Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia algoritmov a dátových štruktúr

Bakalárska práca

2011 **Viktor Tomkovič**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia algoritmov a dátových štruktúr

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra Informatiky FMFI
Vedúci práce: Mgr. Jakub Kováč

Bratislava, 2011 **Viktor Tomkovič**

Poďakovanie

Tu si nemôžem vyliať dušu na celý svet, ale len stroho poďakovať za „odbornú“ pomoc.

Abstrakt

Táto práca sa zaoberá vizualizáciou dátových štruktúr. Nadväzuje na bakalársku prácu Jakuba Kováča (2007) a rozvíja kompiláciu o ďalšie dátové štruktúry. V prvej časti sú tieto dátové štruktúry, konkrétne union-find, lexikografický strom a sufixový strom, definované a konkrétne algoritmy popísané. V druhej časti sa pojednáva samotná vizualizácia a popisuje sa implementácia. Práca je prehľadom dátových štruktúr s priloženým Java appletom na CD ako pomôckou pri vizualizácií.
Kľúčové slová: vizualizácia, ADŠ, algoritmy a dátové štruktúry, union-find, stringológia, lexikografický strom, trie, suffix tree.

Abstract

Here should be an abstract in English language.

Predhovor

Toto je zaujímavá časť, ale ešte som ju u nikoho nevidel, tak asi pôjde preč. Tu si veľmi vylejem dušu na celý svet. :) Zatiaľ tu patrí poďakovanie najmä Marekovi Zemanovi, Michalovi Forišekovi, Kubovi Kováčovi a všetkým učiteľom, ktorí mi pomohli nadobunúť vedomosti, aby som mohol vypracovať túto prácu.

Obsah

Úvod

Dátové štruktúry a algoritmy tvoria základnú, prvotnú časť výučby informatiky. Vizualizácia algoritmov a dátových štruktúr je grafické znázornenie, ktoré abstrahuje od implementačných detajlov a reprezentácie v pamäti. Je teda vhodnou pomôckou pri výučbe i samoštúdiu.

Ukážka nášho softvéru *Gnarley Trees* je na obrázku 1. Tento projekt začal ako bakalárska práca Jakuba Kováča (Kováč 2007); v tomto článku popisujeme nové dátové štruktúry, ktoré sme vizualizovali a nové funkcie a vylepšenia, ktoré sme doplnili.

Z vyvažovaných stromov vizualizujeme *B^+-strom*, *strom s prstom* a *strom s reverzami*, z háld to sú *d-árna halda*, *ľavicová halda*, *skew halda* a *párovacia halda*. Taktiež vizualizujeme aj *union-find problém* a *písmenkový strom (trie)*.

Okrem vizualizácie softvér prerábame a neustále vylepšujeme. Doplnili sme ho o históriu krokov a operácií, jednoduchšie ovládanie a veľa ďalších funkcií. Softvér je celý v slovenčine aj angličtine a je implementovaný v jazyku `Java`. Dostupný je na stránke `http://people.ksp.sk/~kuko/gnarley-trees` vo forme appletov s jednotlivými dátovými štruktúrami, a tiež vo forme samostatného programu, ktorý obsahuje všetky dátové štruktúry a je určený na používanie offline.

