KAPITOLA 1

Vizualizácie

An algorithm must be seen
to be believed.

The Art of Computer Programming
Donald E. Knuth

Cieľom tejto práce je nielen popísať *cache-oblivious* pamäťový model a rôzne dátové štruktúry v ňom, ale aj vytvoriť ich vizualizácie. Tie majú slúžiť na edukačné účely pre študentov (a učiteľov) a pomáhať pri pochopení ich fungovania.

Výsledkom práce sú vizualizácie demonštrujúce dátové štruktúry popísané v predchádzajúcich sekciách: *van Emde Boas* usporiadanie (sekcia **??**) v statickom binárnom vyhľadávacom strome, usporiadané pole (**??**) a dynamický b-strom (**??**). Súčasťou je tiež simulácia *cache* (sekcia **??**) s možnosťou voľby parametrov B a M - veľkosť bloku a celková veľkosť.

Existujuce? nic nie je...

1.1 Gnarley trees

Tieto vizualizácie sú implementované ako rozšírenie programu *Gnarley trees*, ktorý vznikol ako súčasť bakalárskej práca Jakuba Kováča [13]. Tento nástroj na vizualizáciu (prevažne stromových) dátových štruktúr bol následne v bakalárskych prácach [12, 14, 16] a ročníkových projektoch rozšírený o mnohé ďalšie dátové štruktúry a v súčastnosti podporuje desiatky štruktúr, ako napríklad červeno-čierne, sufixové a intervalové stromy, *union-find*, haldy a mnohé ďalšie.

1.1.1 Spustenie

Súčasťou práce je priložené CD obsahujúce zdrojový kód tohto programu a tiež skompilovanú verziu. Aplikácia na spustenie požaduje *JVM*[1]. Spustenie je možné vykonať spustením súborov `start.sh` prípadne `start.bat`, podla operačného systému. Dostupné sú tiež individuálne vizualizácie vo forme *Java appletov*, ktoré je možné spustiť otvorením súboru `index.html` v preferovanom internetovom prehliadači.

Obsah tejto CD prílohy je tiež dostupný na . url

1.1.2 Funkcionalita

Program umožňuje užívateľom zobrazovať tieto štruktúry a manipulovať s nimi. Všetky operácie sú rozložené na malé, jednoduché kroky a každý je vysvetlený keď sa vykonáva. Je možné posúvať sa po krokoch dopredu, ale aj vracať sa dozadu, a teda sa dá kedykoľvek vrátiť až k počiatočnému stavu. Toto je dôležité pri experimentovaní s danou štruktúrou, kedy dve rôzne operácie (alebo jedna operácia s dvoma rôznymi parametrami) spôsobia rôzne správanie a výsledky. Užívateľ má takto možnosť jednoducho sa po vykonaní prvej operácie vrátiť do predošlého stavu a preskúmať správanie druhej z nich.

Celý program je taktiež dvojjazyčný - je možné prepnúť medzi angličtinou a slovenčinou, čo umožňuje širšie použitie týchto vizualizácií.

