Hľadanie slov v texte pomocou konečného automatu

Štefan Raschmann, Pavol Zajac *
Katedra aplikovanej informatiky a výpočtovej techniky, Fakulta elektrotechniky a informatiky STU
stefan.raschmann@gmail.com

Abstrakt

Práca sa zaoberá problematikou rýchleho hľadania slov v texte, predovšetkým za účelom ohodnocovania textov pri automatickej kryptoanalýze. Náš postup porovnávame s najčastejšími spôsobmi hľadania slov v texte. V závere sú uvedené dosiahnuté výsledky .

1. Úvod

V automatizovanom lúštení šifier použitím počítača, je nesmierne dôležité zostrojiť funkcie ‚ktoré dokážu číselne ohodnotiť zmysluplnosť textu. Takýmto funkciám hovoríme „ohodnocovacie funkcie“ a výstup tejto funkcie nazývame skóre. Najčastejšie ohodnocovacia funkcia vracia tým vyššiu hodnotu, čím viac sa skúmaný text podobá na zmysluplný text z daného jazyka.[1]

2. Slovníkové metódy

Pomocou slovníka môžeme implementovať veľmi spoľahlivé ohodnocovacie funkcie. Označme s skóre, N_d je počet d - písmenových podreťazcov z textu dĺžky L, ktoré sa nachádzajú v slovníku, v ktorom najdlhšie slovo má dĺžku n. Potom môžeme zostaviť napr. nasledujúce ohodnocovacie funkcie:

1. Počet slov v texte:
$$s = \frac{1}{L}\sum_{d=3}^{n} N_d \qquad (1)$$
2. Súčet písmen nájdených slov v texte:
$$s = \frac{1}{L}\sum_{d=3}^{n} d.N_d \qquad (2)$$
3. Súčet štvorcov počtu písmen nájdených slov v texte:
$$s = \frac{1}{L}\sum_{d=3}^{n} d^2 N_d \qquad (3)$$

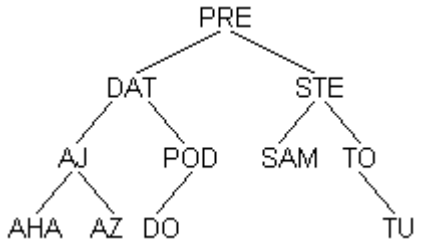
3. Implementácia slovníka

3.1. Binárne vyhľadávanie

Vyhľadáva hľadané slovo v usporiadanom zozname pomocou skracovania zoznamu o polovicu v každom kroku. V každom kroku, algoritmus porovná hľadané slovo s prostredným slovom v zozname a rozhodne sa či bude pokračovať v dolnej alebo hornej časti zoznamu. Pre N slov v zozname je potrebných $1 + log_2 N$ krokov algoritmu na nájdenie slova, čo pre typický slovník obsahujúci 200 000 hesiel predstavuje 18 krokov. Výhodou tejto metódy je ľahká implementácia. Nevýhodou je, že je možné vyhľadávať až po utriedení zoznamu. Zdroj.: [5]

3.2. AVL strom

Je samovyvažovaný binárny vyhľadávací strom. Každý uzol má maximálne dvoch potomkov a pre každý uzol stromu platí, že naľavo od uzla sa nachádzajú uzli z menšími hodnotami a napravo z väčšími (platí aj pre potomkov). Prakticky ide o metódu podobnú binárnemu vyhľadávaniu, len sa použije štruktúra do ktorej je ľahké vkladať nové hodnoty. Pre N slov v slovníku je zložitosť nájdenia slova v slovníku $log_2 N$. V porovnaní z binárnym vyhľadávaním má AVL strom zložitejšiu implementáciu, trochu väčšie pamäťové nároky ale na druhej stane sú hodnoty v strome vždy utriedené aj po vkladaní nových hodnôt. Zdroj.:[6]