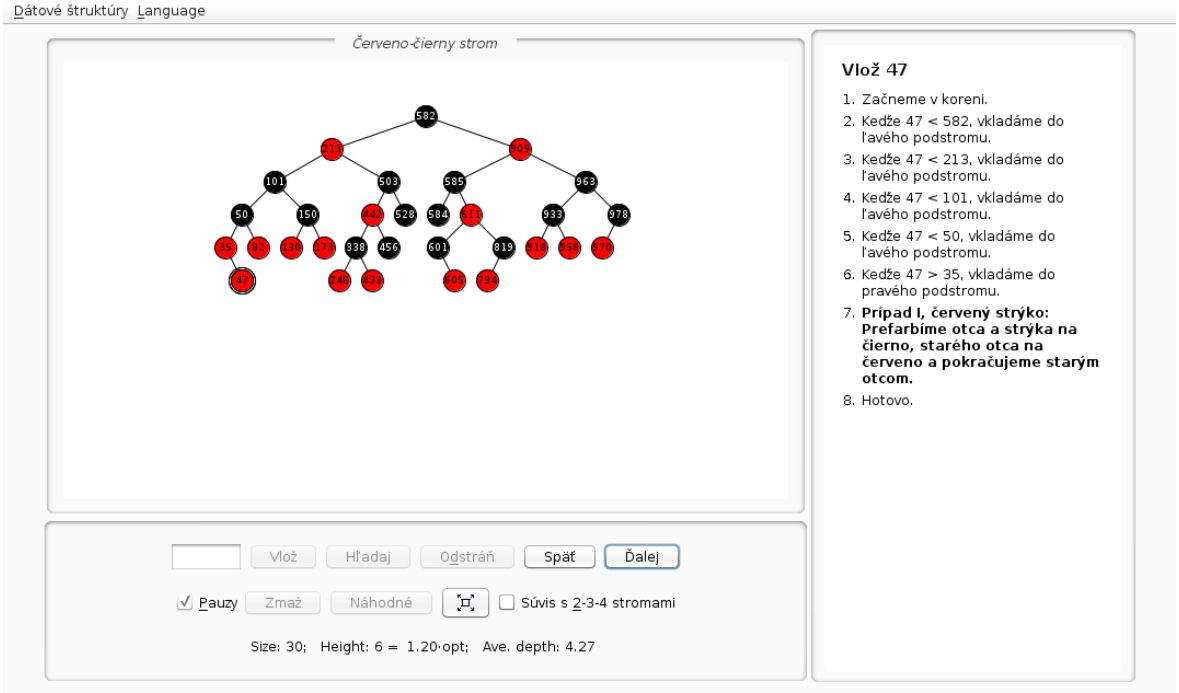
Našou snahou je vytvoriť kvalitný softvér nezávislý od operačného systému, ktorý bude vyhovovať ako pomôcka pri výučbe ako aj pri samoštúdiu a bude voľne prístupný.

Predchádzajúci výskum v oblasti pedagogiky zatiaľ nedokázal úplne preukázať pedagogickú efektívnosť vizualizácií (Shaffer a spol. 2010), avšak viacero štúdií potvrdilo zvýšený záujem a zapojenosť študentov (C.D. Hundhausen, Douglas a Stasko 2002; Naps a spol. 2002).

Rozmach vizualizácie algoritmov priniesla najmä `Java` a jej fungovanie bez viazanosti na konkrétny operačný systém. Kvalita iných existujúcich vizualizácií sa líši a keďže ide o ľahko naprogramovateľné programy, je ich veľa a sú pomerne nekvalitné (Shaffer a spol. 2010). Zbieraním a analyzovaním kvality sa venuje skupina AlgoViz (`http://algoviz.org/`).

Zaujímavé je pozorovanie, že určovanie si vlastného tempa pri vizualizácií je veľká pomôcka. Naopak, ukazovanie pseudokódu alebo nemožnosť určenia si vlastného tempa (napríklad animácia bez možnosti pozastavenia), takmer žiadne zlepšenie neprináša (Saraiya a spol. 2004; Shaffer a spol. 2010).

Zvyšok článku je organizovaný nasledovne: V sekcií 2 popisujeme implementované vylepšenia týkajúce sa vizualizácie, grafiky a ovládania, v sekciách 3 až 6 nové dátové štruktúry,



Obr. 1: *Softvér Gnarley Trees.* V ovládacom paneli dole môže užívateľ zvoliť operáciu a vstupnú hodnotu a sledovať priebeh algoritmu (momentálne vkladanie prvku 47). Užívateľ postupuje vlastným tempom pomocou tlačidiel „Ďalej“ alebo „Späť“. Vpravo je popis vykonávaných krokov; kliknutím na konkrétny krok v histórii sa môže užívateľ vrátiť.

ktoré sme implementovali a vizualizovali: vyvážené stromy, haldy, union-find a písmenkové stromy.

Kapitola 1

Union-find problém

V niektorých aplikáciach potrebujeme udržiavať prvky rozdelené do skupín (disjunktných množín), pričom skupiny sa môžu zlučovať a my potrebujeme pre daný prvok efektívne zistiť, do ktorej skupiny patrí. Predpokladáme, že každá množina S je jednoznačne určená jedným svojim zástupcom $x \in S$ a potrebujeme implementovať nasledovné tri operácie:

- *makeset*(x) – vytvorí novú množinu $S = \{x\}$ s jedným prvkom;
- *union*(x, y) – ak x, y sú zástupcovia množín S a T, *union* vytvorí novú množinu $S \cup T$, pričom S aj T zmaže. Zástupcom novej množiny $S \cup T$ je x alebo y.
- *find*(w) – nájde zástupcu množiny, v ktorej sa prvok x nachádza.