1.1.3 Prehľad programu

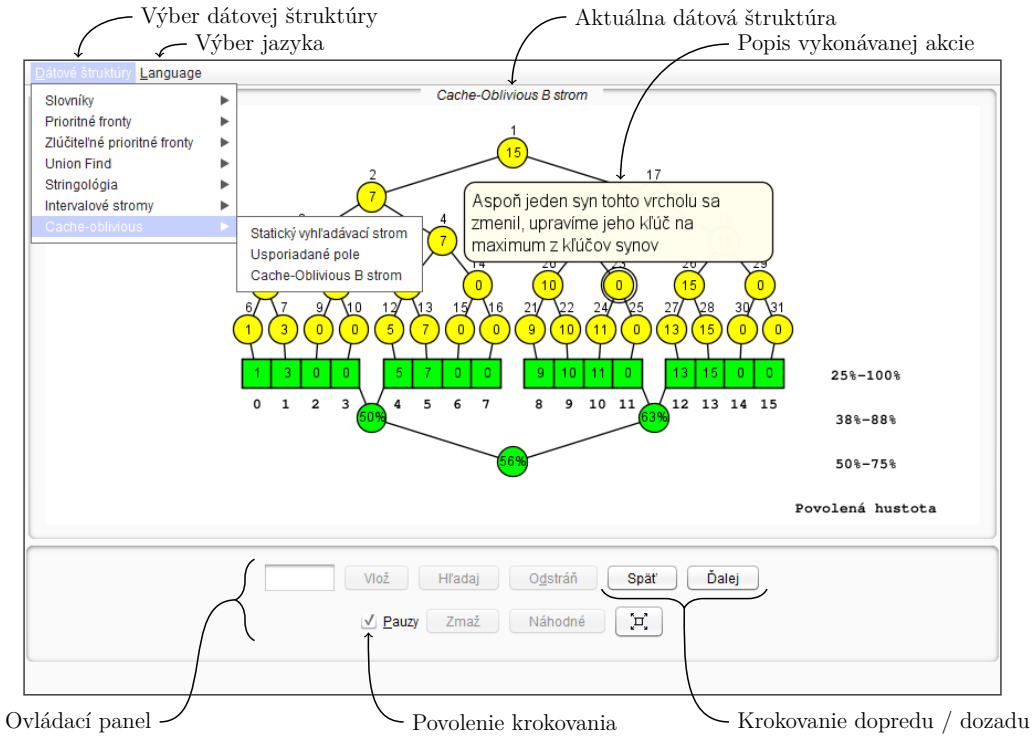
Program sa skladá z troch hlavných častí (obrázok 1.1). Najvrchnejšia časť okna tvorí hlavné menu, v ktorej môžeme voliť dátové štruktúry a prepínať jazyk rozhrania. Dátové štruktúry sú pre prehľadnosť rozdelené do niekoľkých kategórií. Tie popísané a implementované v tejto práci sa nachádzajú v kategórii `Cache-oblivious`.

V spodnej časti okna sa nachádza ovládací panel, ktorý obsahuje vstupné pole pre hodnotu, ktorú chceme vyhľadať alebo vložiť a tlačidlá na vykonanie týchto akcií. Ďalej obsahuje tlačidlá na prechod do ďalšieho kroku a návrat do predchádzajúceho stavu s možnosťou toto krokovanie vypnúť.

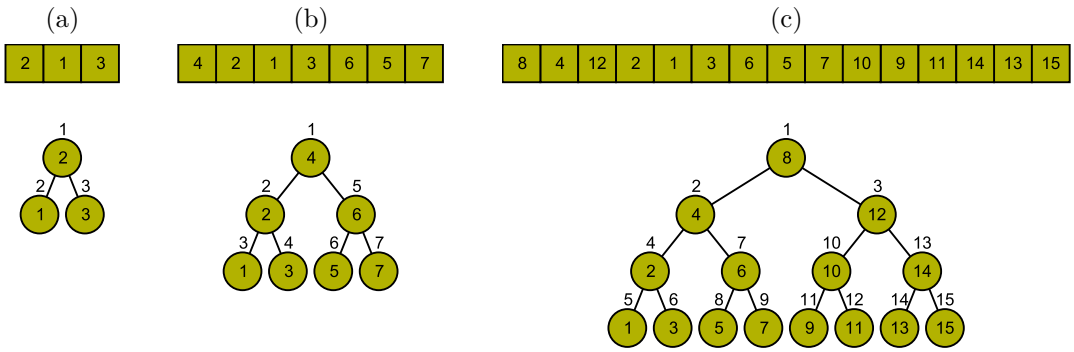
Najväčšia, prostredná časť okna zobrazuje vizualizáciu aktuálnej dátovej štruktúry. Toto zobrazenie je vektorové a je možné ho posúvať, približovať a oddialovať. Zároveň sa tu zobrazujú informácie o aktuálne vykonávanej akcií (ak je povolené krokovanie) a ďalšie vizualizačné prvky ako šípky alebo význačné vrcholy.

Podrobnejší popis užívateľského rozhrania a návod na používanie sa nachádza v bakalárskej práci Jakuba Kováča [13] v siedmej kapitole.

[1] *Java Virtual Machine*, dostupné na `https://www.java.com/en/download/`



Obrázok 1.1: Užívateľské rozhranie počas operácie vkladania kľúča 10 do dynamického *cache-oblivious* B-stromu (sekcia **??**).

