

Datové struktury I

NTIN066

Jirka Fink

<https://ktiml.mff.cuni.cz/~fink/>

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Zimní semestr 2016/17
Poslední změna 13. února 2017

Licence: Creative Commons BY-NC-SA 4.0

Jirka Fink Datové struktury I 1

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Jirka Fink Datové struktury I 3

1. domácí úkol: Externí třídění

Stručné zadání

- Napsat program, který setřídí velký soubor čísel
- Celý vstup se nevejde do paměti RAM
- Všechny datové struktury i algoritmy si musíte naprogramovat sami, tj. nesmíte používat `std::vector`, `std::map`, `std::sort` a podobně
- Počet získaných bodů závisí na době běhu programu
- Programy testujte v Unixové laboratoři na Malé Straně, kde se k ostatním chovejte ohleduplně!
- Termín odevzdání: 30. 10. 2016
- Podrobnosti: <https://ktiml.mff.cuni.cz/~fink/>

Jirka Fink Datové struktury I 5

Amortizovaná analýza

Amortizovaná analýza

V amortizované analýze je čas potřebný k vykonání posloupnosti operací v datové struktuře průměrován počtem vykonaných operací, např.

	nejhorší případ	amortizovaná složitost
Inkrementace n -bitového čítače	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Dynamického pole	$\mathcal{O}(n)$	$\mathcal{O}(1)$

Metody

- Agregovaná analýza
- Účetní metoda
- Potenciální metoda

Jirka Fink Datové struktury I 7

Základní informace

Kontakt

E-mail fink@ktiml.mff.cuni.cz
Homepage <https://ktiml.mff.cuni.cz/~fink/>
Konzultace Individuální domluva

Podmínky zápočtu

Bude zadanych pět domácích úkolů po 100 bodech a k zápočtu musíte získat minimálně 350 bodů.

Jirka Fink Datové struktury I 2

Literatura

- A. Koubková, V. Koubek: Datové struktury I. MATFYZPRESS, Praha 2011.
- T. H. Cormen, C.E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms. MIT Press, 2009
- K. Mehlhorn: Data Structures and Algorithms I: Sorting and Searching. Springer-Verlag, Berlin, 1984
- D. P. Mehta, S. Sahní eds.: Handbook of Data Structures and Applications. Chapman & Hall/CRC, Computer and Information Series, 2005
- E. Demaine: Cache-Oblivious Algorithms and Data Structures. 2002.
- R. Pagh: Cuckoo Hashing for Undergraduates. Lecture note, 2006.
- M. Thorup: High Speed Hashing for Integers and Strings. lecture notes, 2014.
- M. Thorup: String hashing for linear probing (Sections 5.1-5.4). In Proc. 20th SODA, 655-664, 2009.

Jirka Fink Datové struktury I 4

Obsah

- 1 Amortizovaná analýza
 - Inkrementace binárního čítače
 - Dynamické pole
 - BB[α]-strom
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Jirka Fink Datové struktury I 6

Obsah

- 1 Amortizovaná analýza
 - Inkrementace binárního čítače
 - Dynamické pole
 - BB[α]-strom
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Jirka Fink Datové struktury I 8

Binární čítač

- Máme n -bitový čítač
- Při operaci Increment se poslední nulový bit změní na 1 a všechny následující jedničkové bity se změní na 0
- Počet změných bitů v nejhorším případě je n
- Kolik bitů se změní při k operacích Increment?

Agregovaná analýza

- Poslední bit se změní při každé operaci — tedy k -krát
- Předposlední bit se změní při každé druhé operaci — tedy v průměru $k/2$ -krát
- i -tý bit od konce se změní každých 2^i operací — tedy v průměru $k/2^i$ -krát
- Celkový průměrný počet změn bitů je $\sum_{i=0}^{n-1} k/2^i \leq k \sum_{i=0}^{\infty} 1/2^i = 2k$

- Toto je zásadní fakt amortizované analýzy. Potenciál je jako banka, do které můžeme uložit peníze (čas), jestliže operace byla levná (rychle provedená). Při drahých (dlouho trvajících) operacích musíme naopak z banky vybrat (snižít potenciál), abychom operaci zaplatili (stihli provést v amortizovaném čase). V amortizované analýze je cílem najít takovou potenciální funkci, že při rychle provedené operaci potenciál dostatečně vzroste ($\Phi' > \Phi$) a naopak při dlouho trvajících operacích potenciál neklesne příliš moc.

- Sečtení sumy $\sum_{i=1}^k (\Phi_{i-1} - \Phi_i) = \Phi_0 - \Phi_k$ se nazývá "telescopic cancellation". Poslední nerovnost plyne z faktů, že $\Phi_0 \leq n$ a $\Phi_k \geq 0$.

Dynamické pole

Dynamické pole

- Máme pole, do kterého přidáváme i mazeme prvky
- Počet prvků označíme n a velikost pole p
- Jestliže $p = n$ a máme přidat další prvek, tak velikost pole zdvojnásobíme
- Jestliže $p = 4n$ a máme smazat prvek, tak velikost pole zmenšíme na polovinu

Agregovaná analýza: Amortizovaná složitost

- Zkopírování celého pole trvá $\mathcal{O}(n)$
- Jestliže po realokaci pole máme n prvků, pak další realokace nastane nejdříve po $n/2$ operacích Insert nebo Delete
- Amortizovaná složitost je $\mathcal{O}(1)$

Agregovaná analýza: Celkový čas

- Necht k_i je počet operací mezi $(i-1)$ a i -tou realokací $\Rightarrow \sum_i k_i = k$
- Při i -té realokaci se kopíruje nejvýše $2k_i$ prvků pro $i \geq 2$
- Po prvé se kopíruje nejvýše $n_0 + k_1$ prvků, kde n_0 je počáteční počet prvků
- Celkový počet zkopírovaných prvků je nejvýše $n_0 + k_1 + \sum_{i \geq 2} 2k_i \leq n_0 + 2k$

Účetní metoda

- Změna jednoho bitu stojí jeden žeton a na každou operaci dostaneme dva žetony
- U každého jedničkového bitu si uschováme jeden žeton
- Při inkrementu máme vynulování jedničkových bitů předplaceno
- Oba žetony využijeme na jedinou změnu nulového bitu na jedničku a předplacení vynulování

Potenciální metoda

- Potenciál p_i i -tého bitu je 0, jestliže i -tý bit je nulový, jinak $p_i = 1$
- Potenciál čítače Φ je součet potenciálů všech bitů
- Potenciál před provedením operace je Φ a po provedení je Φ'
- Chceme dokázat: amortizovaný čas = skutečný čas + $(\Phi' - \Phi)$
- Necht j je vynulovaných jedniček při jedné operaci Increment
- Skutečný čas operace (počet změných bitů) je $j + 1$
- $\Phi' - \Phi = 1 - j$
- Amortizovaný čas = skutečný čas + $(\Phi' - \Phi) = j + 1 + (1 - j) = 2$

Inkrementace binárního čítače

Na zamýšlení

- Co přesně znamená celkový průměrný čas?
- Co přesně je amortizovaná složitost?
- Jaká je celková doba na provedení k operací?

Agregovaná složitost

- i -tý bit se při k operacích změní nejvýše $\lceil k/2^i \rceil$ -krát
- Celkový počet změn je nejvýše $\sum_{i=0}^{n-1} \lceil k/2^i \rceil \leq \sum_{i=0}^{n-1} (1 + k/2^i) \leq n + 2k$

Účetní metoda

- Na začátku dostaneme čítač a na každý jedničkový bit potřebujeme dát žeton
- Celkový počet spotřebovaných žetonů je nejvýše $n + 2k$

Potenciální metoda

- Potenciál po i -té operaci označme Φ_i a Φ_0 počáteční potenciál
- Amortizovaný čas i -té operace = skutečný čas i -té operace + $(\Phi_i - \Phi_{i-1})$
- Celkový čas = $\sum_{i=1}^k$ skutečný čas i -té operace = $k \cdot$ amortizovaný čas + $\sum_{i=1}^k (\Phi_{i-1} - \Phi_i) = k \cdot$ amortizovaný čas + $\Phi_0 - \Phi_k \leq 2k + n$

Obsah

- 1 Amortizovaná analýza
 - Inkrementace binárního čítače
 - Dynamické pole
 - BB[α]-strom
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- V analýze počítáme pouze čas na realokaci pole. Všechny ostatní činnost při operacích Insert i Delete trvá $\mathcal{O}(1)$ v nejhorším čase. Zajímá nás počet zkopírovaných prvků při realokaci, protože předpokládáme, že kopírování jednoho prvku trvá $\mathcal{O}(1)$.
- Po realokaci a zkopírování je nové pole z poloviny plné. Musíme tedy přidat n prvků nebo smazat $n/2$ prvků, aby došlo k dalšímu kopírování.

Potenciální metoda

- Uvažujeme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a lineární interpolaci v ostatních případech

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

- Změna potenciálu při jedné operaci bez realokace je $\Phi' - \Phi \leq 2$ ①
- Skutečný počet zkopírovaných prvků $+(\Phi' - \Phi) \leq$ amortizovaný počet = 2
- Celkový počet zkopírovaných prvků při k operacích je nejvýše $2k + \Phi_0 - \Phi_k \leq 2k + n_0$
- Celková čas k operací je $\mathcal{O}(n_0 + k)$

Obsah

- Amortizovaná analýza
 - Inkrementace binárního čítače
 - Dynamické pole
 - BB[α]-strom
- Splay strom
- (a,b)-strom a červeno-černý strom
- Halcy
- Cache-oblivious algorithms
- Hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

Váhy vyvážené stromy: BB[α]-stromBB[α]-strom (Nievergelt, Reingold [16])

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u ①
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů ②
- $\frac{1}{2} < \alpha < 1$ ③

Výška BB[α]-stromu

- Podstromy všech vnuků kořene mají nejvýše $\alpha^2 n$ vrcholů
- Podstromy všech vrcholů v i -té vrstvě mají nejvýše $\alpha^i n$ vrcholů
- $\alpha^i n \geq 1$ jen pro $i \leq \log_{\frac{1}{\alpha}}(n)$
- Výška BB[α]-stromu je $\Theta(\log n)$

Operace Build: Vytvoření BB[α]-stromu ze seříděného pole

- Prostřední prvek dáme do kořene
- Rekurzivně vytvoříme oba podstromy
- Časová složitost je $\mathcal{O}(n)$

BB[α]-strom: Operace Insert a Delete

Operace Insert (Delete je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací Build (složitost: amortizovaná analýza) ① ②

Amortizovaná časová složitost operací Insert a Delete: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace Build má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (Cvičení 2.2)
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$ ③
- Při jedné operaci Insert/Delete se prvek přidá/smaže v $\Theta(\log n)$ podstromech
- Amortizovaný čas vyvažování při jedné operaci Insert nebo Delete je $\mathcal{O}(\log n)$
- Jaký je celkový čas k operací? ④

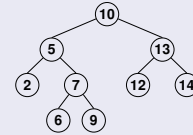
$$\Phi' - \Phi = \begin{cases} 2 & \text{pokud přidáváme a } p \leq 2n \\ -2 & \text{pokud mažeme a } p \leq 2n \\ -1 & \text{pokud přidáváme a } p \geq 2n \\ 1 & \text{pokud mažeme a } p \geq 2n \end{cases}$$

Binární vyhledávací strom

Vlastnosti

- Binární strom (každý vrchol obsahuje nejvýše dva syny)
- Klíč v každém vnitřním vrcholu je větší než všechny klíče v levém podstromu a menší než všechny klíče v pravém podstromu
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)

Příklad



Složitost

- Paměť: $\mathcal{O}(n)$
- Časová složitost operace Find je lineární ve výšce stromu
- Výška stromu může být až $n - 1$

- Do s_u započítáváme i vrchol u .

- V literatuře můžeme najít různé varianty této podmínky. Podstatné je, aby oba podstromy každého vrcholu měly „zhruba“ stejný počet vrcholů.

- Pro $\alpha = \frac{1}{2}$ lze BB[α]-strom sestavit, ale operace Insert a Delete by byly časově náročné. Pro $\alpha = 1$ by výška BB[α]-strom mohla být lineární.

- Při hledání listu pro nový vrchol stačí na cestě od kořene k listu kontrolovat, zda se přidáním vrcholu do podstromu syna neporuší vyvažovací podmínka. Pokud se v nějakém vrcholu podmínka poruší, tak se hledání ukončí a celý podstrom včetně nového prvku znovu vybuduje.

- Existují pravidla pro rotování BB[α]-stromů, ale ta se nám dnes nehodí.

- Operace Build podstromu vrcholu u trvá $\mathcal{O}(s_u)$ a mezi dvěma operacemi Build podstromu u je $\Omega(s_u)$ operací Insert nebo Delete do podstromu u . Všimněte si analogie a dynamickým polem.

- Intuitivně bychom mohli říct, že v nejhorším případě BB[α]-strom nejprve vyvážíme v čase $\mathcal{O}(n)$ a poté provádíme jednotlivé operace, a proto celkový čas je $\mathcal{O}(n + k \log n)$, ale není to pravda. Proč?

Amortizovaná časová složitost operací Insert a Delete: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$
- Necht postavení podstromu s s_u prvky vyžaduje nejvýše ds_u instrukcí
- Potenciál $\Phi(u)$ vrcholu u definujeme

$$\Phi(u) = \begin{cases} 0 & \text{pokud } s_{u.left} = s_{u.right} = \frac{s_u}{2} \\ ds_u & \text{pokud } \max\{s_{u.left}, s_{u.right}\} = \alpha s_u \end{cases}$$

a lineární interpolací v ostatních případech

- Potenciál BB[α]-stromu Φ je součet potenciálů vrcholů
- Při vložení/smazání prvku potenciál $\Phi(u)$ zvýší nejvýše o cd (Cvičení 2.3) ①
- Pro každý vrchol u : Skutečný počet instrukcí ve vrcholu $u + \Phi'(u) - \Phi(u) \leq \begin{cases} cd & \text{pokud } u \text{ leží na cestě z kořene do vloženého/smazaného prvku} \\ 0 & \text{jinak} \end{cases}$ ②
- Počet instrukcí pro jednu operaci $+ \Phi' - \Phi \leq cdh$, kde h je výška stromu
- $0 \leq \Phi \leq dhn = \mathcal{O}(n \log n)$ ③
- Celkový čas na k operací Insert nebo Delete je $\mathcal{O}((k+n) \log n)$

Cvičení:

1. Dokažte, že všechny listy v BB[α]-stromu jsou v hloubce $\Theta(\log n)$. Pokuste se určit co přesněji minimální a maximální hloubku listů.
2. Počet vrcholů v podstromu vrcholu u po vybalancování je s_u . Najděte minimální počet operací Insert a Delete, který způsobí porušení vyvažovací podmínky ve vrcholu u .
3. Najděte přesnou formuli pro potenciál vrcholu v amortizované analýze BB[α]-stromu v potenciální metodě. Dále přesně spočítejte, o kolik se maximálně zvýší potenciál vrcholu při vložení/smazání vrcholu v podstromu.
4. Pokud nejprve BB[α]-strom vybudujeme ze seříděného seznamu a poté provedeme k operací, tak celkový čas je $\mathcal{O}(n + k \log n)$, což je pro $k \ll n$ lepší odhad než $\mathcal{O}((k+n) \log n)$. Je tedy celkový čas na k operací Insert nebo Delete $\mathcal{O}(n + k \log n)$ začínáme-li z libovolného BB[α]-stromu s n listy (bez počátečního vyvážení)?
5. Vymyslete pravidla pro rotování v BB[α]-stromech při operacích Insert a Delete tak, aby složitost v nejhorším případě byla $\mathcal{O}(\log n)$. Pro jaké hodnoty parametru α dokážete splnit vyvažovací podmínku BB[α]-stromu?