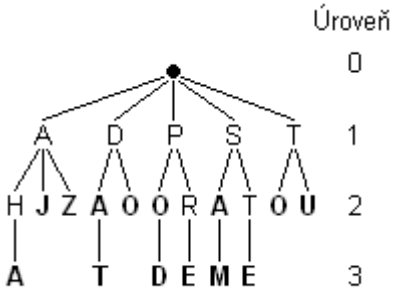


Obr. 1. AVL – samovyvažovací binárny vyhľadávací strom

* Vedúci práce

3.3. Znakový strom (Trie)

Označovaný tiež ako prefixový strom. Každý uzol znakového stromu predstavuje jedno písmeno abecedy preto každý uzol môže mať práve toľko potomkov koľko znakov má použitá abeceda. Slovo je v strome uložené od koreňa postupne a vetví sa podľa odlišných znakov. Slovo tvoria písmená ľubovoľnej cesty, zhora nadol, z koreňového uzla po uzol označený ako koniec slova. Na nasledujúcom obrázku máme znakový strom v ktorom sa nachádzajú slová: AHA, AJ, AZ, DAT, DO, PO, POD, PRE, SAM, SA, STE, TO, TU.

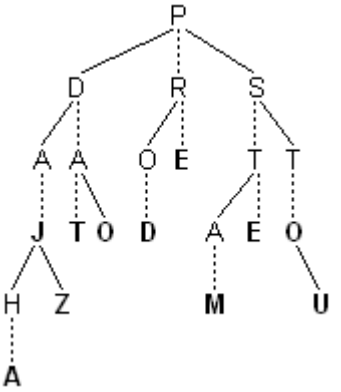


Obr. 2. Znakový strom

Ak *n* je dĺžka hľadaného slova, zložitosť nájdenia slova je *n*. Tento spôsob realizácie slovníka má hlavnú výhodu v tom, že zložitosť nájdenia slova nezávisí od počtu slov v slovníku, ale od dĺžky hľadaného slova. Najčastejšie sa Znakový strom implementuje tak, že každý uzol obsahuje pole ukazovateľov na ďalšie uzly o veľkosti počtu písmen abecedy. V našom prípade, ak uvažujeme TSA (Telegrafná slovenská abeceda obsahujúca 26 písmen), potrebuje každý uzol pole smerníkov na 26 potomkov čo zaberá v pamäti 26*4B = 104B. Pre strom z Obr.2. ktorý obsahuje 23 uzlov je to 2392B.
Hlavnou výhodou znakového stromu jeho rýchlosť za ktorú ale zaplatíme pomerne vysokou pamäťovou náročnosťou. Zdroj.:[7]

3.3. TST (ternary search tree)

Tento strom (ďalej len TS strom) spája výhody malej pamäťovej náročnosti AVL stromu a veľkej rýchlosti znakového stromu. Každý uzol môže mať najviac troch potomkov. Naľavo alebo napravo od uzla sú potomkovia ktorý ho môžu nahradiť. V strede je uzol ktorý má rovnaký význam ako uzol v znakovom strome. Ďalej platí že všetci potomkovia na ľavo od stredného uzla, tej istej úrovne, sú menší ako stredný uzol a všetci potomkovia napravo od stredného uzla, tej istej úrovne, sú väčší ako stredný uzol. Príklad takéhoto stromu vidieť na obr.3. ktorý obsahuje rovnaké heslá ako stromy z obr.1. a obr.2. Zdroj.:[8]



Obr.3. TS strom (ternary search tree)

4. Hľadania slov v texte pomocou slovníka

Implementovali sme jednoduchý znakový strom v jazyku C++. Do slovníka sme vložili slová zo slovníka zverejneného na stránke predmetu Klasické šifry[4] ktoré mali dĺžku aspoň 4 znaky. Po načítaní slovníka sa v strome nachádzalo 195 715 slov uložených v 437 195 uzloch. Jeden uzol zaberá v pamäti 109B čiže celý slovník zaberá najmenej 45.44MB. Takýto slovník už v dnešných časoch nie je problém implementovať na bežnom PC.
Označme root ako koreňový uzol znakového stromu. Potom sa pre tento znakový strom dá definovať funkcia ktorá vracia súčet dĺžok všetkých prefixov reťazca ktoré sa súčasne nachádzajú v slovníku.

Algoritmus 4.1.: Nájdi_slovo
VSTUP: root, reťazec
VÝSTUP: dĺžka nájdeného slova

```
Nastav i = 0
Nastav b = root
Nastav dlz = 0
Opakuj pre i=0,1,...,dĺžka reťazca:
   Nastav c = reťazec[i] – 'A'
   Ak existuje b->potomok[c]:
      b = b->potomok[c]
      Ak b je koniec slova:
         dlz = dlz + i + 1
   Inak:
      Vráť dlz
```

Pre nájdenie všetkých slov v reťazci môžeme definovať nasledujúci algoritmus.

