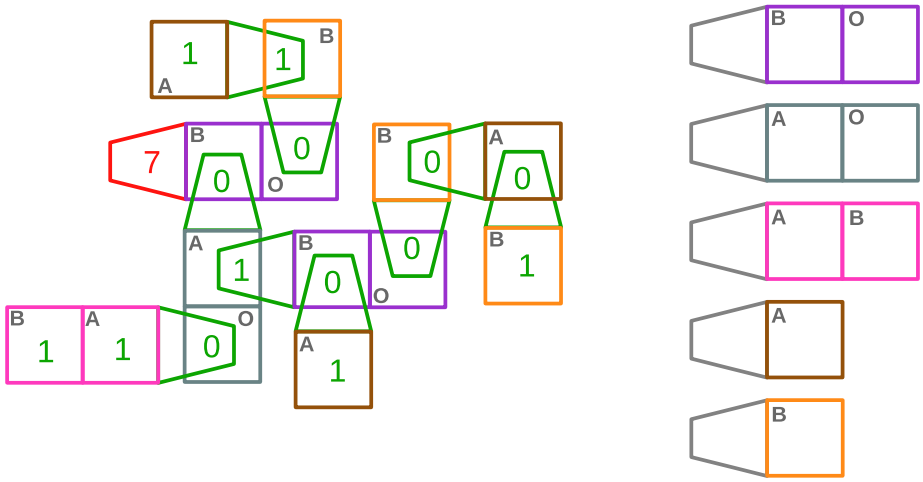


Otesané řazení s ignorací velikosti písmen bude velká i malá písmena indexovat třeba jen do malých písmen nebo jen do velkých. Výsledek pak bude vypsán jen v malých nebo jen velkých písmenech.

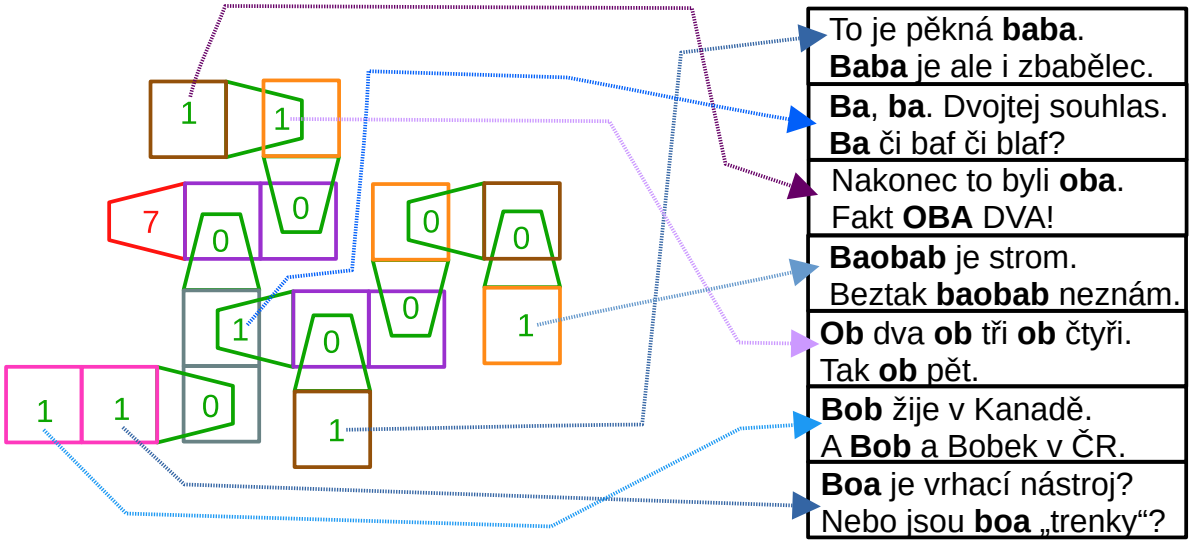


Komplexní varianta

Naskýtá se však i možnost použít různé větve. Pak by to vypadalo třeba takto – různé druhy větví jsou odlišené barevně. Je sice potřeba přidat ještě informaci pro odlišení větví, ale ve stromu pak nejsou žádné nadbytečné prázdné buňky.



