Gnarley Trees

Katka Kotrlová* Pavol Lukča† Viktor Tomkovič‡ Tatiana Tóthová§

Školiteľ: Jakub Kováč[1]¶

Katedra informatiky, FMFI UK, Mlynská Dolina, 842 48 Bratislava

Abstrakt: V tomto článku prezentujeme našu prácu na projekte Gnarley Trees, ktorý začal Jakub Kováč ako svoju bakalársku prácu. Gnarley Trees je projekt, ktorý má dve časti. Prvá časť sa zaoberá kompiláciou dátových štruktúr, ktoré majú stromovú štruktúru, ich popisom a popisom ich hlavných výhod a nevýhod oproti iným dátovým štruktúram. Druhá časť sa zaoberá ich vizualizáciou a vizualizáciou vybraných algoritmoch na týchto štruktúrach.

Kľúčové slová: Gnarley Trees, vizualizácia, algoritmy a dátové štruktúry

1 Úvod

Ako ľudia so záujmom o dátové štruktúry sme sa rozhodli pomôcť vybudovať dobrý softvér na vizualizáciu algoritmov a dátových štruktúr a obohatiť kompiláciu Jakuba Kováča (Kováč, 2007) o ďalšie dátové štruktúry. Vizualizujeme rôznorodé dátové štruktúry. Z binárnych vyvažovaných stromov to sú *finger tree* a *reversal tree*, z háld to sú *d-nárna halda*, *ľavicová halda*, *skew halda* a *párovacia halda*. Taktiež vizualizujeme aj *problém disjuktných množín (union-find problém)* a *písmenkový strom (trie)*.

Okrem vizualizácie prerábame softvér, doplnili sme ho o históriu krokov a operácií, jednoduchšie ovládanie a veľa iných vecí, zlepšujúcich celkový dojem. Softvér je celý v slovenčine a angličtine a je implementovaný v jazyku `Java`.

1.1 Vizualizácia

Dátové štruktúry a algoritmy tvoria základnú, prvotnú časť výučby informatiky. Vizualizácia algoritmov a dátových štruktúr je grafické znázornenie, ktoré abstrahuje spôsob ako algoritmus a dátové štruktúry pracujú od ich vnútornej reprezentácie a umiestnení v pamäti. Je teda vyhľadávaná a všeobecne rozšírená pomôcka pri výučbe. Výsledky výskumov ohľadne jej efektívnosti sa líšia, od stavu „nezaznamenali sme výrazné zlepšenie“ po „je viditeľné zlepšenie“. (Shaffer et al., 2010)

Rozmach vizualizačných algoritmov priniesla najmä Java a jej fungovanie bez viazanosti na konkrétny operačný systém. Kvalita vizualizácií sa líši a keďže ide o ľahko naprogramovateľné programy, je ich veľa a sú pomerne nekvalitné. V takomto množstve je ťažké nájsť kvalitné vizualizácie. Zbieraním a analyzovaním kvality sa venuje skupina AlgoViz, ktorá už veľa rokov funguje na portále `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. (Shaffer et al., 2010; Saraiya et al., 2004)

Motivácia

Z vyššie uvedeného je jasné, že našou snahou je vytvoriť kvalitnú kompiláciu a softvér, ktorý bude nezávislý od operačného systému, bude vyhovovať ako pomôcka pri výučbe ako aj pri samoštúdiu a bude voľne prístupný a náležite propagovaný. Toto sú hlavné body, ktoré nespĺňa žiaden slovenský a len veľmi málo svetových vizualizačných softvérov. Našou hlavnou snahou je teda ponúknuť plnohodnotné prostredie pri učení.

2 Rozšírenie predošlej práce

jednotlive vizualizacie a implementovane ficurie co sa zmenilo od bakalarky? zoomovanie, komentare, tree layouty, historia a nove ds

3 Vyvážené stromy

uz boli, pribudli

*`katkinemail,ktorýchcezverejniť`
†`palyhomail`
‡`viktor.tomkovic@gmail.com`
§`tánička@inmail`
¶`hmm`

3.1 Finger tree

3.2 Reversal tree

Táňa, čiň sa!

4 Haldy

4.1 d-nárna halda

4.2 L'avicová halda

4.3 Skew halda

4.4 Párovacia halda

Katka, zase spíš?! [citacie]

5 Union-Find

Sú problémy, ktoré vyžadujú spájanie objektov do množín a množín navzájom a následné určovanie, do ktorej množiny objekt patrí. Od takejto *dátovej štruktúry pre disjunktné množiny* očakávame, že si bude udržiavať jednoznačného *zástupcu* každej množiny a bude poskytovať tieto tri oprácie:

- **makeset**(x) – vytvorí novú množinu s jedným prvkom, ktorý nepatrí do žiadnej inej množiny;
- **find**(x) – nájde zástupcu množiny, v ktorej sa prvok x nachádza;
- **union**(x,y) – vytvorí novú množinu, ktorá obsahuje všetky prvky v množinách, ktorých zástupcovia sú x a y. Tieto množiny zmaže. Ďalej vyberie nového zástupcu novej množiny. Pre jednoduchosť, táto operácia predpokladá, že x a y sú zástupcovia množín.

Vďaka dvom hlavným operáciam **find**(x) a **union**(x,y) je táto dátová štruktúra známejšia pod pojmom *Union-Find*, ktorý používame aj my. Medzi najznámejšie problémy, ktoré sa riešia pomocou Union-Find patria Kruskalov algoritmus na nájdenie najlacnejšej kostry (Kruskal, 1956) a unifikácia (Knight, 1989). Veľmi triviálne použitie je zistenie počtu komponentov v grafe. Existujú aj iné problémy z teórie grafov, napr. Tarjan (1979).