Algoritmus 4.2.: Nájdi_všetky_slová
VSTUP: root, reťazec
VÝSTUP: súčet dĺžok všetkých nájdených slov

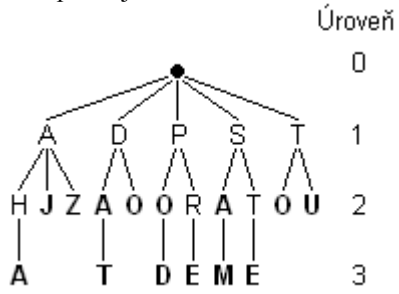
```
Nastav dlz = 0
Nastav r = ""
Opakuj pre i=0,1,…,dĺžka reťazca:
   r = reťazec[i:]
   dlz = dlz + Nájdi slovo(root, r)
Vráť dlz
```

Pozn. zápis reťazec[i:] predstavuje podreťazec reťazca začínajúc *i*-tym znakom po koniec reťazca.

Pre strom z obr.2. a pre vstupný reťazec "DATABAZA" vráti algoritmus 4.1.(Nájdi_slovo) hodnotu 5 nakoľko sa v slovníku nachádzajú slova DA, DAT ktoré sú zároveň prefixom reťazca "DATABAZA".
Algoritmus 4.2.(Nájdi_všetky_slová) potrebuje pre 1000 znakový náhodný text v TSA v priemere 2966 krokov a pre zmysluplný text v priemere 4668 krokov.
Na počítači s CPU 1,65GHZ má algoritmus rýchlosť 24,78 MB dlhý náhodný reťazec za sekundu.

5. Použitie konečného automatu

Myšlienka vytvorenia konečného automatu na hľadanie slov texte vznikla pri implementácii znakového stromu a vo veľkej miere pomohla nevedomosť existencie TS stromu. Ako už bolo spomenuté, pri implementácii znakového stromu musí každý potomok obsahovať 26 ukazovateľov pri použití TSA. To vedie pomerne k veľkému plytvaniu pamäte. Ak sa opätovne pozrieme na znakový strom tak na obrázku predstavuje jedna čiara práve jeden ukazovateľ.



Obr.4. Znakový strom.

Spočítaním čiar prídeme na to, že ich je presne o jeden menej ako počet všetkých uzlov. Ináč povedané na každý uzol okrem koreňového ukazuje práve jeden ukazovateľ. Každý uzol má vytvorených 26 ukazovateľov z čoho vyplýva, že iba každý 26-ty vytvorený ukazovateľ sa v strome využíva. 25 ukazovateľov z 26-tich má nulovú hodnotu.
V TS strome sa problém rieši rapídne zmenšenou spotrebou pamäte nakoľko každý uzol tam môže mať len troch potomkov. My sa uberieme cestou 100% využitia pamäte ponechaním všetkých 26 potomkov na uzol. Zavedieme si nasledujúce definície:

Definícia 5.1. *Abeceda je konečná neprázdna množina symbolov (písmen, znakov, symbolov a pod.) Často sa označuje znakom* $\sum$*, ich prvky znakmi a, b, c (s prípadnými indexmi a pod.)*
Slovo v abecede $\sum$ *je ľubovoľná konečná postupnosť* $a_1,a_2,...,a_n$ *prvkov* $\sum$ *a píšeme ho obvykle bez čiarok. Slová obvykle označujeme symbolmi u, v, w a pod. Prirodzeným spôsobom definujeme dĺžku slova, označujeme* $|u|$*; pre* $u = a_1a_2...a_n$ *je* $|u| = n$*. Prázdne slovo označujeme symbolom* ε*. Spájanie slov u, v označujeme* $w = uv$*.* [3]

Definícia 5.2. *Formálny jazyk, stručne jazyk, nad abecedou* $\sum$ *je ľubovoľná množina slov v abecede* $\sum$*. Jazyky obvykle označujeme L(s indexmi); Pre jazyk L nad abecedou* $\sum$ *je teda* $L \subseteq \sum *$*.* [3]

5.1. Konečný automat

Konečný automat je model systému, ktorý môže nadobúdať konečne mnoho stavov. Daný stav sa mení na základe vonkajšieho podnetu s tým, že pre daný stav a daný podnet je jednoznačne určené, aký stav bude nasledujúci. (t.j. do akého stavu systém prejde)
Konečné automaty sa často zobrazujú diagramom, ktorý taktiež nazývame stavový diagram alebo graf automatu.
Matematický zápis konečného automatu:

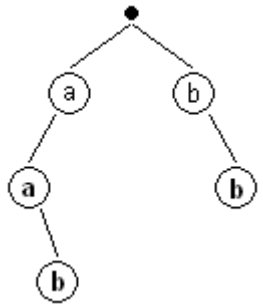
Definícia 5.3. *Konečný automat A je pätica* $A = (Q, \Sigma, \delta, q_0, F)$ *kde:*

- Q *- je konečná neprázdna množina stavov*
- $\sum$ *- je konečná neprázdna množina zvaná vstupná abeceda*
- $\delta : Q \times \sum \to Q$ *- je prechodová funkcia*
- $q_0 \in Q$ *- je počiatočný stav*
- $F \subseteq Q$ *- je množina prijímacích (koncových) stavov* [3]

Jazyk rozpoznávaný daným konečným automatom je množina postupnosti symbolov (prvkov abecedy), ktoré automat prijíma (akceptuje), t.j. tých postupností, ktoré automat prevedú z počiatočného stavu do niektorého prijímacích stavov.

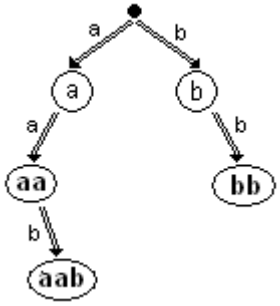
5.2. Vytvorenie konečného automatu

Nech abeceda $\sum = \{a, b\}$ a jazyk $L = \{aa, aab, bb\}$. V prvom kroku si môžeme zostrojiť znakový strom obsahujúci slová v jazyku *L*.



Obr.5. Znakový strom obsahujúci slová {aa, aab, bb}.

V druhom kroku označíme uzly ako stavy, kde každý stav musí byť jedinečný, a taktiež si označíme všetky hrany, aby sme vedeli, ktorá hrana bola pridaná, a ktorá patrí pôvodnému znakovému stromu. Následne ešte hrany ohodnotíme. Výsledný graf, označme symbolom *G,* vidieť na obr.6.



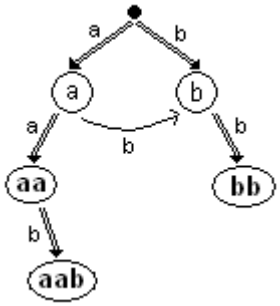
Obr.6. Graf *G* - Druhý krok tvorby automatu.

Definícia 5.4. *Hovoríme, že w je suffixom reťazca x, označujeme w* $\sqsupset$ *x, ak* $|w| \leq |x|$ *a x = yw pre nejaký* $y \in \Sigma*$ Napr. báza je sufixom reťazca databáza. [2]

Tretí krok algoritmu tvorby konečného automatu:
Nech *u* je ľubovoľný uzol z *G*. Potom reťazec ktorý predstavuje označme `*u*`. Pre všetky uzly $u \in G$ urobíme nasledujúcu operáciu:

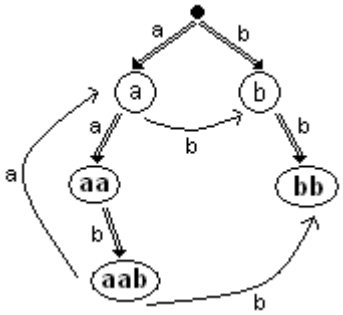
Opakuj pre: $\forall i \in \Sigma$
  Ak uzol u nemá hranu i:
    r = `u`i
    Nájdi uzol u_2 taký že: `$u_2$` = max{`$u_2$` $\sqsupset$ r}
    K uzlu *u* na hranu *i* pripoj uzol u_2.

Na obr.7 vidieť vykonanie operácie pre uzol *a*. Uzol *a* nemal hranu *b*. Preto bolo treba nájsť taký uzol u_2 ktorý je najväčším suffixom reťazca ab. Keďže uzol ab neexistuje, ostal už len uzol *b*.



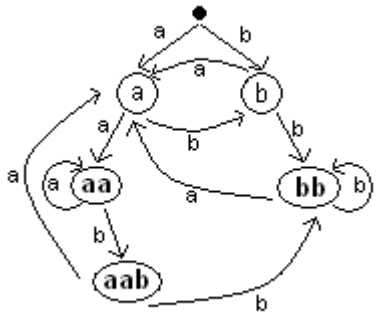
Obr.7 Vykonanie operácie pre uzol *a*.