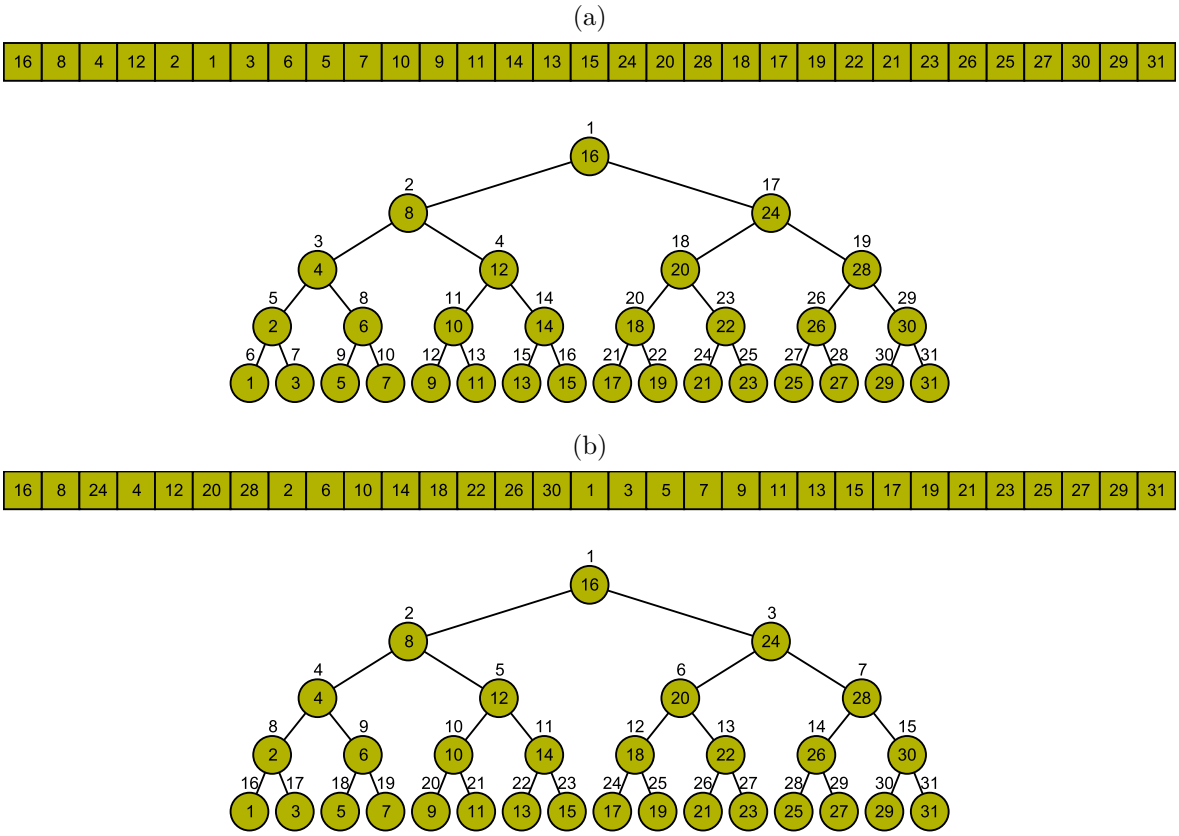


Obrázok 1.2: Statické stromy rôznych veľkostí (výšok) vo *van Emde Boas* usporiadaní.

1.2 Statický strom

Najjednoduchšou dátovou štruktúrou je statický vyhľadávací strom. Implementovali sme vytvorenie tohto stromu a jeho uloženie v pamäti. Je možné strom zväčšiť alebo zmenšiť podľa preferencii - ukážky stromov rôznych veľkostí sú na obrázku 1.2. Čísla vo vnútorných vrcholoch sú kľúče, čísla nad vrcholom určujú jeho pozíciu v pamäti. Obdĺžnik nad stromom reprezentuje uloženie tohto stromu v poli. Vnútorné čísla sú opäť kľúče, pričom sú zoradené podla svojich pozícii zľava (pozícia 1) doprava.