1.1 Použitie

Union-find sa dá použiť na reprezentáciu neorientovaného grafu, do ktorého pridávame hrany a odpovedáme na otázku "sú dané dva vrcholy spojené nejakou cestou?"(t. j. sú v rovnakom komponente súvislosti?). Medzi najznámejšie aplikácie patria Kruskalov algoritmus na nájdenie najlacnejšej kostry (Kruskal 1956) a unifikácia (Knight 1989).

Gilbert, Ng a Peyton (1994) ukázali, ako sa dá union-find použiť pri Choleského dekompozícii riedkych matíc. Autori navrhli efektívny algoritmus, ktorý zistí počet nenulových prvkov v každom riadku a stĺpci výslednej matice, čo slúži na efektívnu alokáciu pamäte.

Pre offline verziu úlohy, kde sú všetky operácie dopredu známe, Gabow a R. Tarjan (1985) navrhli lineárny algoritmus. Článok obsahuje tiež viacero aplikácií v teoretickej informatike.

1.2 Popis

Dátová štruktúra *union-find* sa reprezentuje ako les, kde každý strom zodpovedá jednej množine a korene stromov sú zástupcovia množín. Pri implementácií si stačí pre každý prvok x udržiavať smerník $p(x)$ na jeho otca (pre koreň je $p(x) = \texttt{NULL}$).

Operácia *makeset*(x) teda vytvorí nový prvok x a nastaví $p(x) = \texttt{NULL}$.

Operáciu *find*(w) vykonáme tak, že budeme sledovať cestu po smerníkoch, až kým nenájdeme zástupcu.

Operáciu *union*(x, y) ide najjednoduchšie vykonať tak, že presmerujeme smerník $p(y)$ na prvok x, teda $p(y) \leftarrow x$. Môžeme ľahko pozorovať, že takýto *naivný* spôsob je neefektívny, lebo nám operácia *find*(w) v najhoršom prípade, na n prvkoch, trvá $\Omega(n)$ krokov.

Existujú dva prístupy ako zlepšiť operácie a tým aj zrýchliť ich vykonanie. Sú to: heuristika *spájanie podľa ranku* a rôzne heuristiky na *kompresiu cesty*.

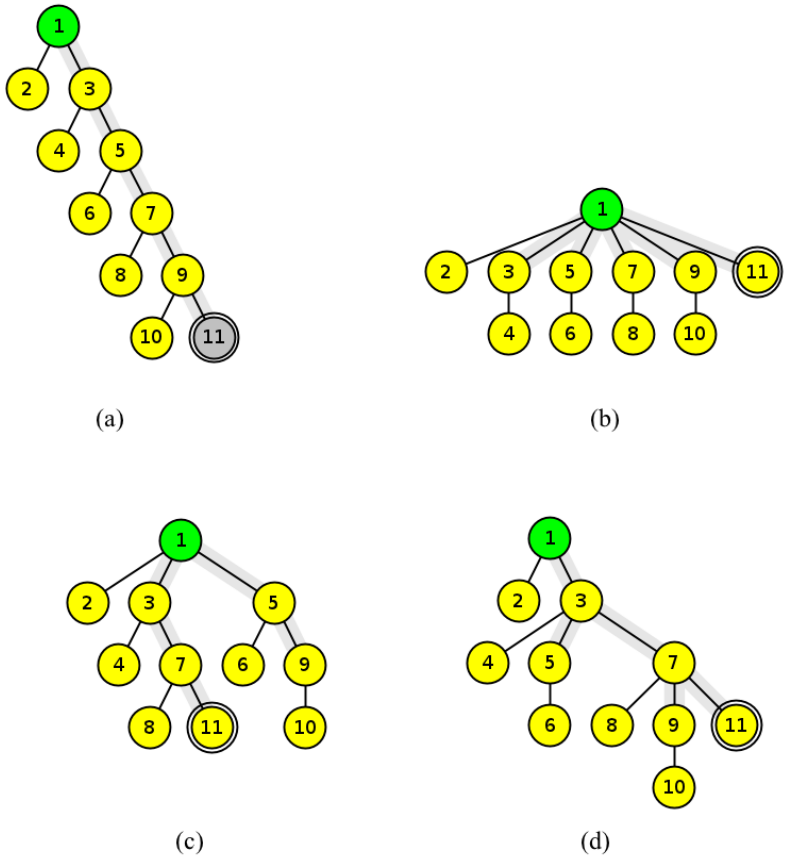
1.3 Heuristika na spájanie

Prvá heuristika pre každý vrchol x udržuje hodnotu *rank*(x), ktorá určuje najväčšiu možnú hĺbku podstromu s koreňom x. Pri operácií *makeset*(x) zadefinujeme *rank*$(x) = 1$. Pri operácií *union*(x, y) porovnáme *rank*(x) a *rank*(y) a vždy napojíme strom s menším rankom pod strom s väčším rankom. Ak majú oba stromy rovnaký rank, napojíme povedzme x pod y a *rank*(y) zvýšime o 1.