Staticky optimální strom

Cíl

Pro danou posloupnost operací Find najít binární vyhledávací strom minimalizující celkovou dobu vyhledávání.

Formálně

Máme prvky x_1, \dots, x_n s váhami w_1, \dots, w_n . Cena stromu je $\sum_{i=1}^n w_i h_i$, kde h_i je hloubka prvku x_i . Staticky optimální strom je binární vyhledávací strom s minimální cenou.

Konstrukce (Cvičení 3.1)

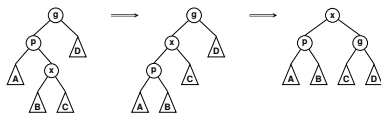
- $\mathcal{O}(n^2)$ – triviálně dynamickým programováním
- $\mathcal{O}(n^2)$ – vylepšené dynamické programování (Knuth [13])

Jak postupovat, když neznáme váhy předem?

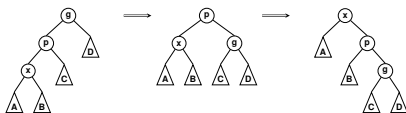
- Pomocí rotací bude udržovat často vyhledávané prvky blízko kořene
- Operací Splay „rotujeme“ zadaný prvek až do kořene
- Operace Find vždy volá Splay na hledaný prvek

Splay strom: Operace Splay prvku x

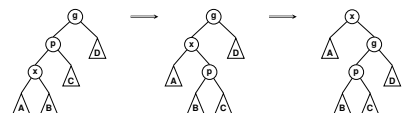
- Zig-zag step jsou pouze dvě (jednoduché) rotace prvku x s aktuálním otcem



- Zig-zig step jsou taky dvě rotace,



- ale dvě rotace prvku x s aktuálním otcem by vedly ke špatnému výsledku



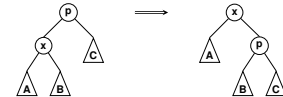
1. Potenciál $\Phi(u)$ může i klesnout. Konstanta c je závislá jen na parametru α .
2. Pokud u není na cestě z kořene do vloženého/smazaného prvku, tak skutečný čas ve vrcholu u je nula a $\Phi(u) = \Phi'(u)$. Pokud vrcholu u nepřestavíme podstrom, tak skutečný čas je nula potenciál vrcholu u vzroste nejvýše o cd . Na přestavení podstromu vrcholu u potřebujeme ds_u instrukcí a přesně o tolik klesne potenciál vrcholu u .
3. Součet potenciálů všech vrcholů v jedné libovolné vrstvě je nejvýše dn , protože každý vrchol patří do nejvýše jednoho podstromu vrcholu z dané vrstvy. Tudiž potenciál stromu Φ je vždy nejvýše dnh . Těž lze nahlédnout, že každý vrchol je započítán v nejvýše h potenciálech vrcholů.

Obsah

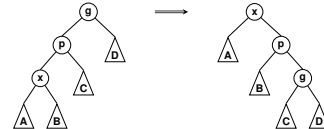
1. Amortizovaná analýza
2. Splay strom
3. (a,b)-strom a červeno-černý strom
4. Haldy
5. Cache-oblivious algorithms
6. Hešování
7. Geometrické datové struktury
8. Dynamizace
9. Bloom Filtry
10. Literatura

Splay strom (Sleator, Tarjan [25]): Operace Splay prvku x

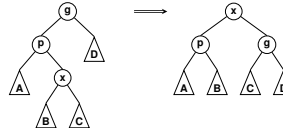
- Zig step: Otec p prvku x je kořen



- Zig-zig step: x a p jsou oba pravými nebo oba levými syny



- Zig-zag step: x je pravý syn a p je levý syn nebo opačně



Splay strom: Amortizovaná analýza

Lemma

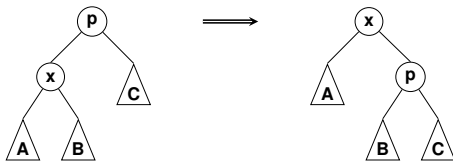
Jestliže $a + b \leq 1$, pak $\log_2(a) + \log_2(b) \leq -2$.

Důkaz

- Platí $4ab = (a+b)^2 - (a-b)^2$
- Z nerovnosti $(a-b)^2 \geq 0$ a $a+b \leq 1$ plyne $4ab \leq 1$
- Zlogaritmováním dostáváme $\log_2 4 + \log_2 a + \log_2 b \leq 0$

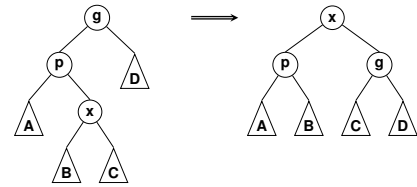
Značení

- Necht velikost $s(x)$ je počet vrcholů v podstromu x (včetně x)
- Potenciál vrcholu x je $\Phi(x) = \log_2(s(x))$
- Potenciál Φ stromu je součet potenciálů všech vrcholů
- s' a Φ' jsou velikosti a potenciály po jedné rotaci
- Analyzujeme počet jednoduchých rotací a potenciál je banka rotací



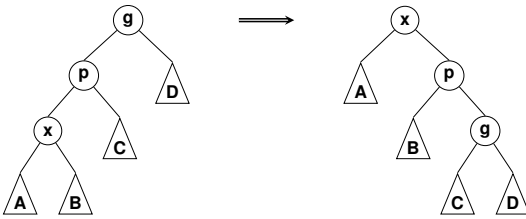
Análýza

- $\Phi'(x) = \Phi(p)$
- $\Phi'(p) < \Phi'(x)$
- $\Phi'(u) = \Phi(u)$ pro všechny ostatní vrcholy u
- $\Phi' - \Phi = \sum_u (\Phi'(u) - \Phi(u))$
 $= \Phi'(p) - \Phi(p) + \Phi'(x) - \Phi(x)$
 $\leq \Phi'(x) - \Phi(x)$



Análýza

1. $\Phi'(x) = \Phi(g)$
2. $\Phi(x) < \Phi(p)$
3. $\Phi'(p) + \Phi'(g) \leq 2\Phi'(x) - 2$
 - $s'(p) + s'(g) \leq s'(x)$
 - $\frac{s'(p)}{s'(x)} + \frac{s'(g)}{s'(x)} \leq 1$
 - Použijeme lemma: $a + b \leq 1 \Rightarrow \log_2(a) + \log_2(b) \leq -2$
 - $\log_2 \frac{s'(p)}{s'(x)} + \log_2 \frac{s'(g)}{s'(x)} \leq -2$
 - $\log_2 s'(p) + \log_2 s'(g) \leq 2 \log_2 s'(x) - 2$
4. $\Phi' - \Phi = \Phi'(g) - \Phi(g) + \Phi'(p) - \Phi(p) + \Phi'(x) - \Phi(x) \leq 2(\Phi'(x) - \Phi(x)) - 2$



Análýza

- $\Phi'(x) = \Phi(g)$
- $\Phi(x) < \Phi(p)$
- $\Phi'(p) < \Phi'(x)$
- $s(x) + s'(g) \leq s'(x)$
- $\Phi(x) + \Phi'(g) \leq 2\Phi'(x) - 2$
- $\Phi' - \Phi = \Phi'(g) - \Phi(g) + \Phi'(p) - \Phi(p) + \Phi'(x) - \Phi(x) \leq 3(\Phi'(x) - \Phi(x)) - 2$

• Zig rotaci použijeme nejvýše jednou a proto započítáme „+1“. Rozdíly $\Phi'(x) - \Phi(x)$ se teleskopicky odečtou a zůstane nám rozdíl potenciálů vrcholu x na konci a na začátku operace Splay. Na počátku je potenciál vrcholu x nezáporný a na konci je x kořenem, a proto jeho potenciál je $\log_2(n)$.

Amortizovaný čas

- Amortizovaný čas jedné zigzig nebo zigzag rotace: skutečný čas $+ \Phi' - \Phi \leq 2 + 3(\Phi'(x) - \Phi(x)) - 2 = 3(\Phi'(x) - \Phi(x))$
- Amortizovaný čas jedné zig rotace: skutečný čas $+ \Phi' - \Phi \leq 1 + \Phi'(x) - \Phi(x) \leq 1 + 3(\Phi'(x) - \Phi(x))$
- Nechť Φ_i je potenciál po i -tém kroku (zig, zig-zag nebo zig-zig step)
- Amortizovaný čas (počet jednoduchých rotací) jedné operace Splay:

$$\sum_{i\text{-tý krok}} (\text{skutečný čas} + \Phi_i - \Phi_{i-1}) \leq 1 + \sum_{i\text{-tý krok}} 3(\Phi_i(x) - \Phi_{i-1}(x))$$

$$\leq 1 + 3(\Phi_{\text{konec}}(x) - \Phi_0(x))$$

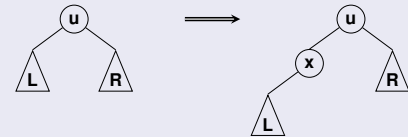
$$\leq 1 + 3 \log_2 n = \mathcal{O}(\log n)$$
- Amortizovaný čas jedné operace Splay je $\mathcal{O}(\log n)$

Skutečný čas k operací Splay

- Potenciál vždy splňuje $0 \leq \Phi \leq n \log_2 n$
- Rozdíl mezi konečným a počátečním potenciálem je nejvýše $n \log_2 n$
- Celkový čas k operací Splay je $\mathcal{O}((n+k) \log n)$

Vložení prvku x

1. Najdeme vrchol u s klíčem, který je nejbližší k x
2. Splay(u)
3. Vložit nový vrchol s prvkem x



Amortizovaná složitost

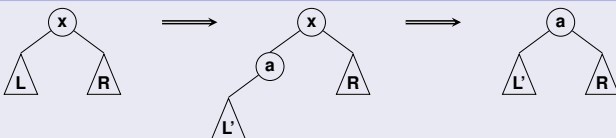
- Operace Find a Splay: $\mathcal{O}(\log n)$
- Vložením nového vrcholu potenciál Φ vzroste nejvýše o $\Phi'(x) + \Phi'(u) \leq 2 \log_2 n$
- Amortizovaná složitost operace Insert je $\mathcal{O}(\log n)$

Algoritmus

```

1 Splay(x)
2 L ← levý podstrom x
3 if L je prázdný then
4   Smazat vrchol x
5 else
6   Najít největší prvek a v L
7   Splay(a)
8   L' ← levý podstrom a
9   # a nemá pravého syna
10  Sloučit vrcholy x a a
    
```

Pokud L je neprázdný, tak



Stručné zadání

- Implementujte Splay strom s operacemi Splay, Find, Insert
- Implementujte „naivní Splay strom“, který v operaci Splay naivně používá jen jednoduché rotace místo dvojitých
- Mějte průměrnou hloubku hledaného prvku při operacích Find
- Analyzujte závislost průměrné hloubky hledaných prvků na počtu prvků v Splay stromu a velikosti hledané podmnožiny
- Analyzujte průměrnou hloubku hledaných prvků v „zákeřném“ testu
- Termín odevzdání: 13. 11. 2016
- Generátor dat a další podrobnosti: <https://ktiml.mff.cuni.cz/~fink/>

Cvičení:

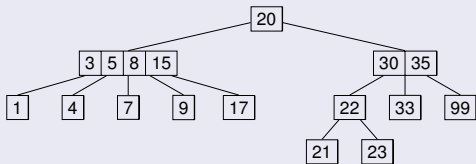
- Najděte algoritmus, který sestojí staticky optimální strom v čase $\mathcal{O}(n^2)$.
- Nechť $K[i, j]$ je prvek v kořeni staticky optimálního stromu obsahující pouze prvky x_i, \dots, x_j . Dokažte, že $K[i, j-1] \leq K[i, j] \leq K[i+1, j]$.
- Pomocí nerovnosti z předchozího příkladu najděte algoritmus, který sestojí staticky optimální strom v čase $\mathcal{O}(n^2)$.
- Najděte posloupnost operací Find, která z libovolného počátečního Splay stromu vytvoří cestu (strom výšky $n-1$).
- Uvažujme naivní splay strom, který využívá jen jednoduché rotace. Najděte posloupnost operací Find, která má v naivním splay stromu složitost $\Omega(nk)$, kde $k \geq n$ je vámi zvolený počet operací Find.
- Mějme ve splay stromu uloženu množinu prvků S a analyzujme vyhledávání podmnožiny $S' \subseteq S$. Nejprve každý prvek S' jednou vyhledáme (samozřejmě s využitím operace Splay). Dokažte, že následné (opakované) vyhledání prvků z množiny S' má amortizovanou složitost $\mathcal{O}(\log |S'|)$ a spočítejte celkovou složitost k vyhledání.

Vyhledávací strom

Vlastnosti

- Vnitřní vrcholy mají libovolný počet synů (typicky alespoň dva)
- Vnitřní vrchol s k syny má $k-1$ setříděných klíčů
- V každém vnitřním vrcholu je i -tý klíč větší než všechny klíče v i -tém podstromu a menší než všechny klíče v $(i+1)$ podstromu pro všechny klíče i
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)

Příklad

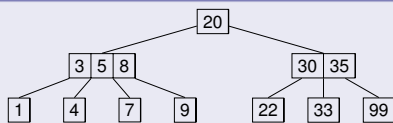


(a,b)-strom (Bayer, McCreight [1])

Vlastnosti

- a, b jsou celá čísla splňující $a \geq 2$ a $b \geq 2a-1$
- (a,b)-strom je vyhledávací strom
- Všechny vnitřní vrcholy kromě kořene mají alespoň a synů a nejvýše b synů
- Kořen má nejvýše b synů
- Všechny listy jsou ve stejné výšce
- Pro zjednodušení uvažujeme, že prvky jsou jen v listech

Příklad: (2,4)-strom



(a,b)-strom: Operace Insert

Algoritmus

- Najít otce v , kterému nový prvek patří
- Přidat nový list do v
- while** $\text{deg}(v) > b$ **do**
 - # Najdeme otce u vrcholu v
 - if** v je kořen **then**
 - Vytvořit nový kořen u s jediným synem v
 - else**
 - $u \leftarrow$ otec v
 - # Rozdělíme vrchol v na v and v'
 - Vytvořit nového syna v' utci u a umístit jej vpravo vedle v
 - Přesunout nejpravějších $\lfloor (b+1)/2 \rfloor$ synů vrcholu v do v'
 - Přesunout nejpravějších $\lfloor (b+1)/2 \rfloor - 1$ klíčů vrcholu v do v'
 - Přesunout poslední klíč vrcholu v do u
 - $v \leftarrow u$

Časová složitost

Lineární ve výšce stromu (předpokládáme, že a, b jsou pevné parametry)

Obsah

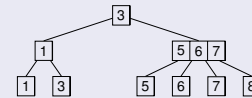
- Amortizovaná analýza
- Splay strom
- (a,b)-strom a červeno-černý strom
 - (a,b)-strom
 - Červeno-černý strom
- Haldy
- Cache-oblivious algorithms
- Hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

Obsah

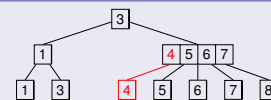
- Amortizovaná analýza
- Splay strom
- (a,b)-strom a červeno-černý strom
 - (a,b)-strom
 - Červeno-černý strom
- Haldy
- Cache-oblivious algorithms
- Hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

(a,b)-strom: Operace Insert

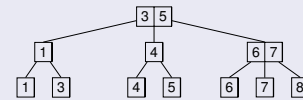
Vložte prvek s klíčem 4 do následujícího (2,4)-stromu



Nejprve najdeme správného otce, jemuž přidáme nový list

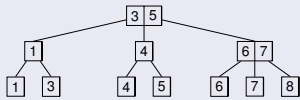


Opakovaně rozdělujeme vrchol na dva

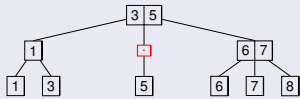


Musíme ještě dokázat, že po provedení všech operací doopravdy dostaneme (a,b)-strom. Ověříme, že rozdělené vrcholy mají alespoň a synů (ostatní požadavky jsou triviální). Rozdělovaný vrchol má na počátku právě $b+1$ synů a počet synů po rozdělení je $\lfloor \frac{b+1}{2} \rfloor$ a $\lceil \frac{b+1}{2} \rceil$. Protože $b \geq 2a-1$, počet synů po rozdělení je alespoň $\lfloor \frac{b+1}{2} \rfloor \geq \lfloor \frac{2a-1+1}{2} \rfloor = \lfloor a \rfloor = a$.