Medzi uložením vo *van Emde Boas* usporiadaní a klasickom *BFS* usporiadaní (ako v časti **??**) je možné prepínať. Zmenia sa pritom čísla udávajúce pozície vrcholov v pamäti a ich poradie v poli nad stromom. Rozdiel medzi týmito dvoma usporiadaniami



Obrázok 1.3: Rozdiel medzi *BFS* a *van Emde Boas* usporiadaním na strome výšky 5. Kľúče zostávajú rovnaké, líšia sa však ich pozície v pamäti reprezentované malými číslami nad vrcholmi a poľom nad koreňom.

vidieť na obrázku 1.3. Pozície sa zhodujú s obrázkom **??**.

1.2.1 Simulácia *cache*

Porovnanie týchto dvoch usporiadaní je rozšírené o simuláciu *cache*. Užívateľ si môže zvoliť parametre cache - počet vrcholov B, ktoré sa zmestia do jedného bloku a počet blokov $\frac{M}{B}$ v *cache*. Tiež je možné *cache* kedykoľvek vyprázdniť – odstrániť z nej všetky načítané bloky. Simulácia na výmenu stránok používa stratégiu *FIFO*, ktorá je popísaná v časti **??**.

Táto simulácia zároveň počíta počet prístupov k vrcholom pri vyhľadávaní a počet presunutých blokov do *cache*. V najhoršom prípade by tieto dve čísla boli rovnaké (ak treba každý vrchol načítať osobitne) avšak pri *cache* s blokmi veľkosti $B > 1$ a s *van Emde Boas* usporiadaním dochádza k podstatnému zlepšeniu - ušetreniu počtu presunutých blokov.

To môžeme vidieť na jednoduchom príklade, kedy v strome výšky 5 postupne vyhľadáme všetkých 16 kľúčov, ktoré sa nachádzajú v listoch tohto stromu. V oboch usporiadaniach bude počet prístupov rovnaký, avšak počet načítaní blokov z disku do *cache* je v tomto príklade pri *BFS* usporiadaní takmer dvakrát väčší ako pri *van Emde*

Boas usporiadaní. Obrázok 1.4 ukazuje stav týchto štatistík po nájdení posledného listu.

(a) Strom v *BFS* usporiadaní (b) Strom vo *van Emde Boas* usporiadaní

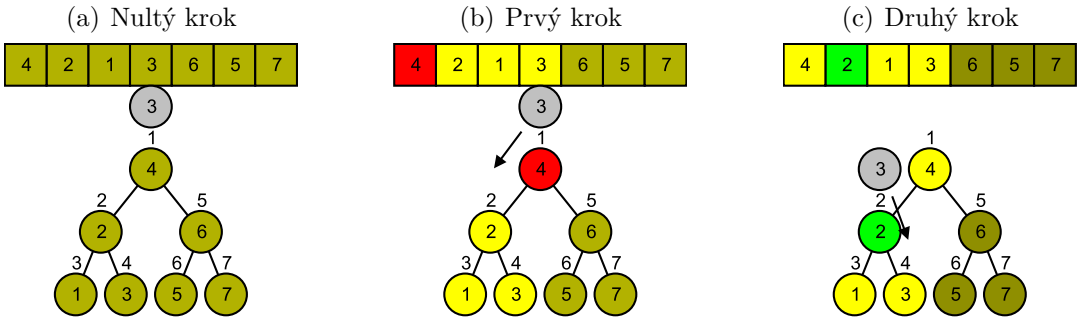
Obrázok 1.4: Porovnanie štatistík simulovanej *cache* po prechode listov $1, 3, \ldots, 31$ stromu výšky 5. Parametre cache sú v oboch prípadoch rovnaké ($B = 4$, $M = 2B = 8$), rozdiel je však v usporiadaní v pamäti.

Ako vizualizácia prítomnosti v *cache* slúži farba - vrcholy a položky poľa obsahujúce kľúče majú svetlejšiu farbu pozadia v prípade, že je daný blok v *cache* a tmavšiu ak je mimo. V strome je vďaka tomu ľahko vidieť, ktorá časť je načítaná a je možné ňou prechádzať bez ďalších presunov. V prípade *van Emde Boas* usporiadanie pôjde prevažne o časť podstromu aktuálne porovnávaného vrcholu, avšak pri klasickom usporiadaní to budú práve vrcholy mimo tohto podstromu, o ktorých už vieme, že nie sú pri vyhľadávaní potrebné.