1.4 Heuristiky na kompresiu cesty

Druhou heuristikou je kompresia cesty. Algoritmov na efektívnu kompresiu cesty je veľa (R. E. Tarjan a Jan van Leeuwen 1984), tu popíšeme tie najefektívnejšie. Prvou z nich je *úplná kompresia* (Hopcroft a Ullman 1973). Pri vykonávaní operácie *find*(w), po tom, ako nájdeme zástupcu, napojíme všetky vrcholy po ceste priamo pod koreň (obr. 1.1(b)). Toto síce trochu spomalí prvé hľadanie, ale výrazne zrýchli ďalšie hľadania pre všetky prvky na ceste ku koreňu. Druhou heuristikou je *delenie cesty* (J. Leeuwen a Weide 1977). Pri vykonávaní operácie *find*(w) pripojíme každý vrchol v ceste od vrcholu x po koreň stromu na otca jeho otca (obr. 1.1(c)). Treťou heuristikou je *pólenie cesty* (J. Leeuwen a Weide 1977). Pri vykonávaní operácie *find*(w) pripojíme každý druhý vrchol v ceste od vrcholu x po koreň stromu na otca jeho otca (obr. 1.1(d)).

Časová zložitosť union-findu záleží od toho, koľko prvkov je v množinách a koľko je operácií celkovo vykonaných operácií. Všetky uvedené spôsoby ako vykonať operáciu *find*(w) sa dajú použiť s obomi realizáciami operácie *union*(x, y). Počet prvkov označme n a počet operácií m. V praxi je zvyčajne počet operácií oveľa väčší ako počet prvkov. Pri tomto predpoklade ($m \geq n$) je pri použití spájania podľa ranku časová zložitosť pre algoritmus bez kompresie $\Theta(m \log n)$ a pre všetky tri uvedené typy kompresií $\Theta(m\,\alpha(m, n))$ (R. E. Tarjan a Jan van Leeuwen 1984).



Obr. 1.1: *Kompresia cesty z vrcholu 11 do koreňa.* Cesta je vyznačená šedou. (a) Pred vykonaním kompresie. Pri úplnej kompresii (b) sa všetky vrcholy napoja na zástupcu. Pri delení cesty (c) a pólení cesty (d) sa cesta skráti približne na polovicu.

1.5 Vizualizácia

Dátovú štruktúru union-find sme vizualizovali ako les. Pre názorné oddelenie množín sme si zvolili pravidlo, ktoré zakazovalo vykresliť vrchol napravo od najľavejšieho vrcholu a naľavo od napravejšieho vrcholu inej množiny. Jednotlivé množiny sme už vykreslovali tesným Walkerovým algoritmom (Walker II 1990). Vizualizácia poskytuje všetky vyššie spomínané heuristiky a aj tlačidlo na vykonanie viacerých náhodných spojení naraz. Toto je užitočné, keď chce užívateľ vidieť, ako dátová štruktúra vyzerá, po vykonaní veľa operácií.

Kapitola 2

Písmenkový strom

2.1 Písmenkový strom

Písmenkový strom reprezentuje množinu slov. Oproti binárnym vyhľadávacím stromom je hlavný rozdiel v tom, že kľúče nie sú uložené vo vrcholoch, ale samotná poloha v strome určuje kľúč (slovo).

2.2 Popis

Písmenkový strom je *zakorenený strom*, v ktorom každá hrana obsahuje práve jeden znak z abecedy alebo *ukončovací znak*. Teda, každá cesta z koreňa do listu so znakmi $w_1, w_2, \ldots, w_n$, \$ prirodzene zodpovedá slovu $w = w_1 w_2 \cdots w_n$. *Ukončovací znak* je ľubovoľný, dopredu dohodnutý symbol, ktorý sa v abecede nenachádza (napr. \$).