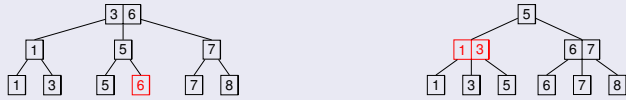
Smažte prvek s klíčem 4 z následujícího (2,4)-stromu



Nalezneme a smažeme list



Přesuneme jednoho syna od bratra nebo spojíme vrchol s bratrem

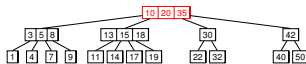


- 1 Při přesunu je nutné upravit klíče ve vrcholech u, v a jejich otci.
- 2 Vrchol u měl a , vrchol v měl $a - 1$ synů. Po jejich sjednocení máme vrchol s $2a - 1 \leq b$ syny.

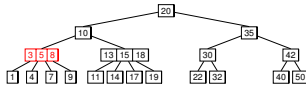
- 1 Při jedné vyvažovací operaci (štěpení vrcholu) je počet modifikovaných vrcholů omezený konstantou (štěpení vrcholu, otec a synové). Asymptoticky jsou počty modifikovaných vrcholů a vyvažovacích operací stejné.

(a,b)-strom: Paralelní přístup: Příklad

- 1 Vložte prvek s klíčem 6 do následujícího (2,4)-stromu



- 2 Nejprve rozdělíme kořen



- 3 Pak pokračujeme do levého syna, který taky rozdělíme



- 4 Vrchol s klíčem 8 není třeba rozdělovat a nový klíč můžeme vložit



Algoritmus

```

1 Najít list l obsahující prvek s daným klíčem
2 v ← otec l
3 Smažat l
4 while deg(v) < a & v není kořen do
5   u ← sousední bratr v
6   if deg(u) > a then
7     Přesunout správného syna u pod v ⊕
8   else
9     Přesunout všechny syny u pod v ⊙
10    Smažat u
11    if v nemá žádného bratra then
12      Smažat kořen (otec v) a nastavit v jako kořen
13    else
14      v ← otec v

```

(a,b)-strom: Analýza

Výška

- (a,b)-strom výšky d má alespoň a^{d-1} a nejvýše b^d listů.
- Výška (a,b)-stromu splňuje $\log_b n \leq d \leq 1 + \log_a n$.

Složitost

Časová složitost operací Find, Insert and Delete je $O(\log n)$.

Počet modifikovaných vrcholů při vytvoření stromu operací Insert

- Vytváříme (a,b)-strom pomocí operace Insert
- Zajímá nás celkový počet vyvažovacích operací ⊕
- Při každém štěpení vrcholu vytvoříme nový vnitřní vrchol
- Po vytvoření má strom nejvýše n vnitřních vrcholů
- Celkový počet štěpení je nejvýše n a počet modifikací vrcholů je $O(n)$
- Amortizovaný počet modifikovaných vrcholů na jednu operaci Insert je $O(1)$

(a,b)-strom: Paralelní přístup

Cíl

Umožnit efektivní paralelizaci operací Find, Insert a Delete (předpoklad: $b \geq 2a$).

Operace Insert

Preventivně rozdělit každý vrchol na cestě od kořene k hledanému listu s b syny na dva vrcholu.

Operace Delete

Preventivně sloučit každý vrchol na cestě od kořene k hledanému listu s a syny s bratrem nebo přesunout synovce.

A-sort (Guibas, McCreight, Plass, Roberts [9])

Cíl

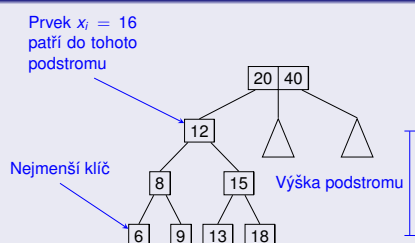
Setřídít „skoro“ setříděné pole

Modifikace (a,b)-stromu

Máme uložený ukazatel na vrchol s nejmenším klíčem

Příklad: Vložte klíč s hodnotou $x_i = 16$

- Začneme od vrcholu s nejmenším klíčem a postupujeme ke kořeni, dokud x_i nepatří podstromu aktuálního vrcholu
- V rámci tohoto podstromu spustíme operaci Insert
- Výška podstromu je $\Theta(\log f_i)$, kde f_i je počet klíčů menších než x_i



Input: Posloupnost x_1, x_2, \dots, x_n

```

1  $T \leftarrow$  prázdný (a,b)-strom
2 for  $i \leftarrow n$  to 1 # Prvky procházíme od konce
3 do
    # Najdeme podstrom, do kterého vložíme  $x_i$ 
4      $v \leftarrow$  list s nejmenším klíčem
5     while  $v$  není kořen a  $x_i$  je větší než nejmenší klíč v otcí vrcholu  $v$  do
6          $v \leftarrow$  otec  $v$ 
7     Vložíme  $x_i$  do podstromu vrcholu  $v$ 

```

Output: Projdeme celý strom a vypíšeme všechny klíče (in-order traversal)

(a,b)-strom: Závěr

Počet modifikovaných vrcholů při operaci Insert a Delete (Cvičení 4.8) [11]

- Předpoklad: $b \geq 2a$
- Počet modifikovaných vrcholů při l operacích Insert a k Delete je $O(k + l + \log n)$
- Amortizovaný počet modifikovaných vrcholů při operacích Insert a Delete je $O(1)$

Podobné datové struktury

- B-tree, B+ tree, B* tree
- 2-4-tree, 2-3-4-tree, etc.

Aplikace

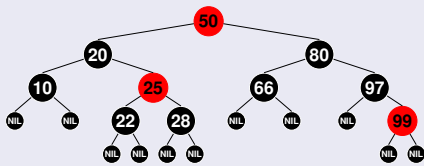
- File systems např. Ext4, NTFS, HFS+, FAT
- Databáze

Červeno-černé stromy (Guibas, Sedgewick [10])

Definice

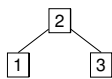
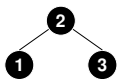
- Binární vyhledávací strom s prvky uloženými ve všech vrcholech
- Každý vrchol je černý nebo červený
- Všechny cesty od kořene do listů obsahují stejný počet černých vrcholů
- Otec červeného vrcholu musí být černý
- Listy jsou černé

Příklad

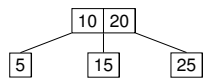
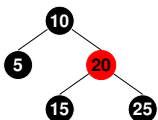


Červeno-černé stromy: Ekvivalence s (2,4)-stromy

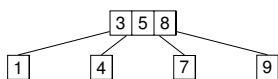
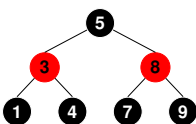
- Vrchol bez červených synů



- Vrchol s jedním červeným synem



- Vrchol s dvěma červenými syny



Nerovnost mezi aritmetickým a geometrickým průměrem

Jestliže a_1, \dots, a_n nezáporná reálná čísla, pak platí

$$\frac{\sum_{i=1}^n a_i}{n} \geq \sqrt[n]{\prod_{i=1}^n a_i}$$

Časová složitost

- Necht $f_i = |\{j > i; x_j < x_i\}|$ je počet klíčů menších než x_i , které již jsou ve stromu při vkládání x_i
- Necht $F = |\{(i, j): i > j, x_i < x_j\}| = \sum_{i=1}^n f_i$ je počet inverzí
- Složitost nalezení podstromu, do kterého x_i patří: $O(\log f_i)$
- Nalezení těchto podstromů pro všechny podstromy $\sum_i \log f_i = \log \prod_i f_i = n \log \sqrt[n]{\prod_i f_i} \leq n \log \frac{\sum_i f_i}{n} = n \log \frac{F}{n}$
- Rozdělování vrcholů v průběhu všech operací Insert: $O(n)$
- Celková složitost: $O(n + n \log(F/n))$
- Složitost v nejhorším případě: $O(n \log n)$ protože $F \leq \binom{n}{2}$
- Jestliže $F \leq n \log n$, pak složitost je $O(n \log \log n)$

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
 - (a,b)-strom
 - Červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- Nepovinná podmínka, která jen zjednodušuje operace. V příkladu uvažujeme, že listy jsou reprezentovány NIL/NULL ukazateli, a tedy imaginární vrcholy bez prvků. Někdy se též vyžaduje, aby kořen byl černý, ale tato podmínka není nutná, protože kořen můžeme vždy přebarvit na černý bez porušení ostatních podmínek.

- Převod mezi červeno-černými stromy a (2,4)-stromy není jednoznačný, protože vrchol (2,4)-stromu se třemi syny a prvky $x < y$ lze převést na černý vrchol červeno-černého stromu s prvkem x a pravým červeným synem y nebo s prvkem y a levým červeným synem x .

Vytvoření nového vrcholu

- Najít list pro nový prvek n
- Přidat nový vrchol



- Pokud otec p je červený, pak je nutné strom vybalancovat

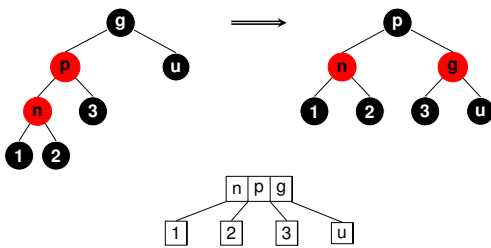
Balancování

- Vrchol n a jeho otec p jsou červené vrcholy a toto je jediná porušená podmínka
- Děda g vrcholu n je černý

Musíme uvažovat tyto případy:

- Strýc u je černý nebo červený
- Vrchol n je pravým nebo levým synem p (podobně pro vrchol p)

Červeno-černé stromy: Operace Insert, strýc je černý



Pořadí prvků v (2,4)-stromu a výsledný červeno-černý strom závisí na tom, zda vrchol n je pravým nebo levým synem p a zda vrchol p je pravým nebo levým synem g .

Červeno-černé stromy: Vlastnosti

Důsledky ekvivalence s (2,4)-stromy

- Výška červeno-černého stromu je $\Theta(\log n)$
- Časová složitost operací Find, Insert a Delete je $\mathcal{O}(\log n)$
- Amortizovaný počet modifikovaných vrcholů při operaci Insert a Delete je $\mathcal{O}(1)$
- Paralelní přístup (top-down balancování)

Aplikace

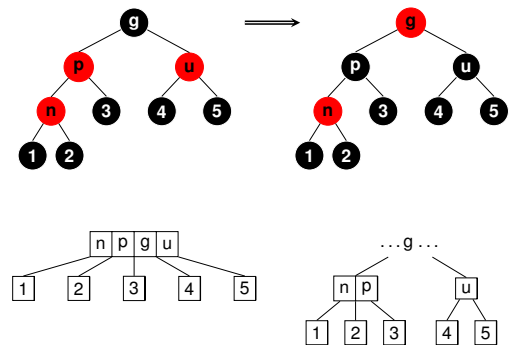
- Asociativní pole např. `std::map` and `std::set` v C++, `TreeMap` v Java
- The Completely Fair Scheduler in the Linux kernel
- Computational Geometry Data structures

Cvičení:

1. Jak v (a,b)-stromu (obecném vyhledávacím stromu) najít prvek s nejmenším/největším klíčem? Jak najít prvek s nejbližším klíčem (tj. nejmenší větší klíč nebo největší menší klíč k danému klíči)?
2. Upravte (a,b)-strom (obecný vyhledávací strom) tak, aby bylo možné efektivně najít k -tý nejmenší/největší prvek pro dané přirozené k . Dokážete v takto modifikovaném stromu určit pořadí daného prvku?
3. V klasické implementaci (a,b)-stromu má každý vrchol alokováno pole velikosti b pro klíče a ukazatele na syny. Toto pole může být až z poloviny nevyužito, což může být významným plýtváním paměti. Pokuste se upravit operace Insert a Delete tak, aby bylo zaručeno efektivnější využití tohoto pole.
4. Přesně popište implementaci (a,b)-stromu a zamykání jednotlivých vrcholů v operacích Insert a Delete tak, aby správně fungoval paralelní přístup.
5. Jak vytvořit (a,b)-strom ze setříděného pole (co nejrychleji)?
6. Existuje asymptoticky rychlejší postup vytvoření (a,b)-stromu z nesetříděného pole než vložení všech prvků?
7. Najděte asymptoticky nejrychlejší způsob sjednocení dvou (a,b)-stromů, jestliže jeden (a,b)-strom nemá všechny prvky menší než je nejmenší prvek druhého.
8. Dokažte, že v (a,b)-stromu pro $b \geq 2a$ je počet modifikovaných vrcholů $\mathcal{O}(k + l + \log n)$ při l operacích Insert a k operacích Delete. Pro zjednodušení můžete uvažovat (2,4)-strom. Dále ukažte, že podmínka $b \geq 2a$ je nutná.

- S využitím symetrií lze počet případů snížit.

Červeno-černé stromy: Operace Insert, strýc je červený



Po rozdělení vrchol (2,4)-stromu se prvek g přesouvá do otce, a proto je vrchol g červený.

- Počet černých vrcholů na cestě ke kořeni je stejný jako výška odpovídajícího (2,4)-stromu, a tedy výška červeno-černého stromu je nejvýše dvojnásobek výšky (2,4)-stromu.

9. Vymyslete operace Insert a Delete v (2,5)-stromu pro paralelní aplikace tak, aby amortizovaný počet modifikovaných vrcholů byl $\mathcal{O}(1)$.
10. Vymyslete pravidla pro operaci Delete v červeno-černých stromech pomocí ekvivalence s (2,4)-stromy.
11. Upravte operace Insert a Delete v červeno-černém stromu tak, že tyto operace projdou strom jen jednou od kořene k listu (bez zpětného průchodu od listu ke kořeni). Lze tuto modifikaci použít k paralelizaci?

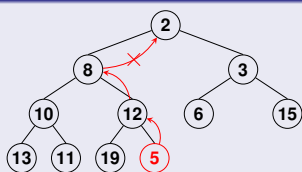
- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
 - d -regulární halda
 - Binomiální halda
 - Fibonacciho halda
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
 - d -regulární halda
 - Binomiální halda
 - Fibonacciho halda
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- 4 Necht h je nejnižší plná hladina. Jelikož h -tá hladina obsahuje d^h vrcholů, tak platí $n \geq d^h$, z čehož plyne $h \leq \log_d n$. Tudiš výška d -regulární haldy s n prvky je nejvýše $1 + \log_d n$. Najděte formuli udávající přesnou výšku d -regulární haldy.

d -regulární halda: Operace Insert

Příklad: Vložme prvek s klíčem 5



Základní operace

- Insert
- FindMin
- DeleteMin
- DecreaseKey: Snižit hodnotu klíče v daném vrcholu

Haldivý invariant

Klíč v každém vrcholu větší nebo roven klíči v otci.