Na obr.8 Sme urobili rovnakú operáciu z uzlom *aab*. Ten nemal hrany *a,* b. Pre hranu *a* treba hľadať uzol ktorý je najväčší suffix reťazca *aaba*. Takým uzlom je uzol *a*. Pre hranu *b* treba hľadať uzol ktorý je najväčší suffix reťazca *aabb*. Takým uzlom je uzol *bb*.



Obr.8. Vykonanie operácie pre uzol *aab*.

Výsledný automat vidieť na obr.9. Klasický konečný automat slúži na rozpoznávanie, či sa v reťazci nachádza slovo patriace jazyku L. To znamená že konečný automat ukončuje svoju činnosť, ak prejde do prijímacieho (ukončovacieho) stavu. Pre naše potreby sme automat upravili tak, že svoju činnosť ukončuje až vyčerpaním celého vstupného reťazca. Výstup automatu závisí od aplikácie, v prípade ohodnocovacej funkcie je to napr. koľko krát bol automat v prijímacom stave (1), alebo pre každý prijímací stav môže ku skóre pripočítať vzdialenosť prijímacieho stavu od „koreňového uzla“, t.j. dĺžku akceptovaného slova (2).



Obr.9. Výsledný konečný automat.

5.3. Výsledky

Výsledný algoritmus, ktorý spočíta dĺžky všetkých nájdených slov v reťazci pomocou automatu vyzerá nasledovne:

Algoritmus 5.1.: Nájdi_všetky_slová
VSTUP: root, reťazec
VÝSTUP: súčet dĺžok všetkých nájdených slov

```
Nastav i = 0
Nastav b = root
Nastav score = 0
Opakuj pre i=0,1,...,dĺžka reťazca:
   Nastav c = reťazec[i] – 'A'
   b->potomok[c]:
      b = b->potomok[c]
   Ak b je prijímací stav:
         score = score + b->dĺžka
   Inak:
      Vráť score
```

Tento algoritmus je jednoduchší ako algoritmus pre hľadanie slov pomocou znakového stromu. Kým algoritmus používajúci znakový strom potreboval pre náhodný reťazec v TSA dĺžky 1000 v priemere 2966 krokov a pre zmysluplný text v priemere 4668 krokov algoritmus používajúci automat potrebuje v oboch prípadoch len 1000 krokov.
Na počítači s CPU 1,65GHZ má algoritmus rýchlosť 92,56MB/s oproti 24,78MB/s pri použití znakového stromu.

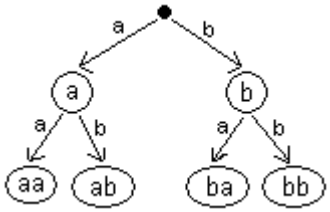
7. Paralelné ohodnocovanie

Nami navrhnutý automat môže slúžiť aj ako n-gramový test. Stačí ako namiesto slov budeme do neho vkladať jednotlivé n-gramy. My si ukážeme ako sa dá súčasne v jednom automate vykonať slovníkový aj n-gramový test.
Nech abeceda $\sum=\{a, b\}$ a jazyk $L = \{aa, aab, bb, baa\}$. A zároveň chceme vykonať digramový test, taký že, každému nájdenému digramu v reťazci pridelíme skóre podľa tab.1.

Digram	skóre
aa	2
ab	4
ba	7
bb	1

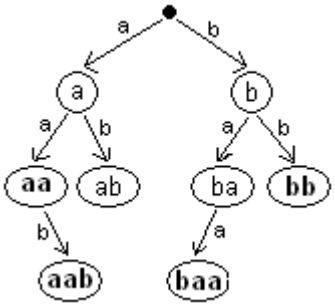
Tab.1. Skóre pre jednotlivé digramy.

V prvom kroku si vytvoríme znakový strom obsahujúci všetky digramy.



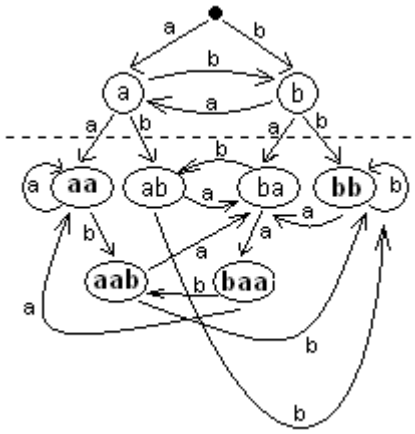
Obr.10. Prvý krok tvorby automatu.