Písmenkový strom je *asociatívne pole (slovník)*, čiže poskytuje tieto tri operácie:

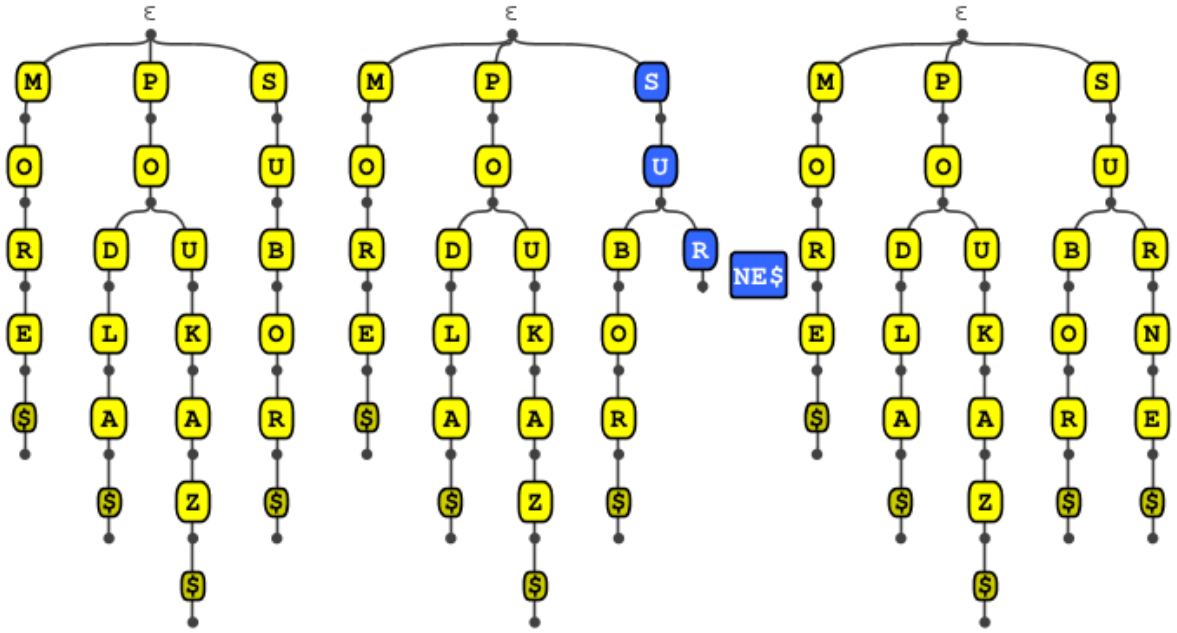
- *insert*(w) – pridá do stromu slovo w;
- *find*(w) – zistí, či sa v strome slovo w nachádza;
- *delete*(w) – odstráni zo stromu slovo w.

Všetky operácie začínajú v koreni a ku slovu pridávajú ukončovací znak, teda pracujú s reťazcom w\$.

Operácia *insert*(w) vloží do stromu vstupný reťazec tak, že z reťazca číta znaky a prechádza po príslušných hranách. Ak hrana s daným symbolom neexistuje, pridá ju (pozri obr. 2.1).

Operácia *find*(w) sa spustí z koreňa podľa postupnosti znakov. Ak hrana, po ktorej sa má spustiť neexistuje, dané slovo sa v strome nenachádza. Ak prečíta celý vstupný reťazec, dané slovo sa v strome nachádza.

Operácia *delete*(w) najprv pomocou operácie *find*(w) zistí umiestnenie slova. Ak sa slovo v strome nachádza, algoritmus odstráni hranu s ukončovacím znakom a vrchol, ktorý bol na nej



Obr. 2.1: *Vloženie slova „*`SURNE`*“.* Začiatok slova „`SU`“ sa v strome nachádza, ešte treba pripojiť hrany so znakmi `R`, `N`, `E` a `$`.

zavesený. V tomto štádiu sa nám môže stať, že v strome ostane *mŕtva vetva* – nie je ukončená ukončovacím znakom. Pre fungovanie stromu to nevadí, všetky operácie by prebiehali správne, ale takto štruktúra zaberá zbytočne veľa miesta. Preto je dobré túto mŕtvu vetvu odstrániť (pozri obr. 2.2).

Všetky tri operácie majú časovú zložitosť $O(|w|)$, kde $|w|$ je dĺžka slova.