Aplikace

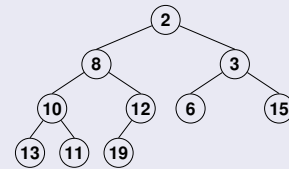
- Prioritní fronta
- Heap-sort
- Dijkstrův algoritmus (nejkratší cesta)
- Jarníkův (Primův) algoritmus (minimální kostra)

d -regulární halda (Johnson [12])

Vlastnosti

- Každý vrchol má nejvýše d synů
- Všechny vrstvy kromě poslední jsou úplně zaplněné
- Poslední hladina je zaplněná zleva

Příklad 2-regulární (binární) haldy

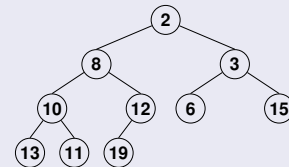


Cvičení 5.1

Jaká je přesná výška d -regulární haldy s n prvky? ☹

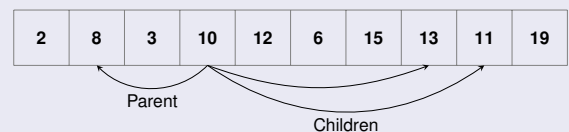
d -regulární halda: Representace

Binární halda uložená ve stromu



Binární halda uložená v poli

A vrchol na pozici i má otce na pozici $\lfloor (i-1)/2 \rfloor$ a syny na pozici $2i+1$ a $2i+2$:



Cvičení 5.2: Určete pozice otce a synů pro obecnou d -regulární haldu

d -ary heap: Operace Insert a DecreaseKey

Insert: Algoritmus

- Input:** Nový prvek s klíčem x
- 1 $v \leftarrow$ první volný blok v poli
 - 2 Nový prvek uložíme na pozici v
 - 3 **while** v není kořen a otec p vrcholu v má klíč větší než x **do**
 - 4 Prohodíme prvky na pozicích v a p
 - 5 $v \leftarrow p$

Operace DecreaseKey

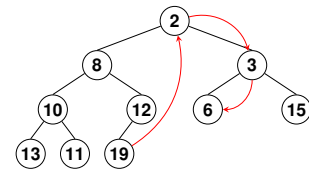
Snižíme hodnotu klíče a pokračujeme podobně jako při operaci Insert

Časová složitost ☹

$\mathcal{O}(\log_d n)$

- Pro přesnější analýzu nás zajímá závislost složitosti na hodnotě d . Později se nám bude hodit nastavovat d podle hodnot na vstupu.

Operace DeleteMin



Algoritmus

```

1 Přesuneme poslední prvek do kořene v
2 while Některý se synů vrcholu v má klíč menší než v do
3   u ← syn vrcholu v s menším klíčem
4   Prohodíme prvky ve vrcholech u a v
5   v ← u

```

Složitost

$\mathcal{O}(d \log_d n)$

d -regulární halda: Operace Build

Cíl

Vytvořit haldu z daného pole prvků

Algoritmus

```

1 for r ← poslední pozice to první pozice v poli do
2   # Zpracujeme vrchol r podobně jako při operaci Delete
3   v ← r
4   while Některý se synů vrcholu v má klíč menší než v do
5     u ← syn vrcholu v s menším klíčem
6     Prohodíme prvky ve vrcholech u a v
7     v ← u

```

Korektnost

Podstromy všech zpracovaných vrcholů tvoří haldu

- Podstromem vrcholu u rozumíme vrchol u a všechny vrcholy pod u .
- Člen „+1“ započítáváme, protože jeden podstrom může být neúplný.

Binomiální strom

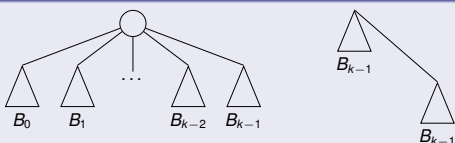
Definice

- Binomiální strom B_0 řádu 0 je jeden vrchol
- Binomiální strom B_k řádu $k \geq 1$ má kořen, jehož synové jsou kořeny binomiálních stromů řádu $0, 1, \dots, k-1$.

Alternativně

Binomiální strom řádu k is je vytvořen z dvou binomiálních stromů řádu $k-1$ tak, že se jeden strom připojí jako nejpravější syn kořene druhého stromu.

Rekurzivní definice binomiálního stromu

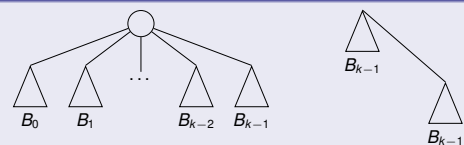


Obsah

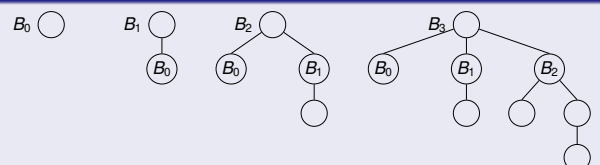
- Amortizovaná analýza
- Splay strom
- (a,b)-strom a červeno-černý strom
- Haldu
 - d -regulární halda
 - Binomiální halda
 - Fibonacciho halda
- Cache-oblivious algorithms
- Hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

Binomiální strom: Příklad

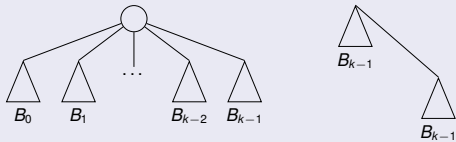
Rekurzivní definice binomiálního stromu



Binomiální stromy řádu 0, 1, 2 a 3



Rekurzivní definice binomiálního stromu



Vlastnosti

Binomiální strom B_k má

- 2^k vrcholů,
- výšku k ,
- k synů v kořeni,
- maximální stupeň k ,
- $\binom{k}{d}$ vrcholů v hloubce d .

Podstrom vrcholu s k syny je izomorfní B_k .

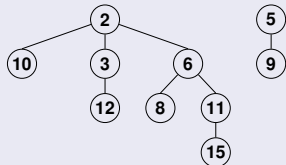
Binomiální halda (Vuillemin [26])

Definice

Binomiální halda je množina binomiálních stromů taková, že:

- Každý prvek je uložen právě v jednom vrcholu jednoho binomiálního stromu
- Každý binomiální strom je halda (otec má menší klíč než syn)
- Žádné dva binomiální stromy nemají stejný řád

Příklad



Binomiální halda: Representace

Struktura pro vrchol binomiálního stromu obsahuje

- prvek (klíč a hodnota),
- ukazatel na otce,
- ukazatel na nejlevějšího a nejpravějšího syna,
- ukazatel na levého a pravého bratra a \ominus
- řád postromu.

Binomiální halda

- Binomiální stromy jsou uloženy ve spojovém seznamu pomocí ukazatelů na bratry. \ominus
- Odstraněním kořene binomiálního stromu vznikne binomiální halda v čase $\mathcal{O}(1)$.
- Binomiální halda si udržuje ukazatel na strom s minimálním prvkem.

Operace FindMin

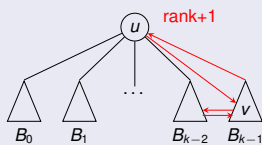
Triviálně v čase $\mathcal{O}(1)$

Operace DecreaseKey

Stejně jako v regulární haldě.

Binomiální halda: Spojení dvou binomiálních hald

Spojení dvou binomiálních stromů stejného řádu v čase $\mathcal{O}(1)$



Join

Spojení dvou binomiálních hald je jako sčítání binární čísel: sjednocujeme binomiální stromy od nejmenších. Složitost je $\mathcal{O}(\log n)$, kde n je celkový počet prvků

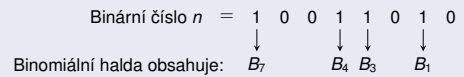
Příklad

Binomiální strom	B_6	B_5	B_4	B_3	B_2	B_1	B_0
První halda	0	1	1	0	1	1	0
Druhá halda	0	1	1	0	1	0	0
Spojení	1	1	0	1	0	1	0

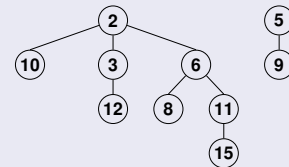
Pozorování

Pro každé n existuje (právě jedna) množina binomiálních stromů různých řádů taková, že celkový počet vrcholů je n .

Vztah mezi binárními čísly a binomiálními stromy



Příklad pro 1010_2 prvků

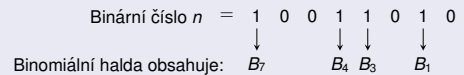


Binomiální halda: Výška

Pozorování

Binomiální halda obsahuje nejvýše $\log_2(n+1)$ stromů a každý má výšku nejvýše $\log_2 n$.

Vztah mezi binárními čísly a binomiálními stromy



1. Ukazatele tvoří obousměrný spojový seznam synů a tento seznam udržujeme seřazený podle řádu.
2. Binomiální stromy jsou ve spojovém seznamu taky seřazené podle řádu.

Binomiální halda: Operace Insert a DeleteMin

Operace Insert

- Vytvoříme binomiální strom řádu 0 s novým prvkem
- Procházíme seznam stromů od nejmenších: \ominus
 - Pokud strom B_0 je v haldě, tak jej sjednotíme s novým stromem B_0 , čímž vytvoříme B_1
 - Pokud strom B_1 je v haldě, tak jej sjednotíme s novým stromem B_1 , čímž vytvoříme B_2
 - Takto pokračujeme až k tomu stromu s nejmenším řádem, který není uložený v haldě, a nový strom vložíme do haldy \ominus
- Složitost v nejhorším případě je $\mathcal{O}(\log n)$
- Amortizovaná složitost je $\mathcal{O}(1)$ podobně jako inkrementace binárního čítače
 - Zde je důležité, že neprocházíme všechny stromy v haldě

Operace DeleteMin

Odstraníme kořen s minimálním prvkem, čímž vznikne nová binomiální halda, kterou sjednotíme se zbytkem původní haldy v čase $\mathcal{O}(\log n)$.

- 1 Stromy v haldě udržujeme seříděné podle řádu.
- 2 Nový strom je nejmenší, takže jej vložíme na začátek seznamu.

Líná binomiální halda: Rekonstrukce poctivé binomiální haldy

Idea

- Dokud máme v haldě binomiální haldy stejného řádu, jak je spojujeme
- Použijeme pole indexované řádem stromu k vyhledávání stromů stejného řádu

Algoritmus

```

1 Inicializujeme pole velikosti  $\lceil \log_2(n+1) \rceil$  ukazatelem NIL
2 for pro každý strom  $h$  v líné binomiální haldě do
3    $o \leftarrow$  řád stromu  $h$ 
4   while  $pole[o] \neq NIL$  do
5      $h \leftarrow$  spojení stromů  $h$  a  $pole[o]$ 
6      $pole[o] \leftarrow NIL$ 
7      $o \leftarrow o + 1$ 
8    $pole[o] \leftarrow h$ 
9 Pole stromů převedeme na spojový seznam, čímž vytvoříme poctivou binomiální haldu

```

Cvičení 5.6

Amortizovaná složitost operace DecreaseKey je $\mathcal{O}(\log n)$.

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
 - d -regulární halda
 - Binomiální halda
 - Fibonacciho halda
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- 1 Doposud probírané datové struktury mají jasně definovanou strukturu a operace jsou navrženy tak, aby tuto strukturu zachovávaly. Fibonacciho halda je definovaná povolenými operacemi a vlastnosti se odvozují z operací.
- 2 Podobně jako binomiální halda, ale stromy nemusí být izomorfní s binomiálními stromy.
- 3 Podobně jako binomiální halda, ale vztahy pro počet vrcholů nebo výšku neplatí. Tvzení, že podstromy vrcholu k mají řád $0, 1, \dots, k-1$ budeme muset upravit.
- 4 Podobně jako v binomiální haldě kořen jednoho stromu připojíme jako syna kořene druhého stromu.
- 5 Nové prvky vkládáme podobně jako v líné binomiální haldě.
- 6 Nejmenší prvek mažeme podobně jako v líné binomiální haldě, a to včetně následné rekonstrukce, kde spojujeme stromy stejného řádu.

Líná binomiální halda

Změna v poctivé binomiální haldě

Líná binomiální halda může obsahovat libovolný počet binomiálních stromů stejného řádu.

Operace Insert a Join

- Pouze spojíme seznamy stromů
- Složitost $\mathcal{O}(1)$ v nehorším případě

Operace DeleteMin

- Smažeme kořen s minimálním prvkem
- Spojíme seznam synů smazaného kořene s ostatními stromy v haldě
- Zrekonstruujeme poctivou binomiální haldou
- Najdeme nový minimální prvek

Haldy: Přehled

Složitosti různých hald

	Binární	Binomiální		Líná binomiální	
	nejhorší	nejhorší	amortizovaně	nejhorší	amortizovaně
Insert	$\log n$	$\log n$	1	1	1
DecreaseKey	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$
DeleteMin	$\log n$	$\log n$	$\log n$	n	$\log n$

Cvičení 5.7

Je možné vytvořit haldou, která má amortizovanou složitost operací Insert a DeleteMin lepší než $\mathcal{O}(\log n)$?

Další cíl

Zrychlit operaci DecreaseKey

Postup

V líné binomiální haldě musí každý strom být izomorfní binomiálnímu stromu. Ve Fibonacciho haldě tento požadavek neplatí.

Fibonacciho halda (Fredman, Tarjan [7])

Základní vlastnosti a povolené operace ①

- Fibonacciho halda je seznam haldových stromů ②
- Řád stromu je počet synů kořene ③
- Smíme spojit dva stromy stejného řádu ④
- Každému vrcholu kromě kořene smíme odpojit nejvýše jednoho syna
 - Do reprezentace vrcholu přidáme bitovou informaci, zda vrchol již o syna přišel
- Kořen může přijít o libovolný počet synů
 - Stane-li se vrchol kořenem, tak jej odznačíme
 - Je-li kořen připojen do jiného stromu, tak smí ztratit nejvýše jednoho syna, dokud se nestane znovu kořenem
- Smíme vytvořit nový strom s jediným prvkem ⑤
- Smíme smazat kořen stromu ⑥

Operace stejné jako v líné binomiální haldě

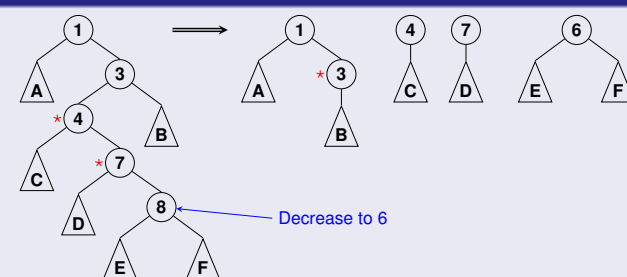
Insert, FindMin, DeleteMin

Fibonacciho halda: Operace DecreaseKey

Idea

- Danému vrcholu snížíme hodnotu klíče a odpojíme jej od otce
- Pokud otec je označený, tak jej taky odpojíme
- Pokud je děda taky označený, tak jej taky odpojíme
- Takto pokračuje, dokud nenarazíme na neoznačený vrchol nebo kořen

Příklad



Algoritmus

Input: Vrchol u a nový klíč k

```

1 Snížíme klíč vrcholu  $u$ 
2 if  $u$  je kořen nebo otec  $p$  vrcholu  $u$  má klíč nejvýše  $k$  then
3   return # Haldový invariant je zachován
4  $p \leftarrow$  otec vrcholu  $u$ 
5 Označit vrchol  $u$ 
6 Odpojit vrchol  $u$  od otce  $p$  a připojit  $u$  k seznamu stromů
7 while  $p$  není kořen a  $p$  je označený do
8    $u \leftarrow p$ 
9    $p \leftarrow$  otec  $u$ 
10  Označit vrchol  $u$ 
11  Odpojit vrchol  $u$  od otce  $p$  a připojit  $u$  k seznamu stromů
12 if  $p$  není kořen then
13   Označit vrchol  $p$ 

```

- 1 Předpokládáme, že synové jsou očíslováni podle „věku“, tj. později vložený syn má větší index
- 2 Halda nemá žádný vrchol, a proto neexistuje vrchol porušující invariant.
- 3 Nový vrchol nemá žádného syna, a tak nemá syna porušující invariant.
- 4 Spojíme stromy u a p řádu $k-1$. Po spojení je k -tý syn u vrcholu p neoznačený a má $k-1$ synů. Pořadí ostatních synů vrcholu p je zachováno.
- 5 Invariant je zachován, protože se minimální počet požadovaných synů těchto vrcholů sníží o jedna a skutečný počet je zachován.
- 6 Neoznačený j -tý vrchol u musel mít alespoň $j-1$ synů. Po odstranění vrcholu u se počet synů vrcholu u sníží o jedna, a proto má alespoň $j-2$ synů, což je minimální požadovaný počet synů j -tého označeného syna vrcholu p .