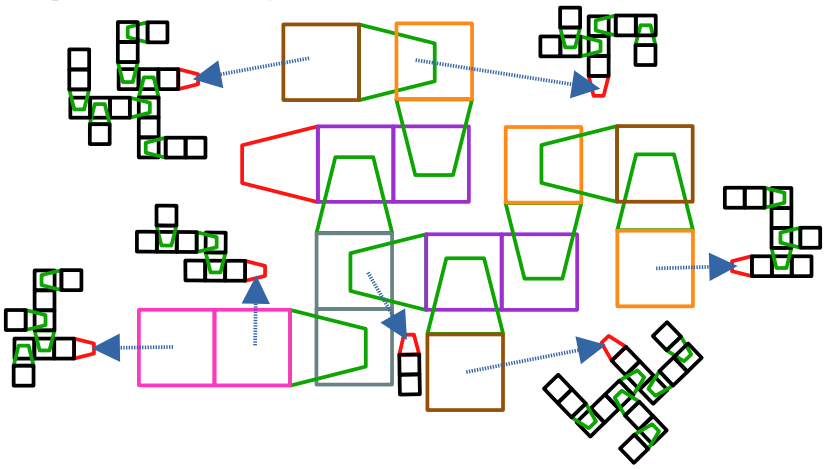
Komplexní varianta počítá s tím, že do stromu je možné přidat odkazy na původní data. To poté umožní rychlé prohledávání původních dat. Výhoda je v tom když je potřeba prohledávat fráze → strom lze vystavět pro všechna jednotlivá slova, ale koncové indexy povedou na původní fráze. Tudíž vyhledání jednoho slova ukáže na všechny fráze ve kterých se slovo vyskytuje.



Do jednoho indexu lze otesat nejen malá i velká písmena, ale i písmena s diakritikou. Např. jakožto **a** by byla brána: **a A á Á**, nebo jakožto **e** by bylo bráno: **e E é É ě Ě**. Takže může být vyhledáváno slovo bez diakritiky v datech s diakritikou. Například otesaný bezdiakritický strom může mít ve větvích slovo **rad**, ale konečné indexy půjdou na slova **rad** *(mnoho rad)* i **raď** i **rád** i **řád** i **řaď**. Indexace na původní data zabere opět víc RAMky, ale může být vhodná i pro opakované prohledávání.

Multistrom

To je případ kdy skupiny dat na které vedou zpětné indexy jsou též nastromkovány, přičemž hlavní strom může být otesaný od diakritiky i velikosti písmen a podstromky mohou být úplné. Hledaný termín pak může být napsaný přesně: **řaď**, ale prohledáván bude jako **rad** a teprve v podstromku bude vyhledána fráze s **řaď**.



Les?

Je však možné přidat do stromu všechny části všech slov, například na slovo AHOJ by vedly indexy z AHOJ, AHO, AH, A, HOJ, HO, H, OJ, O, J. A udělat to i pro všechny kombinace velkých malých písmen i pro všechny kombinace znaků s diakritikou či bez ní. Může to vzít hodně RAMky, ale může to být dobré pro vyhledávání dle částí slov. Anebo si pro to udělat vyloženě nějaký lesík :-).

Testování (rychlost a RAMka)

LibreOffice Basic

Udělal jsem pár testů pro nějaké TXT soubory abych porovnal řazení v LibreBasicu pro: TrýSort (rozšířenou variantu), HeapSort z knihovny ScriptFroge, a QuickSort ze starší verze ScriptForge (LibO primitives). Co se týká rychlosti tak TrýSort docela s přehledem vedl :-). Jednoduchá varianta TrýSortu řadí ještě rychleji, ale je potřeba do ní kdyžtak doplnit více druhů znaků nebo načítání ze souboru – což jsem abych ji nechal jednoduchou pro lehčí studování nedělal; stejně jako se mi nelíbilo ignorovat pro řazení česká ***ch****áčka* :-).

Počet slov	TrýSort	QuickSort	HeapSort
2 014	0*m*:01*s*	0:00	0:08
6 897	0:04	0:01	0:30
58 613	0:27	0:46	5:33
121 494	0:53	2:44	12:15
340 094	2:40	*	**
1 015 803	7:54	**	**

** asi po 4 minutách to ukázalo chybové hlášení: Nedostatek paměti v zásobníku. (tak nějak)*

*** Zelený úděl (greendeal) je zelený úděl; a byl jsem laskavý k větráčku v laptopu :-).*

Co se týká spotřeby RAM, tak pro Rozšířenou variantu kde větve nejsou nijak otesány zabral největší testovaný soubor s cca miliónem položek asi 300MB. Pro český slovník synonym z LibreOffice který má cca 260 tisíc slov (téměř unikáty) byl strom velký asi 2,2GB.