2.3 Použitie

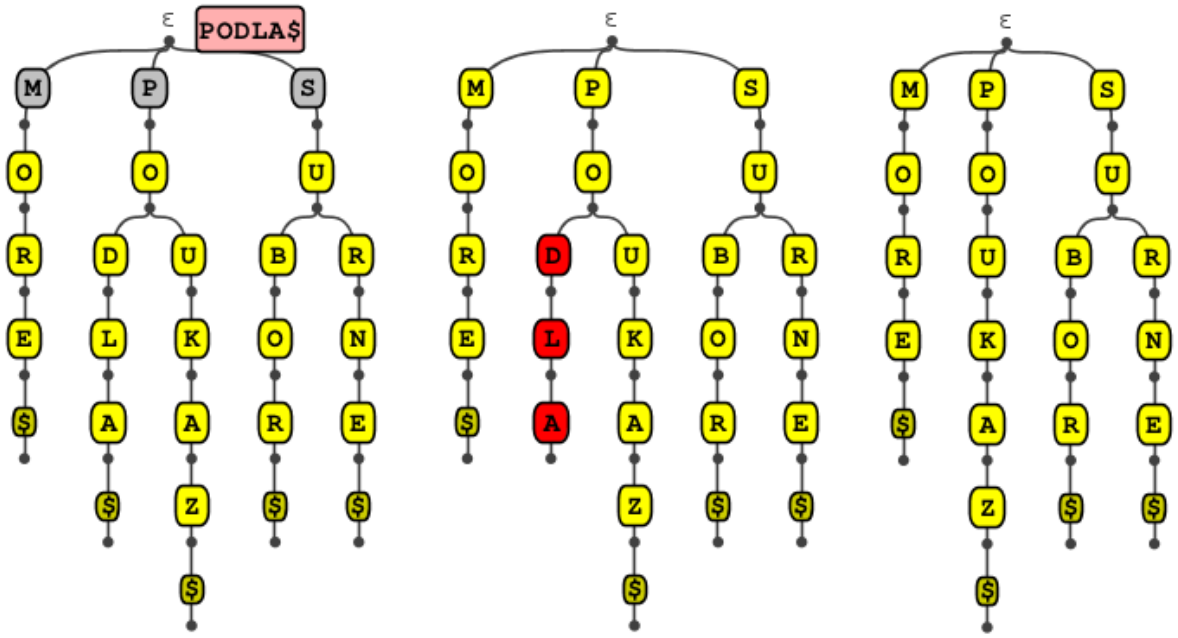
Prvýkrát navrhol písmenkový strom Fredkin (1960), ktorý používal názov *trie memory*[1], keďže išlo o spôsob udržiavania dát v pamäti.

Morrison (1968) navrhol písmenkový strom, v ktorom sa každá cesta bez vetvení skomprimuje do jedinej hrany (na hranách potom nie sú znaky, ale slová). Táto štruktúra je známa pod menom *PATRICIA* (tiež *radix tree*, resp. *radix trie*) a využíva sa napríklad v *routovacích tabuľkách* (Sklower 1991).

Písmenkový strom (tzv. *packed trie* alebo *hash trie*) sa používa napríklad v programe TEX na slabikovanie slov (Liang 1983). Pôvodný návrh (Fredkin 1960) ako uložiť trie do pamäte zaberal príliš veľa nevyužitého priestoru. Liang (1983) však navrhol, ako tieto nároky zmenšiť.

Písmenkové stromy sa podobajú na *konečné automaty*. Vznikli rôzne modifikácie stromov na automaty, ktorých hlavnou výhodou je, že v komprimovanej podobe spájajú nielen predpony, ale aj prípony slov a teda v slovách ľudských jazykov výrazne znižujú pamäťový priestor

[1] Z anglického re*trie*val – získanie.



Obr. 2.2: *Odstránenie slova „*PODLA*“.* Po odstránení $ nám v strome ostane nepotrebá prípona „DLA“ (mŕtva vetva), ktorá je vyznačená červenou.

potrebný na uchovanie dátovej štruktúry. Vďaka tomu sa využívajú na jazykovú korekciu, automatické dopĺňanie slov a podobne (Appel a Jacobson 1988; Claudio L. Lucchesi, Cľaudio L. Lucchesi a Kowaltowski 1992).

Ďalšie použitie písmenkového stromu je pri triedení zoznamu slov. Všetky slová sa pridajú do stromu a potom sa spraví *preorderový prechod* stromu. Túto myšlienku spracovali Ranjan Sinha a Justin Zobel (2004) a veľmi výrazne zrýchlil triedenie dlhých zoznamov slov. Neskôr tento algoritmus vylepšili R. Sinha, Ring a J. Zobel (2006). Kvôli tomu, ako algoritmus pracuje, sa nazýva *burstsort*.