Fibonacciho halda: Struktura

Velikost podstromu

Nechť s_k je minimální počet vrcholů v podstromu vrcholu s s k syny. Pak platí $s_k \geq s_{k-2} + s_{k-3} + s_{k-4} + \dots + s_2 + s_1 + s_0 + s_0 + 1$.

Fibonacciho čísla (Cvičení 5.9)

- $F_0 = 0$ a $F_1 = 1$ a $F_k = F_{k-1} + F_{k-2}$
- $\sum_{i=1}^k F_i = F_{k+2} - 1$
- $F_k = \frac{(1+\sqrt{5})^k - (1-\sqrt{5})^k}{2^k \sqrt{5}}$
- $F_k \geq \left(\frac{1+\sqrt{5}}{2}\right)^k$
- $s_k \geq F_{k+2}$
 - $s_k \geq 1 + s_0 + \sum_{i=0}^{k-2} s_i \geq 1 + F_1 + \sum_{i=0}^{k-2} F_{i+2} \geq 1 + \sum_{i=1}^k F_i = 1 + F_{k+2} - 1$

Důsledek

Strom řádu k má alespoň $s_k \geq F_{k+2} \geq \left(\frac{1+\sqrt{5}}{2}\right)^{k+2}$ vrcholů. Proto,

- kořen stromu s m vrcholy má $\mathcal{O}(\log m)$ synů a
- Fibonacciho halda má $\mathcal{O}(\log n)$ stromů po operaci DeleteMin. ①

Fibonacciho halda: Složitost

Složitost v nejhorším případě

- Operace Insert: $\mathcal{O}(1)$
- Operace Decrease-key: $\mathcal{O}(n)$ (Cvičení 5.10)
- Operace Delete-min: $\mathcal{O}(n)$

Amortizovaná složitost: Potenciál

Uvažujme potenciál $\Phi = t + 2m$, kde

- t je počet stromů v haldě
- m je celkový počet označených vrcholů

Amortizovaná složitost: Operace Insert

- Skutečný čas: $\mathcal{O}(1)$
- Změna potenciálu $\Phi' - \Phi = 1$
- Amortizovaná složitost $\mathcal{O}(1)$

Invariant

Pro každý vrchol p a jeho i -tého syna s platí, že s má alespoň

- $i-2$ synů, pokud s je označený, a
- $i-1$ synů, pokud s není označený. ①

Důkaz

Všechny povolené operace zachovávají platnost invariantu

- Iniz: Prázdná halda invariant splňuje ②
- Insert: Vytvoření nového stromu s jedním vrcholem ③
- DeleteMin: Pro nesmazané vrcholy se počty synů ani jejich pořadí nezmění
- Join: Připojení stromu u řádu $k-1$ jako k -tého syna vrcholu p ④
- Odstranění i -tého syna x z vrcholu u řádu k , který je kořenem
 - Pořadí $(i+1)$ -tého až k -tého syna vrcholu u se sníží o jedna ⑤
- Odstranění i -tého syna x z neoznačeného vrcholu u řádu k , který j -tým synem p
 - Pořadí $(i+1)$ -tého až k -tého syna vrcholu u se sníží o jedna
 - Před odstraněním x platilo $k \geq j-1$ a po odstranění x je vrchol u označený a počet synů u splňuje $k-1 \geq j-2$ ⑥

Fibonacciho halda: Struktura

Invariant

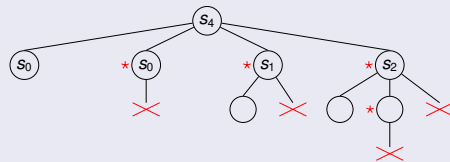
Pro každý vrchol p a jeho i -tého syna s platí, že s má alespoň

- $i-2$ synů, pokud s je označený, a
- $i-1$ synů, pokud s není označený.

Velikost podstromu

Nechť s_k je minimální počet vrcholů v podstromu vrcholu s s k syny. Pak platí $s_k \geq s_{k-2} + s_{k-3} + s_{k-4} + \dots + s_2 + s_1 + s_0 + s_0 + 1$.

Příklad



- V obecně můžeme mít Fibonacciho halda až n stromů, ale po konsolidaci (součást operace DeleteMin) mají každé dva stromy různý řád a maximální řád stromu je $\mathcal{O}(\log n)$.

Fibonacciho halda: Amortizovaná složitost operace DecreaseKey

Potenciál

$\Phi = t + 2m$, kde t je počet stromů v haldě a m je celkový počet označených vrcholů

Jedna iterace while-cyklu (odznačení vrcholu a odpojení od otce)

- Skutečný čas: $\mathcal{O}(1)$
- Změna potenciálu $\Phi' - \Phi = 1 - 2$
- Amortizovaná složitost: ①

Ostatní instrukce

- Skutečný čas: $\mathcal{O}(1)$
- Změna potenciálu $\Phi' - \Phi = 3$
- V nejhorším případě vytvoříme nový strom a jeden vrchol označíme
- Amortizovaná složitost: $\mathcal{O}(1)$

Amortizovaná složitost operace DecreaseKey

$\mathcal{O}(1)$

Smazání kořene a připojení synů k seznamu stromů

- Skutečný čas: $\mathcal{O}(\log n)$
- Změna potenciálu $\Phi' - \Phi = \mathcal{O}(\log n)$
- Amortizovaná složitost: $\mathcal{O}(\log n)$

Jedna iterace while-cyklu při rekonstrukci (spojení dvou stromů)

- Skutečný čas: $\mathcal{O}(1)$
- Změna potenciálu $\Phi' - \Phi = -1$
- Amortizovaná složitost: \emptyset

Ostatní instrukce

- Skutečný čas: $\mathcal{O}(\log n)$
- Změna potenciálu $\Phi' - \Phi = 0$
- Amortizovaná složitost: $\mathcal{O}(\log n)$

Amortizovaná složitost operace DeleteMin

$\mathcal{O}(\log n)$

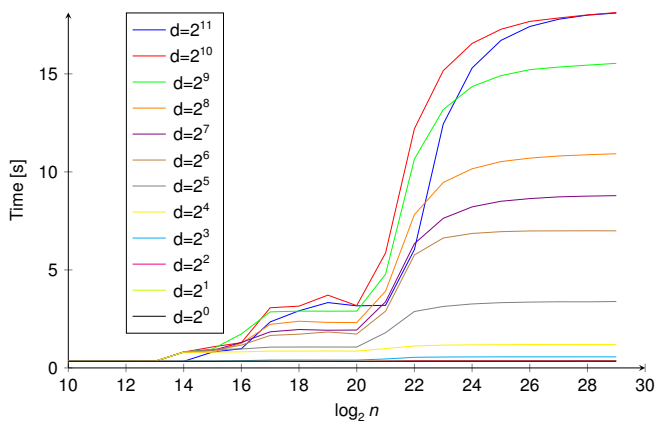
Cvičení:

- Určete přesně výšku d -regulární haldy s n vrcholy.
- d -regulární haldy podobně jako binární je možné uložit v poli. Najděte vzorce pro výpočet pozice otce a synů pro daný prvek v poli.
- Jak byste zvýšili hodnotu klíče v d -regulární haldě?
- Jak byste smazali prvek v daném vrcholu v d -regulární haldě?
- Pro $d > 1$ dokažte $\sum_{h=0}^{\infty} \frac{h}{d^h} = \frac{d}{(d-1)^2}$.
- Dokažte, že amortizovaná složitost operace DecreaseKey v líné binomiální haldě je $\mathcal{O}(\log n)$. Dále určete celkovou složitost k_I operací Insert, k_M operací DeleteMin a k_D operací DecreaseKey.
- Je možné vytvořit haldy, která má amortizovanou složitost operací Insert a DeleteMin lepší než $\mathcal{O}(\log n)$?
- Jaký je maximální možný počet vrcholů ve stromu řádu k ve Fibonacciho haldě?
- Dokažte, že Fibonacciho čísla splňují $\sum_{i=1}^k F_i = F_{k+2} - 1$ a $F_k \geq \left(\frac{1+\sqrt{5}}{2}\right)^k$.
- Jaká je maximální možná výška stromu ve Fibonacciho haldě s n prvky? Může se stát, aby složitost operace DecreaseKey byla $\Omega(n)$?
- Proč Fibonacciho haldy nepovoluje odstranit nejvýše jednoho syna? Uvažujme neoznačující Fibonacciho haldy, která při operaci DecreaseKey pouze odebere daný vrchol od otce, ale neoznačí otce a ani nepokračuje k předkům. Která tvrzení o Fibonacciho haldě přestanou pro neoznačující Fibonacciho haldy platit? Dojde ke zhoršení časových složitostí studovaných operací?

Obsah

- Amortizovaná analýza
- Splay strom
- (a,b)-strom a červeno-černý strom
- Haldy
- Cache-oblivious algorithms
- Hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

Paměťová hierarchie: Triviální program



Přehled časových složitostí

	Binární		Binomiální		Líná binomiální		Fibonacciho	
	worst	amort	worst	amort	worst	amort	worst	amort
Insert	$\log n$	1	$\log n$	1	$\log n$	1	1	1
DecreaseKey	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	n	1
DeleteMin	$\log n$	$\log n$	$\log n$	$\log n$	n	$\log n$	n	$\log n$

- Změnili by se časové složitosti operací, kdybychom ve Fibonacciho haldě dovolili smazat dva syny (nebo libovolný pevný počet synů)?
- Jakou časovou složitost má Dijkstraův algoritmus při použití probíraných hald? Při jakém poměru počtu hran a vrcholů grafu dokážete získat lineární časovou složitost Dijkstraova algoritmu?

Paměťová hierarchie

Příklad velikostí a rychlostí různých typů pamětí

	size	speed
L1 cache	32 KB	223 GB/s
L2 cache	256 KB	96 GB/s
L3 cache	8 MB	62 GB/s
RAM	32 GB	23 GB/s
SDD	112 GB	448 MB/s
HDD	2 TB	112 MB/s
Internet	∞	10 MB/s

Triviální program

```
# Inicializace pole 32-bitových čísel velikosti n
1 for (i=0; i+d<n; i+=d) do
2   A[i] = i+d # Vezmeme každou d-tou pozici a vytvoříme cyklus
3 A[i]=0
# Měříme dobu průběhu cyklu v závislosti na parametrech n a d
4 for (j=0; j< 2^28; j++) do
5   i = A[j] # Dokola procházíme cyklus d-tých pozic
```

Cache-oblivious (Frigo, Leiserson, Prokop, Ramachandran, 1999 [8])

Zjednodušený model paměti

- Uvažujme pouze dvě úrovně paměti: pomalý disk a rychlá cache
- Paměť je rozdělena na bloky (stránky) velikosti B
- Velikost cache je M , takže cache má $P = \frac{M}{B}$ bloků
- Procesor může přistupovat pouze k datům uložených v cache
- Paměť je plně asociativní
- Data se mezi diskem a cache přesouvají po celých blocích a naším cílem je určit počet bloků načtených do cache

Cache-aware algoritmus

Algoritmus zná hodnoty M a B a podle nich nastavuje parametry (např. velikost vrcholu B-stromu při ukládání dat na disk).

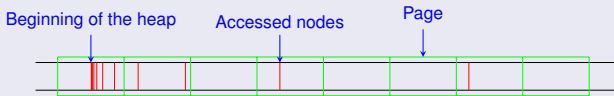
Cache-oblivious algoritmus

- Algoritmus musí efektivně fungovat bez znalosti hodnot M a B . Důsledky:
- Není třeba nastavovat parametry programu, který je tak přenositelnější
 - Algoritmus dobře funguje mezi libovolnými úrovněmi paměti (L1 – L2 – L3 – RAM)

- Pro zjednodušení předpokládáme, že jeden prvek zabírá jednotkový prostor, takže do jednoho bloku se vejde B prvků.
- Předpokládáme, že každý blok z disku může být uložený na libovolné pozici v cache. Tento předpoklad výrazně zjednodušuje analýzu, i když na reálných počítačích moc neplatí, viz https://en.wikipedia.org/wiki/CPU_cache#Associativity.

Cache-oblivious analýza: Binární halda a vyhledávání

Binární halda v poli: Průchod od listu ke kořeni



- Cesta má $\Theta(\log n)$ vrcholů
- Posledních $\Theta(\log B)$ vrcholů leží v nejvýše dvou blocích
- Ostatní vrcholy jsou uloženy v po dvou různých blocích
- $\Theta(\log n - \log B) = \Theta(\log \frac{n}{B})$ přenesených bloků

Binární vyhledávání

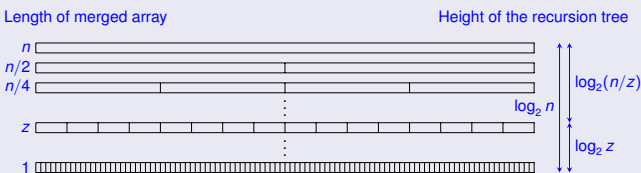
- Porovnáváme $\Theta(\log n)$ prvků s hledaným prvkem
- Posledních $\Theta(\log B)$ prvků je uloženo v nejvýše dvou blocích
- Ostatní prvky jsou uloženy v po dvou různých blocích
- $\Theta(\log n - \log B)$ přenesených bloků

Cache-oblivious analýza: Mergesort

Případ $n \leq M/2$

Celé pole se vejde do cache, takže přenášíme $2n/B + \mathcal{O}(1)$ bloků.