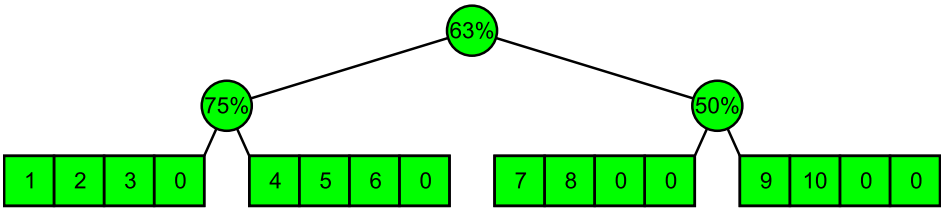
Pri krokovaní vyhľadávania je pri prístupe k vrcholu tiež použitá zelená alebo červená farba na jeho dočasné zafarbenie (podobne ako v časti **??**) podľa toho, či sa v danom momente v *cache* nachádzal (*cache hit*) alebo nie (*cache miss*). Ukážka tohto zafarbovania je na obrázku 1.5.

1.3 Usporiadané pole

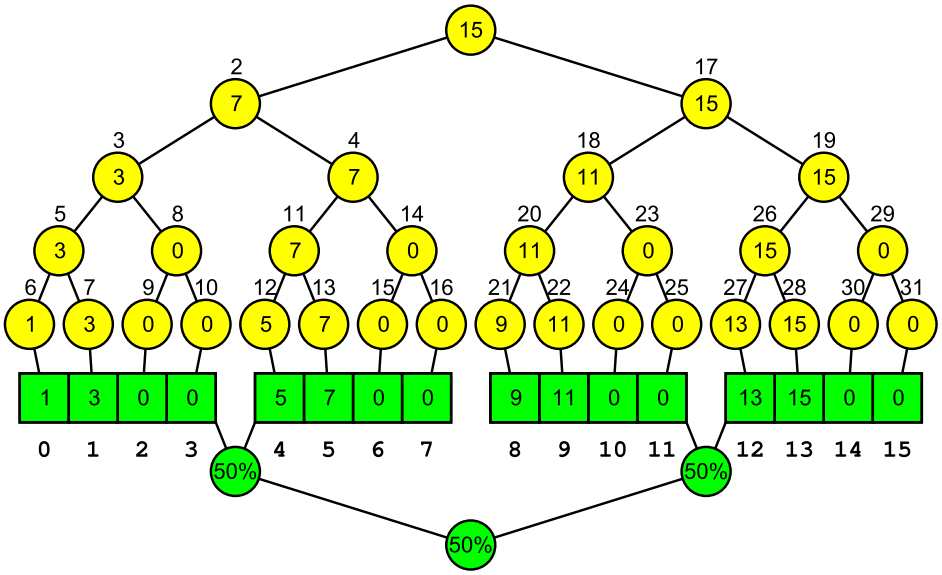
Ďalšou implementovanou vizualizáciou je usporiadané pole (obrázok 1.6), ktoré bolo popísané v časti **??**. Bloky, na ktoré je toto pole imaginárne rozdelené sú znázornené



Obrázok 1.5: Simulácia *cache* počas vyhľadávania kľúča 3. Pred prvým prístupom je cache prázdna. V prvok kroku načítame koreň, ktorý je označený červenou farbou (*cache miss*), keďže nebol v *cache*. Spolu s ním sa v jednom bloku načítali ďalšie vrcholy, ktoré su označené svetlejšou farbou. V druhom kroku je vrchol s kľúčom 2 označený zelenou farbou (*cache hit*), keďže už bol do *cache* načítaný v predchádzajúcom kroku.



Obrázok 1.6: Usporiadané pole obsahujúce hodnoty 1 až 10. Všetky vrcholu majú hustotu v hraniach hustoty. Hodnoty 0 reprezentujú prázdne pozície.



Obrázok 1.7: Vizualizácia dynamického *cache-oblivious* B-stromu.

tým, že sú od seba oddelené medzerou. Taktiež sa zobrazuje imaginárny strom nad týmito blokmi, ktorým sa pri vkladaní prechádza. Hodnoty vo vrcholoch reprezentujú hustotu (v percentách) intervalu v príslušnom podstrome. Farby (zelená alebo červená) vrcholov indikujú, či sa hustota daného vrcholu nachádza v hraniciach.