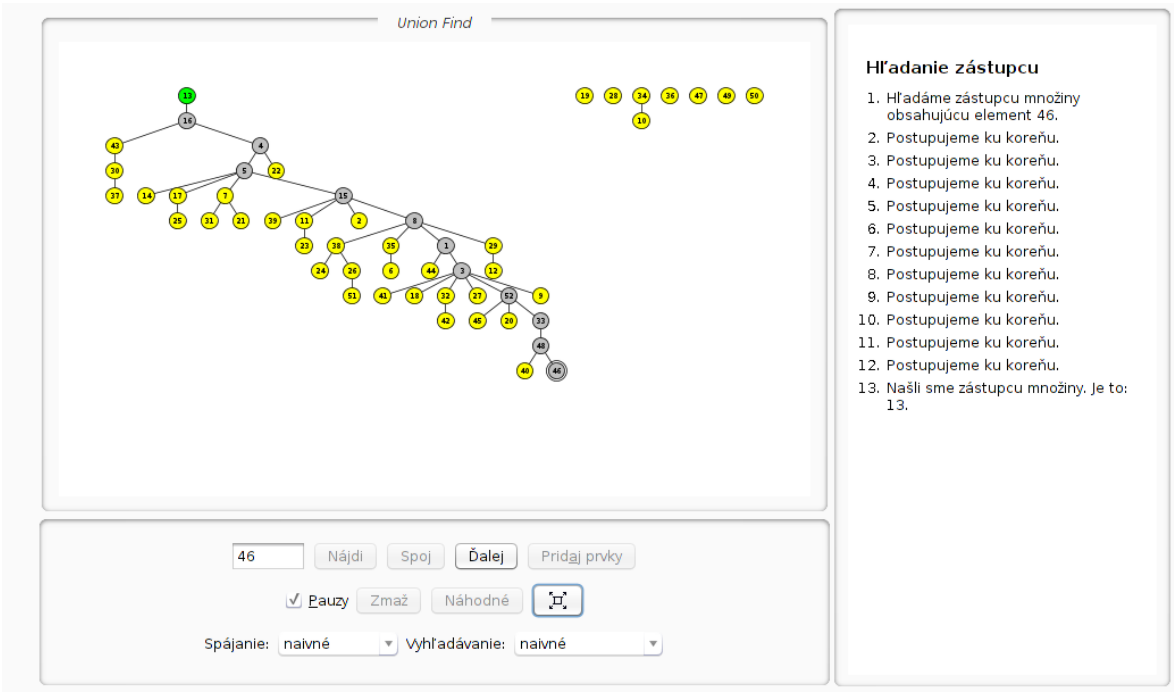
2.4 Vizualizácia

Pri vizualizácií písmenkového stromu sme použili Walkerov algoritumus pre úsporné rozloženie vrcholov v strome (Walker II 1990). Keď má vrchol viacej synov a hrany kreslíme priamo, tak vzniká nedostatok priestoru pre umiestnenie znakov na hrany. Preto sme sa rozhodli kresliť hrany zakrivene, podľa Bézierovej krivky určenej štyrmi bodmi. Vo vizualizácií sa dajú vložiť náhodné slová podľa momentálne nastaveného jazyka. Taktiež sa automaticky odstraňuje diakritika a interpunkcia, takže sa dá naraz vložiť súvislý text.

A na Obrázku 2.3 je väčší Komenský ako na titulke.

```
for(int i=0; i<n; i++)NULL
    cout << "loop";NULL
```

Obr. 2.3: Ján Ámos Logo.



Obr. 2.4: Union-find procedúra find - hľadanie zástupcu.

$$\cos^2\psi + \sin^2\psi = 1, \tag{2.1}$$

$$\cosh^2\omega - \sinh^2\omega = 1. \tag{2.2}$$

$$\square\square x[(Fx\square Gx)\square\square Hx] \tag{2.3}$$

$$\square\neg\square y\square x[x\square y\square x\square x] \tag{2.4}$$

$$\square x\square(y\square z)\neq(x\square y)\square(x\square z) \tag{2.5}$$

$$\square[[]] = {}_0\square\square() \tag{2.6}$$

$$\square[(\square x)x]\square\square\square(\square x)[x\square(\square z)(z\square x = z)\square x]\square \tag{2.7}$$

$$\square(P\square Q)\square(\square P\square\square Q) \tag{2.8}$$

⑳①

Záver

Tu by malo byť niečo v zmysle, že som to strašne nestihol a vymenované všetko, čo by som chcel dorobiť?

Literatúra

Appel, A. W. a G. J. Jacobson (1988). „The world's fastest Scrabble program". V: *Communications of the ACM* 31.5, str. 572–578.

Fredkin, Edward (sep. 1960). „Trie memory". V: *Commun. ACM* 3.9, str. 490–499. ISSN: 0001-0782. DOI: 10.1145/367390.367400. URL: http://doi.acm.org/10.1145/367390.367400.