Schéma



Případ $n > M/2$

- Nechť z je maximální velikost pole, která může být setříděná v cache
- Platí $z \leq \frac{M}{2} < 2z$
- Slití jedné úrovně vyžaduje $2\frac{n}{B} + 2\frac{n}{z} + \mathcal{O}(1) = \mathcal{O}(\frac{n}{B})$ přenosů
- Počet přenesených bloků je $\mathcal{O}(\frac{n}{B})(1 + \log_2 \frac{n}{z}) = \mathcal{O}(\frac{n}{B} \log \frac{n}{M})$

Cache-oblivious analýza: Transpozice matic: Triviální přístup

Strategie pro výměnu stránek v cache

- OPT: Optimální off-line algoritmus předpokládající znalost všech přístupů do paměti
- FIFO: Z cache smažeme stránku, která je ze všech stránek v cache nejdelší dobu
- LRU: Z cache smažeme stránku, která je ze všech stránek v cache nejdéle nepoužitá

Triviální algoritmus pro transpozici matice A velikost $k \times k$

```

1 for i ← 1 to k do
2   for j ← i + 1 to k do
3     Swap(Aij, Aji)

```

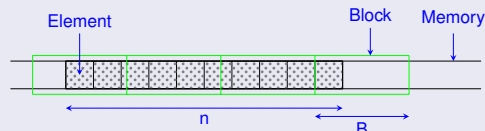
Předpoklady

Uvažujeme pouze případ

- $B < k$: Do jednoho bloku cache se nevejde celá řádka matice
- $P < k$: Do cache se nevejde celý sloupec matice

Cache-oblivious analýza: Scanning

Přečtení souvislého pole (výpočet maxima, součtu a podobně)



- Minimální možný počet přenesených bloků je $\lceil n/B \rceil$.
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$.
- Předpokládáme, že máme k dispozici $\mathcal{O}(1)$ registrů k uložení iterátoru a maxima.

Obrácení pole

Počet přenesených bloků je stejný za předpokladu, že $P \geq 2$.

- Přesněji $\Theta(\max\{1, \log n - \log B\})$. Dále předpokládáme, že $n \geq B$.
- Pro jednoduchost uvažujeme neúspěšné vyhledávání.

- Polovina cache je použita na vstupní pole a druhá polovina na slití pole.
- Pro jednoduchost předpokládáme, že velikosti polí v jedné úrovni rekurze jsou stejné. z odpovídá velikosti pole v úrovni rekurze takové, že dvě pole velikost $z/2$ mohou být slity v jedno pole velikost z .
- Slití všech polí v jedné úrovni do polovičního počtu polí dvojnásobné délky vyžaduje přečtení všech prvků. Navíc je třeba uvažovat nezarovnaní polí a bloků, takže hraniční bloky mohou patřit do dvou polí.
- Funnelsort přeneše $\mathcal{O}(\frac{n}{B} \log_P \frac{n}{B})$ bloků.

Cache-oblivious analýza: Transpozice matic: Triviální přístup

Příklad: Reprezentace matice 5×5 v paměti

11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45	51	52	53	54	55
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

LRU a FIFO strategie

Při čtení matice po sloupcích si cache pamatuje posledních P řádků, takže při čtení prvku $A_{3,2}$ již prvek $A_{3,1}$ není v cache. Počet přenesených bloků je $\Omega(k^2)$.

OPT strategie

- Transpozice prvního řádku/sloupce vyžaduje alespoň $k - 1$ přenosů.
- Nejvýše P prvků z druhého sloupce zůstane v cache.
- Proto transpozice druhého řádku/sloupce vyžaduje alespoň $k - P - 2$ přenosů.
- Transpozice i -tého řádku/sloupce vyžaduje alespoň $\max\{0, k - P - i\}$ přenosů.
- Celkový počet přenosů je alespoň $\sum_{i=1}^{k-P} k - P - i = \Omega((k - P)^2)$.

Cache-aware algoritmus pro transpozici matice A velikost $k \times k$

```
# Nejprve si rozdělíme danou matici na submatice velikosti  $z \times z$ 
1 for ( $i = 0; i < k; i += z$ ) do
2   for ( $j = i; j < k; j += z$ ) do
3     # Transponujeme submatici začínající na pozici ( $i, j$ )
4     for ( $i1 = i; i1 < \min(k, i + z); i1 += z$ ) do
5       for ( $j1 = \max(j, i1 + 1); j1 < \min(k, j + z); j1 += z$ ) do
6         Swap( $A_{i1, j1}, A_{j1, i1}$ )
```

Hodnocení

- Optimální hodnota z závisí na konkrétním počítači
- Využíváme jen jednu úroveň cache
- Při správně zvolené hodnotě z bývá tento postup nejrychlejší

Cache-oblivious analýza: Transpozice matic: Rekurzivní transpozice

```
1 Procedure transpose_on_diagonal ( $A$ )
2   if Matice  $A$  je malá then
3     Transponujeme matici  $A$  triviálním postupem
4   else
5      $A_{11}, A_{12}, A_{21}, A_{22} \leftarrow$  souřadnice submatic
6     transpose_on_diagonal ( $A_{11}$ )
7     transpose_on_diagonal ( $A_{22}$ )
8     transpose_and_swap ( $A_{12}, A_{21}$ )
9 Procedure transpose_and_swap ( $A, B$ )
10  if Matice  $A$  a  $B$  jsou malé then
11    Prohodíme a transponujeme matice  $A$  a  $B$  triviálním postupem
12  else
13     $A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22} \leftarrow$  souřadnice submatic
14    transpose_and_swap ( $A_{11}, B_{11}$ )
15    transpose_and_swap ( $A_{12}, B_{21}$ )
16    transpose_and_swap ( $A_{21}, B_{12}$ )
17    transpose_and_swap ( $A_{22}, B_{22}$ )
```

- Stačilo by předpokládat, že počet bloků je alespoň $\Omega(B)$. Máme-li alespoň $4B$ bloků, pak je postup algebraicky jednodušší.
- Pokud začátek řádky není na začátku bloku, tak je jeden řádek submatice uložen ve dvou blocích.
- Funkce `transpose_and_swap` pracujeme se dvěma submaticemi.
- Celá matice je uložena v alespoň $\frac{k^2}{B}$ blocích paměti.

- Platí pro B-regulární haldu. B-strom má výšku $\Theta(\log_B(n))$.
- Asymptoticky optimální řešení — důkaz je založen na Information theory.

Idea

Rekurzivně rozdělíme na submatice

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad A^T = \begin{pmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{pmatrix}$$

Matice A_{11} a A_{22} se transponují podle stejného schématu, ale A_{12} a A_{21} se prohazují.

Cache-oblivious analýza: Transpozice matic: Rekurzivní transpozice

Analýza počtu přenesených bloků

- Předpoklad „Tall cache“: $M \geq 4B^2$, tj. počet bloků je alespoň $4B$ ☹
- Necht z je maximální velikost submatice, ve které se jeden řádek vejde do jednoho bloku ☹
- Platí: $z \leq B \leq 2z$
- Jedna submatice $z \times z$ je uložena v nejvýše $2z \leq 2B$ blocích
- Dvě submatice $z \times z$ se vejdou do cache ☹
- Transpozice matice typu $z \times z$ vyžaduje nejvýše $4z$ přenosů
- Máme $(k/z)^2$ submatic velikosti z
- Celkový počet přenesených bloků je nejvýše $\frac{k^2}{z^2} \cdot 4z \leq \frac{8k^2}{B} = O\left(\frac{k^2}{B}\right)$
- Tento postup je optimální až na multiplikativní faktor ☹

Cache-oblivious analýza: Reprezentace binárních stromů

Cíl

Sestrojit reprezentaci binárního stromu efektivně využívající cache. Počítáme počet načtených bloků při průchodu cesty z listu do kořene.

Binární halda

Velmi neefektivní: Počet přenesených bloků je $\Theta(\log n - \log B) = \Theta(\log \frac{n}{B})$

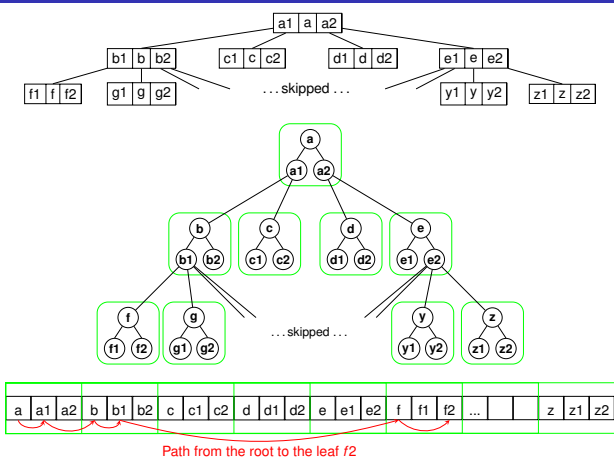
B-regulární halda, B-strom

- Výška stromu je $\log_B(n) + \Theta(1)$ ☹
- Jeden vrchol je uložen v nejvýše dvou blocích
- Počet načtených bloků je $\Theta(\log_B(n))$ ☹
- Nevýhody: cache-aware a chtěli jsme binární strom

Převedení na binární strom

Každý vrchol B-regulární haldy nahradíme binárním stromem.

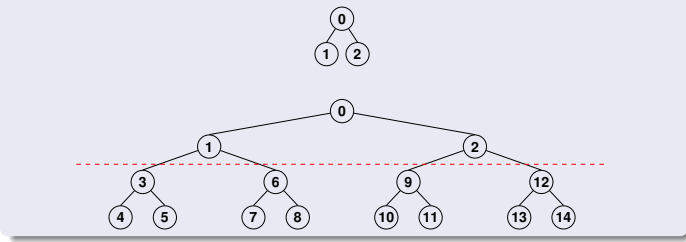
Cache-oblivious analýza: Reprezentace binárních stromů



Rekurzivní „bottom-up“ konstrukce van Emde Boas rozložení

- van Emde Boas rozložení vEB₀ řádu 0 je jeden vrchol
- vEB_k obsahuje jednu „horní“ kopii vEB_{k-1} a každému listu „horní“ kopie má dvě „dolní“ kopie vEB_{k-1}
- V poli jsou nejprve uložena „horní“ kopie a pak následují všechny „dolní“ kopie

Pořadí vrcholů v poli podle van Emde Boas rozložení



Cache-oblivious analýza: Srovnání OPT a LRU strategií

Věta (Sleator, Tarjan [24])

- Nechť s_1, \dots, s_k je posloupnost přístupů do paměti ①
- Nechť P_{OPT} a P_{LRU} je počet bloků v cache pro strategie OPT a LRU ②
- Nechť F_{OPT} a F_{LRU} je počet přenesených bloků ③
- $P_{LRU} > P_{OPT}$

Pak $F_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F_{OPT} + P_{OPT}$

Důsledek

Pokud LRU může uložit dvojnásobný počet bloků v cache oproti OPT, pak LRU má nejvýše dvojnásobný počet přenesených bloků oproti OPT (plus P_{OPT}). ④

Zdvojnásobení velikosti cache nemá většinou vliv na asymptotický počet přenesených bloků

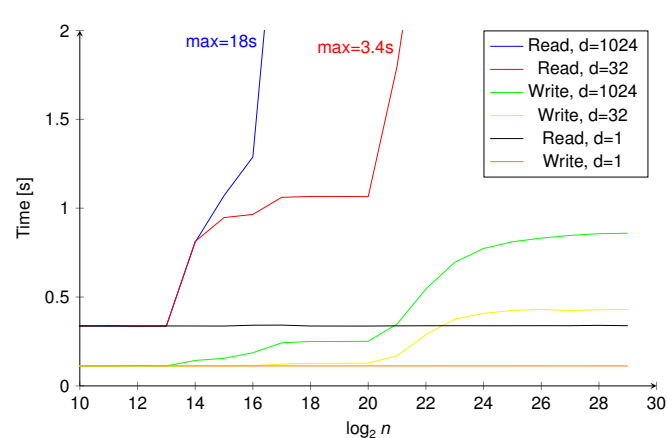
- Scanning: $\mathcal{O}(n/B)$
- Mergesort: $\mathcal{O}(\frac{n}{B} \log \frac{n}{M})$
- Funnelsort: $\mathcal{O}(\frac{n}{B} \log_p \frac{n}{B})$
- The van Emde Boas layout: $\mathcal{O}(\log_B n)$

Cache-oblivious analýza: Srovnání OPT a LRU strategií

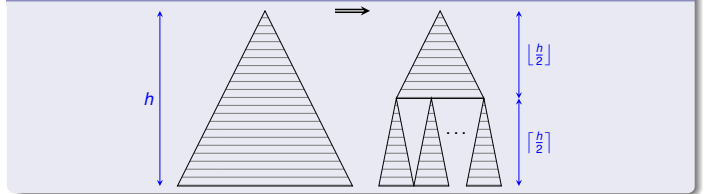
Důkaz ($F_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F_{OPT} + P_{OPT}$)

- Pokud LRU má $f \leq P_{LRU}$ přenesených bloků v podposloupnosti s , pak OPT přeneše alespoň $f - P_{OPT}$ bloků v podposloupnosti s
 - Pokud LRU načte v podposloupnosti f různých bloků, tak podposloupnost obsahuje alespoň f různých bloků
 - Pokud LRU načte v podposloupnosti jeden blok dvakrát, tak podposloupnost obsahuje alespoň $P_{LRU} \geq f$ různých bloků
 - OPT má před zpracováním podposloupnosti nejvýše P_{OPT} bloků z podposloupnosti v cache a zbylých alespoň $f - P_{OPT}$ musí načíst
- Rozdělíme posloupnost s_1, \dots, s_k na podposloupnosti tak, že LRU přeneše P_{LRU} bloků v každé podposloupnosti (kromě poslední)
- Jestliže F'_{OPT} and F'_{LRU} jsou počty přenesených bloků při zpracování libovolné podposloupnosti, pak $F'_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F'_{OPT}$ (kromě poslední)
 - OPT přeneše $F'_{OPT} \geq P_{LRU} - P_{OPT}$ bloků v každé podposloupnosti
 - Tedy $F'_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F'_{OPT}$
- V poslední podposloupnosti platí $F''_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F''_{OPT} + P_{OPT}$
 - Platí $F''_{OPT} \geq F'_{LRU} - P_{OPT}$ a $1 \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}}$
 - Tedy $F''_{LRU} \leq F'_{LRU} + P_{OPT} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F'_{OPT} + P_{OPT}$

Srovnání rychlosti čtení a zápisu z paměti



Rekurzivní „top-down“ konstrukce van Emde Boas rozložení



Výpočet počtu načtených bloků při cestě z kořene do listu

- Nechť $h = \log_2 n$ je výška stromu
- Nechť z je maximální výška podstromu, který se vejde do jednoho bloku
- Platí: $z \leq \log_2 B \leq 2z$
- Počet podstromů výšky z na cestě z kořene do listu je $\frac{h}{z} \leq \frac{2 \log_2 n}{\log_2 B} = 2 \log_B n$
- Počet načtených bloků je $\Theta(\log_B n)$

- s_i značí blok paměti, se kterým program pracuje, a proto musí být načten do cache. Posloupnost s_1, \dots, s_k je pořadí bloků paměti, ve kterém algoritmus pracuje s daty. Při opakovaném přístupu do stejného bloku se blok posloupnosti opakuje.
- Představme si, že OPT strategie pustíme na počítači s P_{OPT} bloky v cache a LRU strategie spustíme na počítači s P_{OPT} bloky v cache.
- Srovnáváme počet přenesených bloků OPT strategie na počítači s P_{OPT} bloky a LRU strategie na počítači s P_{OPT} bloky.
- Formálně: Jestliže $P_{LRU} = 2P_{OPT}$, pak $F_{LRU} \leq 2F_{OPT} + P_{OPT}$.