Táto vizualizácia podporuje vkladanie hodnoty na ľubovolnú pozíciu v poli. V prípade, že sa pole preplní, automaticky sa vytvorí nové, dvojnásobne väčšie. Po vložení hodnoty je tiež vyznačený interval, ktorý sa zmenil.

1.4 Dynamický strom

Vizualizácia dynamického stromu (časť **??**) vznikla spojením predchádzajúcich dvoch vizualizácií (obrázok 1.7). Horná časť je statický strom vo *van Emde Boas* usporiadaní (obrázok 1.2), dolná je usporiadané pole (obrázok 1.6), pričom strom hustôt je tu preklopený nadol, aby sa neprekrýval so statickým stromom. Hrany medzi nimi spájajú listy stromu s prvkami usporiadaného pola.

V tomto strome je možné vyhľadávať a vkladať do neho nové kľúče. Pri vkladaní prebehne najskôr vloženie do usporiadaného poľa rovnako, ako pri jeho samostatnej vi-

zualizácii. Následne sa aktualizujú kľúče statického stromu. V prípade zdvojnásobenia veľkosti statického poľa sa vytvorí nový, väčší statický strom.

1.5 Implementácia

?

Literatúra

[1] Aggarwal, Alok ; Vitter, Jeffrey u. a.: The input/output complexity of sorting and related problems. In: *Communications of the ACM* 31 (1988), Nr. 9, S. 1116–1127

[2] Bayer, Rudolf: Binary B-trees for Virtual Memory. In: *Proceedings of the 1971 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. New York, NY, USA : ACM, 1971 (SIGFIDET '71), 219–235

[3] Bender, Michael A. ; Demaine, Erik D. ; Farach-Colton, Martin: Cache-Oblivious B-Trees. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*. Redondo Beach, California, November 12–14 2000, S. 399–409

[4] Bender, Michael A. ; Demaine, Erik D. ; Farach-Colton, Martin: Cache-Oblivious B-Trees. In: *SIAM Journal on Computing* 35 (2005), Nr. 2, S. 341–358

[5] Bender, Michael A. ; Duan, Ziyang ; Iacono, John ; Wu, Jing: A locality-preserving cache-oblivious dynamic dictionary. In: *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms* Society for Industrial and Applied Mathematics, 2002, S. 29–38

[6] Demaine, Erik D.: Cache-Oblivious Algorithms and Data Structures. In: *Lecture Notes from the EEF Summer School on Massive Data Sets*. BRICS, University of Aarhus, Denmark, June 27–July 1 2002

[7] Drepper, Ulrich: What every programmer should know about memory. In: *Red Hat, Inc* 11 (2007)

[8] Frigo, Matteo ; Leiserson, Charles E. ; Prokop, Harald ; Ramachandran, Sridhar: Cache-oblivious algorithms. In: *Foundations of Computer Science, 1999. 40th Annual Symposium on* IEEE, 1999, S. 285–297

[9] Intel Corporation: *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014 (248966-029)

[10] Intel Corporation: *Intel® 64 and IA-32 Architectures Software Developer's Manual*. 2014 (325462-050US)

[11] Kasheff, Zardosht: *Cache-oblivious dynamic search trees*, Massachusetts Institute of Technology, Diss., 2004

[12] Kotrlová, Katarína: *Vizualizácia háld a intervalových stromov*, Univerzita Komenského v Bratislave, bakalárska práca, 2012

[13] Kováč, Jakub: *Vyhľadávacie stromy a ich vizualizácia*, Univerzita Komenského v Bratislave, bakalárska práca, 2007

[14] Lukča, Pavol: *Perzistentné dátové štruktúry a ich vizualizácia*, Univerzita Komenského v Bratislave, bakalárska práca, 2013

[15] Prokop, Harald: *Cache-oblivious algorithms*, Massachusetts Institute of Technology, Diss., 1999

[16] Tomkovič, Viktor: *Vizualizácia stromových dátových štruktúr*, Univerzita Komenského v Bratislave, bakalárska práca, 2012