Gabow, H.N. a R.E. Tarjan (1985). „A linear-time algorithm for a special case of disjoint set union". V: *Journal of computer and system sciences* 30.2, str. 209–221.

Gilbert, J.R., E.G. Ng a B.W. Peyton (1994). „An efficient algorithm to compute row and column counts for sparse Cholesky factorization". V: *SIAM Journal on Matrix Analysis and Applications* 15, str. 1075.

Hopcroft, John E. a Jeffrey D. Ullman (1973). „Set Merging Algorithms". V: *SIAM J. Comput.* 2.4, str. 294–303. URL: http://dx.doi.org/10.1137/0202024.

Hundhausen, C.D., S.A. Douglas a J.T. Stasko (2002). „A meta-study of algorithm visualization effectiveness". V: *Journal of Visual Languages & Computing* 13.3, str. 259–290.

Knight, Kevin (mar. 1989). „Unification: a multidisciplinary survey". V: *ACM Comput. Surv.* 21.1, str. 93–124. ISSN: 0360-0300. DOI: 10.1145/62029.62030. URL: http://doi.acm.org/10.1145/62029.62030.

Kováč, Jakub (2007). „Vyhľadávacie stromy a ich vizualizácia". Bakalárska práca, Univerzita Komenského v Bratislave.

Kruskal, Joseph B (1956). „On the shortest spanning subtree of a graph and the traveling salesman problem". V: *Proceedings of the American Mathematical Society* 7.1, str. 48–50. URL: http://www.ams.org/journals/proc/1956-007-01/S0002-9939-1956-0078686-7/S0002-9939-1956-0078686-7.pdf.

Leeuwen, J. a Th.P. van der Weide (sep. 1977). *Alternative Path Compression Rules*. Tech. spr. An outline of the results were presented at the Fachtagung on Algorithms and Complexity Theory, Oberwolfach, Oct 1977. University of Utrecht, The Netherlands.

Liang, F. M. (1983). „Word hy-phen-a-tion by com-put-er". Diz práca. Stanford, CA 94305: Stanford University.

Lucchesi, Claudio L., Cl'audio L. Lucchesi a Tomasz Kowaltowski (1992). *Applications of Finite Automata Representing Large Vocabularies*.

Morrison, Donald R. (okt. 1968). „PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric“. V: *J. ACM* 15.4, str. 514–534. ISSN: 0004-5411. DOI: 10.1145/321479.321481. URL: http://doi.acm.org/10.1145/321479.321481.

Naps, T.L. a spol. (2002). „Exploring the role of visualization and engagement in computer science education“. V: *ACM SIGCSE Bulletin*. Č. 35. 2. ACM, str. 131–152.

Saraiya, Purvi a spol. (2004). „Effective features of algorithm visualizations“. V: *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. SIGCSE '04. New York, NY, USA: ACM, str. 382–386. ISBN: 1-58113-798-2. DOI: http://doi.acm.org/10.1145/971300.971432. URL: http://doi.acm.org/10.1145/971300.971432.

Shaffer, Clifford A. a spol. (aug. 2010). „Algorithm Visualization: The State of the Field“. V: *Trans. Comput. Educ.* 10 (3), 9:1–9:22. ISSN: 1946-6226. DOI: http://doi.acm.org/10.1145/1821996.1821997. URL: http://doi.acm.org/10.1145/1821996.1821997.

Sinha, Ranjan a Justin Zobel (dec. 2004). „Cache-conscious sorting of large sets of strings with dynamic tries“. V: *J. Exp. Algorithmics* 9. ISSN: 1084-6654. DOI: 10.1145/1005813.1041517. URL: http://doi.acm.org/10.1145/1005813.1041517.

Sinha, R., D. Ring a J. Zobel (2006). „Cache-efficient String Sorting using Copying“. V: *J. Exp. Algorithmics* 11, str. 1.2. ISSN: 1084-6654.

Sklower, K. (1991). „A tree-based packet routing table for Berkeley Unix“. V: *Proceedings of the Winter 1991 USENIX Conference*, str. 93–104.

Tarjan, Robert E. a Jan van Leeuwen (mar. 1984). „Worst-case Analysis of Set Union Algorithms“. V: *J. ACM* 31.2, str. 245–281. ISSN: 0004-5411. DOI: 10.1145/62.2160. URL: http://doi.acm.org/10.1145/62.2160.

Walker II, John Q. (1990). „A node-positioning algorithm for general trees“. V: *Software: Practice and Experience* 20.7, str. 685–705.