Srovnání rychlosti čtení a zápisu z paměti

Čtení z paměti

```
# Inicializace pole 32-bitových čísel velikosti n
1 for (i=0; i+d<n; i+=d) do
2   | A[i]=i+d # Vezmeme každou d-tou pozici a vytvoříme cyklus
3 A[i]=0
4 # Měříme dobu průběhu cyklu v závislosti na parametrech n a d
5 for (j=0; j< 2^28; j++) do
6   | i = A[i] # Dokola procházíme cyklus d-tých pozic
```

Zápis do paměti

```
1 mask = (1 << n) - 1
2 # Měříme dobu průběhu cyklu v závislosti na parametrech n a d
3 for (j=0; j< 2^28; j++) do
4   | A[(j*d) & mask] = j # Dokola zapisujeme na d-té pozice
```

Pár triků na závěr

Která varianta je rychlejší a o kolik?

```
# Použijeme modulo:
1 for (j=0; j< 2^28; j++) do
2   | A[(j*d) % n] = j
# Použijeme bitovou konjunktci:
3 mask = n - 1 # Předpokládáme, že n je mocnina dvojky
4 for (j=0; j< 2^28; j++) do
5   | A[(j*d) & mask] = j
```

Jak dlouho poběží výpočet vynecháme-li poslední řádek?

```
1 for (i=0; i+d<n; i+=d) do
2   | A[i]=i+d
3 A[i]=0
4 # Měříme dobu průběhu cyklu v závislosti na parametrech n a d
5 for (j=0; j< 2^28; j++) do
6   | printf("%d\n", i);
```

Cvičení:

- 6.1. Vymyslete co nejkratší funkci na transpozici matic. Celou transpozici je možné implementovat pomocí jedné rekurzivní funkce, kde vlastní rekurzivní volání je na 5 řádků a dalších 5-10 řádků je na deklaraci a zakončení rekurze (v závislosti na zvoleném jazyce). Funkce by měla transponovat i čtvercové matice, jejichž velikost není mocnina dvojky.
- 6.2. V praxi rekurzivní volání značně zpomaluje transponování matic, a proto zkuste vymyslet postup nevyužívající rekurzi (ani zásobník). Dokázali byste efektivně vygenerovat posloupnost pozic $[i_1, j_1], [i_2, j_2], \dots, [i_{\binom{k}{2}}, j_{\binom{k}{2}}]$ matice typu $k \times k$, která transponuje matici (zvoláním $\text{Swap}(A_{i_j, j_i}, A_{j_i, j_j})$ pro všechna $l = 1, 2, \dots, \binom{k}{2}$) v cache-oblivious pořadí? Pro zjednodušení můžete předpokládat, že k je mocnina dvojky.
- 6.3. Podobný problém jako u rekurzivní transpozice matic nastává i u van Emde Boas rozložení. Jak pro danou pozici v poli určit pozice otce a synů v van Emde Boas rozložení? Najděte co nejrychlejší způsob nalezení cesty z daného vrcholu do kořene.
- 6.4. Proveďte cache-oblivious analýzu algoritmů na násobení dlouhých čísel.
- 6.5. Analyzujte lineární algoritmus na hledání mediánu.
- 6.6. Analyzujte následující algoritmy na násobení matic:
 - Dle definice
 - Nejprve se druhá matice ztransponuje a poté se vynásobí s první
 - Rekurzivní dělení na bloky

Jirka Fink

Datové struktury I

109

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování**
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry

Jirka Fink

Datové struktury I

110

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování**
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Jirka Fink

Datové struktury I

112

Literatura

- 1 Navíc obvykle vyžadujeme, aby hešovací funkce šlo spočítat v čase $\mathcal{O}(1)$ a aby funkce bylo možné popsat $\mathcal{O}(1)$ parametry.

Jirka Fink

Datové struktury I

113

• Strassen

- 6.7. Určete střední hodnotu počtu přenesených bloků randomizovaného Quick-sortu.
- 6.8. k -way Merge-sort rozdělí pole na k částí, které rekurzivně seřadí a slije dohromady. Najděte optimální hodnotu k pro cache-aware k -way Merge-sort.
- 6.9.

Jirka Fink

Datové struktury I

109

Hešování

Základní pojmy

- Máme univerzum $U = \{0, 1, \dots, u-1\}$ všech prvků
- Chceme uložit podmnožinu $S \subseteq U$ velikosti n
- Uložíme S do pole velikosti m pomocí hešovací funkce $h: U \rightarrow M$, kde $M = \{0, 1, \dots, m-1\}$
- Dva prvky $x, y \in S$ kolidují, jestliže $h(x) = h(y)$
- Hešovací funkce h je perfektní na S , jestliže h nemá žádnou kolizi S

Nepřátelská podmnožina

Pokud $u \geq mn$, pak pro každou hešovací funkci h existuje $S \subseteq U$ velikosti n taková, že h hešuje všechny prvky z S do jedné příhrádky.

Poznámky

- Není možné sestavit jednu funkci dobře hešující libovolnou podmnožinu $S \subseteq U$
- Pro danou podmnožinu $S \subseteq U$ lze sestavit perfektní hešovací funkci
- Sestrojíme množinu hešovacích funkcí \mathcal{H} takovou, že náhodně zvolená funkce $h \in \mathcal{H}$ hešuje libovolnou podmnožinu S v průměrném případě uspokojivým způsobem

Jirka Fink

Datové struktury I

111

Universální hešování

Cíl

Sestavit systém \mathcal{H} hešovacích funkcí $f: U \rightarrow M$ takový, že náhodně zvolená funkce $f \in \mathcal{H}$ hešuje libovolnou množinu S „většinou dobře“.

Úplně náhodná hešovací funkce

- Systém \mathcal{H} obsahuje všechny funkce $f: U \rightarrow M$
- Platí $P[h(x) = z] = \frac{1}{m}$ pro všechna $x \in U$ a $z \in M$
- Náhodné příhrádky $h(x)$ a $h(y)$ jsou nezávislé pro různé $x, y \in U$
- Nepraktické: k zakódování funkce z \mathcal{H} potřebujeme $\Theta(u \log m)$ bitů
- Někdy se používá k analýze hešování

c-universální systém

Systém hešovacích funkcí \mathcal{H} je c-universální, pokud náhodně zvolená $h \in \mathcal{H}$ splňuje $P[h(x) = h(y)] \leq \frac{c}{m}$ pro každé $x, y \in U$ a $x \neq y$. \odot

Příklad c-universálního systému (Cvičení 7.5)

- $\mathcal{H} = \{h_a(x) = (ax \bmod p) \bmod m; 0 < a < p\}$, kde $p > u$ je prvočíslo

Jirka Fink

Datové struktury I

113

Universální hešování

2-nezávislý systém hešovacích funkcí

Systém hešovacích funkcí \mathcal{H} je 2-nezávislý, pokud náhodně zvolená $h \in \mathcal{H}$ splňuje $P[h(x_1) = z_1 \wedge h(x_2) = z_2] = \mathcal{O}(\frac{1}{m^2})$ pro každé $x_1, x_2 \in U$ a $x_1 \neq x_2$ a $z_1, z_2 \in M$.

k-nezávislý systém hešovacích funkcí

Systém hešovacích funkcí \mathcal{H} je k-nezávislý, pokud náhodně zvolená $h \in \mathcal{H}$ splňuje $P[h(x_i) = z_i \text{ pro všechna } i = 1, \dots, k] = \mathcal{O}(\frac{1}{m^k})$ pro všechna po dvou různá $x_1, \dots, x_k \in U$ a všechna $z_1, \dots, z_k \in M$.

Cvičení 7.7, 7.8, 7.9, 7.10

- k-nezávislý systém hešovacích funkcí je (k-1)-nezávislý
- 2-nezávislý systém hešovacích funkcí je c-universální pro nějaké c
- Najděte 1-universální systém, který není 2-nezávislý
- Jestliže $P[h(x_i) = z_i \text{ pro všechna } i = 1, \dots, k] \leq \frac{1}{m^k}$ pro všechna $z_1, \dots, z_k \in M$, pak $P[h(x_i) = z_i \text{ pro všechna } i = 1, \dots, k] = \frac{1}{m^k}$ pro všechna $z_1, \dots, z_k \in M$

1-nezávislý systém není užitečný

Systém $\mathcal{H} = \{h_a(x) = a; a \in M\}$ je 1-nezávislý, ale nepoužitelný.

Jirka Fink

Datové struktury I

114

Universální hešování: Multiply-mod-prime

Systém Multiply-mod-prime

- Necht p je prvočíslo větší než u a $[p]$ značí $\{0, \dots, p-1\}$
- $h_{a,b}(x) = (ax + b \bmod p) \bmod m$
- $\mathcal{H} = \{h_{a,b}; a, b \in [p], a \neq 0\}$
- Systém \mathcal{H} je 1-universální a 2-nezávislý, ale není 3-nezávislý

Lemma

Pro libovolná různá $x_1, x_2 \in [p]$ rovnice

$$y_1 = ax_1 + b \bmod p$$
$$y_2 = ax_2 + b \bmod p$$

definují bijekci mezi $\{(a, b) \in [p]^2 \mid (y_1, y_2) \in [p]^2\}$ a dále bijekci mezi $\{(a, b) \in [p]^2; a \neq 0\}$ a $\{(y_1, y_2) \in [p]^2; y_1 \neq y_2\}$.

Důkaz

- Pro danou dvojici (y_1, y_2) existuje jedinná dvojice (a, b) splňující rovnice
 - Odečtením dostáváme $a(x_1 - x_2) \equiv y_1 - y_2 \pmod p$
 - V tělese \mathbb{Z}_p dostáváme $a = (y_1 - y_2)(x_1 - x_2)^{-1}$, $b = y_1 - ax_1$
- Zřejmě platí $a = 0$ právě tehdy, když $y_1 = y_2$

Jirka Fink

Datové struktury I

115

Universální hešování: Multiply-mod-prime

Systém Multiply-mod-prime

- $h_{a,b}(x) = (ax + b \bmod p) \bmod m$, kde p je prvočíslo větší než u
- $\mathcal{H} = \{h_{a,b}; a, b \in [p], a \neq 0\}$

Lemma

Pro libovolná různá $x_1, x_2 \in [p]$ rovnice

$$y_1 = ax_1 + b \bmod p$$
$$y_2 = ax_2 + b \bmod p$$

definují bijekci mezi $(a, b) \in [p]^2$ a $(y_1, y_2) \in [p]^2$.

Systém \mathcal{H} je 2-nezávislý

- Počet y_1 takových, že $z_1 = y_1 \bmod m$ je nejvýše $\lceil \frac{p}{m} \rceil$
- Počet (y_1, y_2) takových, že $z_1 = y_1 \bmod m$ a $z_2 = y_2 \bmod m$ je nejvýše $\lceil \frac{p}{m} \rceil^2$
- Počet funkcí $h_{a,b}$ takových, že $h_{a,b}(x_1) = z_1$ a $h_{a,b}(x_2) = z_2$ je nejvýše $\lceil \frac{p}{m} \rceil^2$
- $P[h_{a,b}(x_1) = z_1 \wedge h_{a,b}(x_2) = z_2] \leq \lceil \frac{p}{m} \rceil^2 \frac{1}{p(p-1)} \leq \left(\frac{p+m}{m}\right)^2 \frac{2}{p^2} \leq \left(\frac{2p}{m}\right)^2 \frac{2}{p^2} = \mathcal{O}\left(\frac{1}{m^2}\right)$

Jirka Fink

Datové struktury I

117

Universální hešování: Poly-mod-prime

Systém Poly-mod-prime

- Necht p je prvočíslo větší než u a $k \geq 1$ celé číslo
- $h_{a_0, \dots, a_{k-1}}(x) = (\sum_{i=0}^{k-1} a_i x^i \bmod p) \bmod m$
- $\mathcal{H} = \{h_{a_0, \dots, a_{k-1}}; a_0, \dots, a_{k-1} \in [p]\}$

Cvičení 7.16: k-nezávislost

Systém Poly-mod-prime je k-nezávislý.

Jirka Fink

Datové struktury I

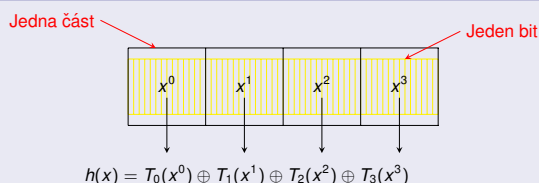
119

Universální hešování: Tabulkové hešování

Tabulkové hešování

- Předpokládáme, že $u = 2^w$ a $m = 2^l$ a w je násobek d
- Bitový zápis čísla $x \in U$ rozdělíme na d částí x^0, \dots, x^{d-1} po $\frac{w}{d}$ bitech
- Pro každé $i \in [d]$ vybereme náhodnou hešovací funkci $T_i: [2^{w/d}] \rightarrow M$
- Hešovací funkce je $h(x) = T_0(x^0) \oplus \dots \oplus T_{d-1}(x^{d-1})$

Ilustrativní příklad



Univerzalita

Tabulkové hešování je 3-nezávislé, ale není 4-nezávislé.

Jirka Fink

Datové struktury I

121

Universální hešování: Multiply-mod-prime

Systém Multiply-mod-prime

- $h_{a,b}(x) = (ax + b \bmod p) \bmod m$, kde p je prvočíslo větší než u
- $\mathcal{H} = \{h_{a,b}; a, b \in [p], a \neq 0\}$

Lemma

Pro libovolná různá $x_1, x_2 \in [p]$ rovnice

$$y_1 = ax_1 + b \bmod p$$
$$y_2 = ax_2 + b \bmod p$$

definují bijekci mezi $\{(a, b) \in [p]^2; a \neq 0\}$ a $\{(y_1, y_2) \in [p]^2; y_1 \neq y_2\}$.

Systém \mathcal{H} je 1-universální

- Pro $x_1 \neq x_2$ platí $h_{a,b}(x_1) = h_{a,b}(x_2)$ právě tehdy, když $y_1 \equiv y_2 \pmod m$ a $y_1 \neq y_2$
- Pro y_1 existuje nejvýše $\lceil \frac{p}{m} \rceil - 1$ hodnot y_2 takových, že $y_1 \equiv y_2 \pmod m$ a $y_1 \neq y_2$
- Počet takových dvojic (y_1, y_2) je nejvýše $p(\lceil \frac{p}{m} \rceil - 1) \leq p(\frac{p+m-1}{m} - 1) \leq \frac{p(p-1)}{m}$
- Počet funkcí $h_{a,b} \in \mathcal{H}$ způsobujících kolizi $h_{a,b}(x_1) = h_{a,b}(x_2)$ je nejvýše $\frac{p(p-1)}{m}$
- Tudíž $P[h_{a,b}(x_1) = h_{a,b}(x_2)] \leq \frac{p(p-1)}{m|H|} \leq \frac{1}{m}$.