Python v editoru Pyzo

Zkoušel jsem Jednoduchou variantu. Testoval jsem na vygenerovaném poli 6-ti písmenných znaků A-Z. Nejvíce pro 20^6 kombinací čili pro 64 miliónů položek. Pro porovnání jsem použil algoritmus Timsort, kde data byla seřazená od *zetek* po *áčka*, aby si přeci jen zatřídil. TrýSortu je jedno jak jsou vstupní data uspořádána, tomu se při stavě stromu neuleví.

Počet slov	TrýSort	Timsort
1 000 000	5*s*	8,2
2 985 984	15	25,56
11 390 625	62,11	116,43
64 000 000	333,27	694,27

Největší tříděné pole mělo asi 4,6 GB, s Timsortem jsem se dostal asi na 8,8 GB a s TrýSortem asi na 15GB RAM, neb vytváření nového výstupního pole a rekurze si vybraly své.

Více otestovat jsem nebyl schopen. Vytváření stromu a jeho přečtění může být jistě mnohem rychlejší v nějakém kompilovaném jazyce. Není však v mých možnostech to však zkusit naprogramovat třeba v C++ nebo Rustu.

Chyby v LibreOffice pro LibreBasic

Při testování jsem narazil na dvě docela nepěkné chyby *(12/2022 LibreOffice 7.4.3.2 Win10x64)*.

Chyba https://bugs.documentfoundation.org/show_bug.cgi?id=152613

LibreOffice při té rekurzi použité ke čtení stromu neuvolňuje RAMku. Při testu na 260 tisících téměř unikátních slov z českého slovníku pro kontrolu pravopisu z LibreOffice při čtení stromu po pár minutách došlo 24GB paměti a Windows udělal zmalebněnou modrou smrt (krom textu tam byl i jakýsi obrázek) :-(.

Chyba https://bugs.documentfoundation.org/show_bug.cgi?id=152466

V určitém případě se pro pole polí neaktualizuje Kukátko pro sledování proměnné.

Počáteční nápad

Má angličtina není nic moc a s oblibou používám jednoduchý offlajn cs/en slovník **ACS Slovník**. Když v něm však dám vyhledat nějaké slovo, objeví se většinou několik slov v druhém jazyce, např. pro slovo ***příklad*** to vypíše: ***exemplar***, ***instance***, ***paragon***, ***example***. No a jak si z toho mám vybrat co je vhodné slovo? Tak přepíšu ty vyhledané termíny jeden po druhém a sleduji co mi to ještě vypíše:

exemplar → *příklad*
instance → *žádost, případ, instance, výzva, příklad, prosba, situace*
paragon → *ideál, příklad, vzor*
example → *ukázka, vzor, příklad*

Z toho už se většinou dá vybrat.

Shrnutí

Naprogramoval jsem jednoduchou a rozšířenou variantu asi během 10-ti dní (12/2022) v tom v čem v posledních letech prostě dělám nejvíc (LibreOffice Basic) a ač algoritmus považuji za triviální, tak indexace v podpolích pro dvojpísmena mi přeci jen dala poněkud zabrat.

Předposlední den roku 2022 jsem vytvořil a otestoval jednoduchou variantu v Pythonu.

Rozhodoval jsem se jestli se pustit do vytvoření ukázek komplexní varianty i multistromu a pak to teprve uveřejnit, ale rozhodl jsem se, že se podělím již o to co je hotové i když to není dodělané finální pobody a i když i v tom hotovém je třeba místo pro nějakou tu optimalizaci.

LibreBasic na to stejně není moc vhodný, a samozřejmě Python pro to také není zrovna zázrak. Vytvoření stromu a jeho přečtení by mělo být mnohem rychlejší v nějakém kompilovaném jazyce, ale mé znalosti nestačí na to abych to zkusil třeba v C++ nebo Rustu.

Otázkou tedy zůstává jestli vytvoření stromu a jeho přečtení může být rychlejší než různé porovnávání řetězců s prohazováním.

Takže nakonec mi byla milejší představa to raději nasdílet a nechat i jiné jestli si s tím třeba vyhrají, poradí nebo to vylepší, zkontrolují apod. :-).

Přesto doufám že to bude ku prospěchu a bude to šířeno i z této podoby :-).