V ďalšom kroku pridáme slová ako vidieť na obr.11.



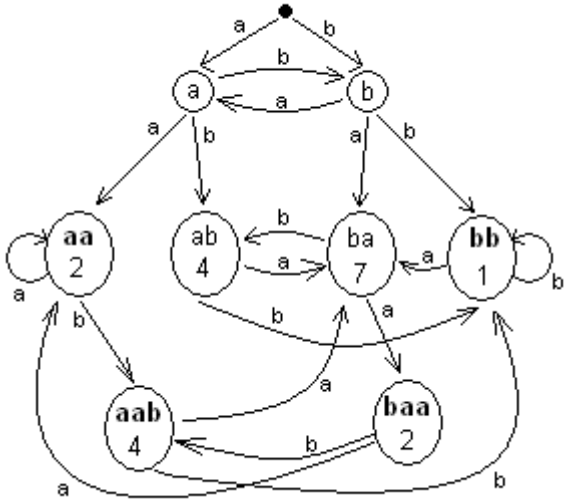
Obr.11. Druhý krok tvorby automatu.

Nasledovne podľa algoritmu popisovaného v kap.5.2. vytvoríme konečný automat.



Obr.12. Tretí krok alg tvorby automatu.

Na obr.12. vidieť, že ak automat prejde do niektorého zo stavov, ktoré sú pod čiarkovanou čiarou, už sa nikdy nevráti do stavu, ktorý je nad čiarkovanou čiarou. Z toho vyplýva, že až na prvý stav vždy bude mať automat informáciu o tom, ktoré boli posledné dve znaky. V štvrtom kroku už len každému uzlu pridáme informáciu, aké skóre priraďujeme jednotlivým digramom.



Obr.13. 4. krok - Automat na paralelne ohodnocovanie.

Algoritmus na ohodnocovanie reťazca pracuje podobne ako algoritmus 5.1. len navyše spočítava pridané čísla. Týmto spôsobom je možné vytvoriť automat ktorý naraz vykoná: digramový, trigramový, tetragramový a slovníkový test. 5-gramový test už nie je možné kombinovať nakoľko v druhom kroku by bolo potrebné pridať 26^5=11.8 mil. 5-gramov.

8. Záver

Využívanie slovníkových metód je veľmi efektívne pri metódach „brute-force“ nakoľko postačuje dešifrovať pre každý kľúč menej znakov ako pri n-gramových testoch. Pri metódach inteligentného prehľadávania už slovníková metóda nemusí byť vhodná pre svoju vysokú „presnosť“. Uvedené slovníkové metódy sa dajú použiť aj pri implementácii n-gramových testov. Vtedy namiesto slov vkladáme do slovníka jednotlivé n-gramy. Implementáciou slovníka, najlepšie pomocou automatu, vytvoríme pomerne univerzálny nástroj na ohodnocovanie textov.

8. Zoznam literatúry

[1] O. GROŠEK, M. VOJVODA, P. ZAJAC.: Klasické šifry [skriptá]. Publikované 2007. str.153-165 ISBN 978-80-227-2653-5

[2] VOJVODA, M.: Prednášky k predmetu Analýza a zložitosť algoritmov kapitola 9. Slovenská technická univerzita v Bratislave, [on-line] Publikované 2008, Dostupné z http://elearn.elf.stuba.sk

[3] JANČAR, P.: Studijní opora k předmětu Teoretická informatika (speciální v oblasti teorie jazyků a automatů) , [on-line] Publikované 25.09.2003, Dostupné z http://kubaz.cz/texty/JancarJazykyAutomaty.pdf

[4] http://147.175.106.2/kaivt/Predmety/KS

[5] http://sk.wikipedia.org/wiki/Bin%C3%A1rne_vyh%C4%BEad%C3%A1vanie

[6] http://sk.wikipedia.org/wiki/AVL_strom

[7] http://cs.wikipedia.org/wiki/Trie

[8] J. BANTLEY, R. SEDGEWICK.: Ternary search tree. Publikované: Dr. Dobbs Journal April, 1998. Dostupné z: http://www.cs.princeton.edu/~rs/strings/