Vďaka častej asociácií objektov a spájania množín ako vrcholy a hrany grafu sa často dátová štruktúra abstraktne reprezentuje ako *les* – množina zakorenených stromov. Konkrétnou implementáciou potom býva pole objektov — vrcholov. Ku každému objektu sa musí udržiavať smerník $p(x)$ na otca v strome. Smerník zástupcu množiny zvyčajne ukazuje na seba ($p(x) = x$). V našej implementácií však smerník zástupcu množiny ukazuje na hodnotu `NULL`.

Operácia **makeset**(x) teda vytvorí nový prvok x a nastaví $p(x) =$ `NULL`. Operáciu **find**(x) 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 $p(y) = x$. Môžeme ľahko pozorovať, že takýto *naivný* spôsob je neefektívny, lebo nám operácia **find**(x) v najhoršom prípade, na n prvkoch, trvá $O(n)$ krokov.

Existujú dva prístupy ako zlepšiť operácie a tým aj zrýchliť ich vykonanie. Sú to: heuristika *union podľa ranku* a rôzne heuristiky na *kompresiu cesty*. Prvá heuristika pridáva ku algoritmom hodnotu $rank(x)$, ktorá bude určovať najväčšiu možnú hĺbku podstromu zakorenenú vrcholom x. V tom prípade pri operácií **makeset**(x) zadefinujeme $rank(x) = 0$. Pri operácií **union**(x,y) vždy porovnáme $rank(x)$ a $rank(y)$, aby sme zistili, ktorý zástupca predstavuje menší strom. Smerník tohto zástupcu potom napojíme na zástupcu s vyšším rankom. Zástupca novej množiny bude ten s vyšším rankom. Ak sú oba ranky rovnaké, vyberieme ľubovoľného zo zástupcov x a y, jeho rank zvýšime o jeden a smerník ostatného zástupcu bude ukazovať na tohto zástupcu. Zástupcom novej množiny bude vybratý zástupca.

Druhou heuristikou je kompresia cesty. Algoritmov na efektívnu kompresiu cesty je veľa (Tarjan and van Leeuwen, 1984). Tu popíšeme tie najefektívnejšie. Prvou z nich je *jednoduchá kompresia cesty* (Hopcroft and Ullman, 1973). Pri vykonávaní operácie **find**(x), po tom, ako nájdeme zástupcu množiny obsahujúcej prvok x, smerníky prvkov navštívených po ceste (včetne x) presmerujeme na zástupcu množiny. Toto síce spomalí prvé vykonávanie, ale výrazne zrýchli ďalšie hľadania. Druhou heuristikou je *delenie cesty* (Leeuwen and Weide, 1977). Pri vykonávaní operácie **find**(x) pripojíme každý vrchol[1] v ceste od vrcholu x po koreň stromu na otca jeho otca. Treťou heuristikou je *pólenie cesty* (Leeuwen and Weide, 1977). Pri vykonávaní operácie **find**(x) pripojíme každý druhý vrchol[2] v ceste od

[1] okrem koreňa a synov koreňa, keďže tie deda a otca resp. deda nemajú

[2] okrem koreňa a synov koreňa, keďže tie deda a otca resp. deda nemajú

vrcholu x po koreň stromu na otca jeho otca.

Podľa Galil and Italiano (1991) je najefektívnejšie pólenie cesty pred delením, ktoré spotrebuje zhruba dva krát viac smerníkov a jednoduchou kompresiou, ktorá vyžaduje dva behy.

6 Písmenkový strom

Friker, neobzeraj baby a pracuj! Nejaké citovateľné práce o trie?

7 História

ako sme ju do..
..robili.
Palyho umelecký opis.

8 Záver

work in progress; co sme spravili, preco sme lepsi, co este chceme/treba spravit, co je rozrobene Paly?

8.1 Príspevky autorov

Paly spravil to, Katka ono, Táňa zase chrastu a Friker si pospal pod stromom.

Poďakovanie

Autori by sa chceli poďakovať školiteľovi za veľa dobrých rád a odborné vedenie pri práci.

Literatúra

Aho, A. V. and Hopcroft, J. E. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Galil, Z. and Italiano, G. F. (1991). Data structures and algorithms for disjoint set union problems. *ACM Comput. Surv.*, 23(3):319–344.

Hopcroft, J. E. and Ullman, J. D. (1973). Set merging algorithms. *SIAM J. Comput.*, 2(4):294–303.

Knight, K. (1989). Unification: a multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124.

Kováč, J. (2007). Vyhľadávacie stromy a ich vizualizácia. Bakalárska práca.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.

Leeuwen, J. and Weide, T. v. d. (1977). Alternative Path Compression Rules. Technical report, University of Utrecht, The Netherlands. An outline of the results were presented at the Fachtagung on Algorithms and Complexity Theory, Oberwolfach, Oct 1977.

Saraiya, P., Shaffer, C. A., McCrickard, D. S., and North, C. (2004). Effective features of algorithm visualizations. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, SIGCSE '04, pages 382–386, New York, NY, USA. ACM.

Shaffer, Clifford A. a Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. H. (2010). Algorithm visualization: The state of the field. *Trans. Comput. Educ.*, 10:9:1–9:22.

Tarjan, R. E. (1979). Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715.

Tarjan, R. E. and van Leeuwen, J. (1984). Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281.

Yao, A. C. (1985). On the expected performance of path compression algorithms. *SIAM J. Comput.*, 14:129–133.