Jirka Fink

Datové struktury I

116

Universální hešování: Multiply-mod-prime

Systém Multiply-mod-prime

- $h_{a,b}(x) = (ax + b \bmod p) \bmod m$, kde p je prvočíslo větší než u
- $\mathcal{H} = \{h_{a,b}; a, b \in [p], a \neq 0\}$

Systém \mathcal{H} není 3-nezávislý

- Zvolme $x_1, x_2, x_3 \in U$ po dvou různé takové, že $x_1 + x_2 = 2x_3$
- Zvolme $z_1, z_2, z_3 \in M$ takové, že $z_1 + z_2 = 2z_3$
- Předpokládejme, že $p = m$ je prvočíslo
- Jestliže $h(x_1) = z_1$ a $h(x_2) = z_2$, pak $h(x_3) = z_3$
 - $2h(x_3) \equiv_p 2(ax_3 + b) \equiv_p (ax_1 + b) + (ax_2 + b) \equiv_p z_1 + z_2 \equiv_p 2z_3$
- Podmíněná pravděpodobnost $P[h(x_3) = z_3 \mid h(x_1) = z_1 \wedge h(x_2) = z_2] = 1$
- $P[h(x_3) = z_3 \wedge h(x_1) = z_1 \wedge h(x_2) = z_2] = P[h(x_3) = z_3 \mid h(x_1) = z_1 \wedge h(x_2) = z_2] P[h(x_1) = z_1 \wedge h(x_2) = z_2] = P[h(x_1) = z_1 \wedge h(x_2) = z_2] = \Theta\left(\frac{1}{m^2}\right)$, což není $\mathcal{O}\left(\frac{1}{m^3}\right)$

Jirka Fink

Datové struktury I

118

Universální hešování: Multiply-shift

Multiply-shift

- Předpokládáme, že $u = 2^w$ a $m = 2^l$
- $h_a(x) = (ax \bmod 2^w) \gg (w-l)$
- $\mathcal{H} = \{h_a; a \text{ je liché } w\text{-bitové číslo}\}$

Implementace v C

```
uint64_t hash(uint64_t x, uint64_t l, uint64_t a)
{ return (a*x) >> (64-l); }
```

2-nezávislost (bez důkazu)

Pro každé $x_1, x_2 \in [2^w]$, $x_1 \neq x_2$ a $z_1, z_2 \in M$ platí $P[h(x_1) = z_1 \wedge h(x_2) = z_2] = \frac{1}{m^2}$.

Jirka Fink

Datové struktury I

120

Universální hešování: Tabulkové hešování

Tabulkové hešování

- Předpokládáme, že $u = 2^w$ a $m = 2^l$ a w je násobek d
- Bitový zápis čísla $x \in U$ rozdělíme na d částí x^0, \dots, x^{d-1} po $\frac{w}{d}$ bitech
- Pro každé $i \in [d]$ vybereme náhodnou hešovací funkci $T_i: [2^{w/d}] \rightarrow M$
- Hešovací funkce je $h(x) = T_0(x^0) \oplus \dots \oplus T_{d-1}(x^{d-1})$

Univerzalita

Tabulkové hešování je 3-nezávislé, ale není 4-nezávislé.

Důkaz 2-nezávislosti (3-nezávislost je Cvičení 7.12)

- Mějme dva prvky x_1 a x_2 lišící se v i -tých částech
- Necht $h_i(x) = T_0(x^0) \oplus \dots \oplus T_{i-1}(x^{i-1}) \oplus T_{i+1}(x^{i+1}) \oplus \dots \oplus T_{d-1}(x^{d-1})$
- $P[h(x_1) = z_1] = P[h_i(x_1) \oplus T_i(x_i^i) = z_1] = P[T_i(x_i^i) = z_1 \oplus h_i(x_1)] = \frac{1}{m}$ \odot
- Náhodné jevy $h(x_1) = z_1$ a $h(x_2) = z_2$ jsou nezávislé
 - Náhodné proměnné $T_i(x_i^i)$ a $T_i(x_i^i)$ jsou nezávislé
 - Náhodné jevy $T_i(x_i^i) = z_1 \oplus h_i(x_1)$ a $T_i(x_i^i) = z_2 \oplus h_i(x_2)$ jsou nezávislé
- $P[h(x_1) = z_1 \wedge h(x_2) = z_2] = P[h(x_1) = z_1]P[h(x_2) = z_2] = \frac{1}{m^2}$

Jirka Fink

Datové struktury I

122

- $T_i(x_i^j)$ nabývá všech hodnot z M se stejnou pravděpodobností $\frac{1}{m}$ a náhodné proměnné $T_i(x_i^j)$ a $z_1 \oplus h_i(x_1)$ jsou nezávislé.

Universální hešování: Tabulkové hešování

Tabulkové hešování není 4-nezávislé

- Zvolíme prvky x_1, x_2, x_3 a x_4 takové, že
 - části x_1 splňují $x_1^0 = 0, x_1^1 = 0, x_1^i = 0$ pro $i \geq 2$
 - části x_2 splňují $x_2^0 = 1, x_2^1 = 0, x_2^i = 0$ pro $i \geq 2$
 - části x_3 splňují $x_3^0 = 0, x_3^1 = 1, x_3^i = 0$ pro $i \geq 2$
 - části x_4 splňují $x_4^0 = 1, x_4^1 = 1, x_4^i = 0$ pro $i \geq 2$
- Jestliže $h(x_1) = h(x_2) = h(x_3) = z$, pak $h(x_4) = z$
 - $Z h(x_1) = h(x_2) = z$ plyne $T_0(0) = T_0(x_1^0) = h_0(x_1) \oplus z = h_0(x_2) \oplus z = T_0(x_2^0) = T_0(1)$
 - $Z h(x_1) = h(x_3) = z$ plyne $T_1(0) = T_1(x_1^1) = h_1(x_1) \oplus z = h_1(x_3) \oplus z = T_1(x_3^1) = T_1(1)$
 - $Z h(x_1) = h(x_2) = h(x_3) = z$ plyne $h(x_4) = h_{0,1}(x_4) \oplus T_0(1) \oplus T_1(1) = h_{0,1}(x_4) \oplus T_0(0) \oplus T_1(0) = h(x_1) = z$, kde $h_{0,1}(x) = T_2(x^2) \oplus \dots \oplus T_{d-1}(x^{d-1})$
- Podmíněná pravděpodobnost $P[h(x_4) = z | h(x_1) = h(x_2) = h(x_3) = z] = 1$
- $P[h(x_1) = h(x_2) = h(x_3) = h(x_4) = z] = P[h(x_4) = z | h(x_1) = h(x_2) = h(x_3) = z] \cdot P[h(x_1) = h(x_2) = h(x_3) = z] = \frac{1}{m^3}$, což není $\mathcal{O}(\frac{1}{m^4})$

Universální hešování: Vektory

Multiply-shift pro vectory pevné délky k

- Chceme hešovat vektor $x_1, \dots, x_d \in U = [2^w]$ do $S = [2^l]$, zvolme $v \geq w + l - 1$
- $h_{a_1, \dots, a_d, b}(x_1, \dots, x_d) = ((b + \sum_{i=1}^d a_i x_i) \bmod 2^v) \gg (w - l)$
- $\mathcal{H} = \{h_{a_1, \dots, a_d, b}; a_1, \dots, a_d, b \in [2^v]\}$
- Systém \mathcal{H} je 2-nezávislý (bez důkazu)

Poly-mod-príme pro různě dlouhé řetězce I

- Chceme hešovat řetězec $x_0, \dots, x_d \in U$ do $[p]$, kde $p \geq u$ je prvočíslo
- $h_a(x_0, \dots, x_d) = \sum_{i=0}^d x_i a^i \bmod p$
- $\mathcal{H} = \{h_a; a \in [p]\}$
- $P[h_a(x_0, \dots, x_d) = h_a(x'_0, \dots, x'_d)] \leq \frac{d+1}{p}$ pro různě řetězce délek $d' \leq d$.

Poly-mod-príme pro různě dlouhé řetězce II

- Chceme hešovat řetězec $x_0, \dots, x_d \in U$ do M , kde $p \geq m$ je prvočíslo
- $h_{a,b,c}(x_0, \dots, x_d) = (b + c \sum_{i=0}^d x_i a^i \bmod p) \bmod m$
- $\mathcal{H} = \{h_{a,b,c}; a, b, c \in [p]\}$
- $P[h_{a,b,c}(x_0, \dots, x_d) = h_{a,b,c}(x'_0, \dots, x'_d)] \leq \frac{2}{p}$ pro různě řetězce délek $d' \leq d \leq \frac{p}{m}$.

Obsah

- Amortizovaná analýza
- Splay strom
- (a,b)-strom a červeno-černý strom
- Halcy
- Cache-oblivious algorithms
- Hešování
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- Geometrické datové struktury
- Dynamizace
- Bloom Filtry
- Literatura

- Předpokládáme, že se každým míčem trefíme do právě jednoho koše, do každého koše se trefíme se stejnou pravděpodobností a jednotlivé hody jsou nezávislé. Důkaz: i -tý míč padne do prázdného koše s pravděpodobností $\frac{m-i+1}{m}$, takže pravděpodobnost, že v každém koši bude nejvýše jeden míč, je $\prod_{i=1}^{n-1} \frac{m-i}{m}$. Použitím aproximací prvního řádu funkce $e^x \sim 1 + x$ dostáváme

$$\prod_{i=1}^{n-1} \left(1 + \frac{-i}{m}\right) \sim \prod_{i=1}^{n-1} e^{-\frac{i}{m}} = e^{-\frac{\sum_{i=1}^{n-1} i}{m}} = e^{-\frac{\binom{n}{2}}{m}} \sim e^{-\frac{n^2}{2m}}$$

Perfektní hešování

Lemma (Cvičení 7.2)

Čekáme-li na událost, která nastane v jednom kroku s pravděpodobností p (nezávisle na ostatních krocích), pak $E[\# \text{ kroků}] = \frac{1}{p}$.

Narozeninový paradox

Pokud n míčů hodíme do $m \geq n$ košů, pak pravděpodobnost, že v každém koši je nejvýše jeden míč, je \mathcal{O}

$$\prod_{i=1}^{n-1} \frac{m-i}{m} \sim e^{-\frac{n^2}{2m}}$$

- $\prod_{i=1}^n \frac{m-i+1}{m} = \prod_{i=1}^{n-1} \left(1 + \frac{-i}{m}\right) \sim \prod_{i=1}^{n-1} e^{-\frac{i}{m}} = e^{-\frac{\sum_{i=1}^{n-1} i}{m}} = e^{-\frac{\binom{n}{2}}{m}} \sim e^{-\frac{n^2}{2m}}$

Důsledek

Pro danou podmnožinu $S \subseteq U$ velikosti n a pro $m = \Theta(n^2)$ lze najít perfektní hešovací funkci do tabulky velikosti m tak, že vyzkoušíme v průměru $\mathcal{O}(1)$ náhodných hešovacích funkcí.

FKS statické hešování (Fredman, Komlós, Szemerédi [6])

Cíl: Dvouúrovňové perfektní hešování do paměti velikosti $\Theta(n)$

- V první úrovni hešujeme všechny prvky do tabulky velikosti $\Theta(n)$
- Nechť n_j je počet prvků v j -té přihrádce
- V druhé úrovni pro každou přihrádku j sestojíme perfektní hešovací funkci do tabulky velikosti $\Theta(n_j^2)$
- K volbě hešovací funkce používáme c -universální systém
- Operaci Find máme v konstantním čase výpočtem dvou hešovacích funkcí

Markovova nerovnost

Jestliže X je nezáporná náhodná veličina a $c > 1$, pak $P[X < cE[X]] > \frac{1}{c}$.

Nalezení primární hešovací funkce s nejvýše n kolizemi

- $E[\# \text{ kolizí}] = \sum_{\{x,y\}} P[h(x) = h(y)] \leq \binom{n}{2} \frac{c}{m} \leq \frac{cn^2}{2m} \leq \frac{n}{2}$ pro $m = \lceil cn \rceil$
- $P[\# \text{ kolizí} < n] \geq P[\# \text{ kolizí} < 2E[\# \text{ kolizí}]] \geq \frac{1}{2}$
- Očekávaný počet pokusů na nalezení hešovací funkce s nejvýše n kolizemi je menší než 2
- Paměťová i očekávaná časová složitost je $\mathcal{O}(n)$

Cíl: Dvouúrovňové perfektní hešování do tabulky velikosti $\Theta(n)$

- V první úrovni hešujeme všechny prvky do tabulky velikosti $\Theta(n)$
- Nechť n_j je počet prvků v j -té přihrádce
- V druhé úrovni pro každou přihrádku j sestojíme perfektní hešovací funkci do tabulky velikosti $\Theta(n_j^2)$
- K volbě hešovací funkce používáme c -universální systém
- Operaci Find máme v konstantním čase výpočtem dvou hešovacích funkcí

Nalezení sekundárních perfektních hešovacích funkcí

- $E[\# \text{ kolizí v } j\text{-té tabulce}] \leq \frac{cn_j^2}{2m_j} \leq \frac{1}{2}$ pro $m_j = \lceil cn_j^2 \rceil$
- $P[\# \text{ kolizí v } j\text{-té tabulce} < 1] > \frac{1}{2}$
- Součet velikostí sekundárních tabulek je $\mathcal{O}(n)$
 - $\sum_{j=1}^m m_j = \sum_{j=1}^m \lceil cn_j^2 \rceil \leq m + c \sum_{j=1}^m n_j + 2c \sum_{j=1}^m \binom{n_j}{2} = m + cn + 2c(\# \text{ kolizí primární tabulky}) = \mathcal{O}(n)$
- Očekávaná časová složitost nalezení všech sekundárních hešovacích funkcí je $\sum_{j=1}^m \mathcal{O}(m_j) = \mathcal{O}(n)$

Hešování se separovanými řetězci

Popis

V přihrádce j jsou uloženy všechny prvky $i \in S$ splňující $h(i) = j$ ve spojovém seznamu, dynamickém poli nebo vyhledávacím stromě.

Implementace

- `std::unordered_map` v C++
- `Dictionary` v C#
- `HashMap` v Java
- `Dictionary` v Python

- Systém hešovacích funkcí \mathcal{H} je silně k -nezávislý, pokud náhodně zvolená $h \in \mathcal{H}$ splňuje $P[h(x_i) = z_i \text{ pro všechna } i = 1, \dots, k] = \frac{1}{m^k}$ pro všechna po dvou různá $x_1, \dots, x_k \in U$ a všechna $z_1, \dots, z_k \in M$.
- Druhá rovnost plyne z linearity střední hodnoty, druhá z definice střední hodnoty a třetí z 1-nezávislosti.
- Druhá rovnost plyne z distribučního zákona a linearity střední hodnoty a poslední rovnost plyne ze silné 2-nezávislosti.

Hešování se separovanými řetězci: Nejdelší řetězec

Definice

Posloupnost náhodných jevů $E_n, n \in \mathbb{N}$ se vyskytuje s velkou pravděpodobností, pokud existují $c > 0$ a $n_0 \in \mathbb{N}$ takové, že pro každé $n \geq n_0$ platí $P[E_n] \geq 1 - \frac{c}{n^c}$.

Triviální příklad

Jestliže náhodně hodíme n míčů do n košů, pak s velkou pravděpodobností jsou alespoň dva koše neprázdné. \odot

Délka nejdelšího řetězce

Pokud $\alpha = \Theta(1)$ a systém hešovacích funkcí je úplně náhodný, pak délka nejdelšího řetězce $\max_{j \in M} A_j = \Theta\left(\frac{\log n}{\log \log n}\right)$ s velkou pravděpodobností. Platí i pro

- $\frac{\log n}{\log \log n}$ -nezávislý systém (Schmidt, Siegel, Srinivasan [23])
- tabulkové hešování (Pătraşcu, Thorup [21])

Očekávaná délka nejdelšího řetězce (Důsledek, Cvičení)

Pokud $\alpha = \Theta(1)$ a systém hešovacích funkcí je úplně náhodný, pak očekávaná délka nejdelšího řetězce je $E[\max_{j \in M} A_j] = \Theta\left(\frac{\log n}{\log \log n}\right)$.

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Hešování se separovanými řetězci: Počet prvků v přihrádce

Značení

- $\alpha = \frac{n}{m}$ je faktor zaplnění; předpokládáme $\alpha = \Theta(1)$
- l_{ij} je náhodná proměnná indikující, zda i -tý prvek patří do j -tého koše
- $A_j = \sum_{i \in S} l_{ij}$ je počet prvků v j -té přihrádce

Pozorování

Pokud je hešovací systém silně 1-nezávislý, pak očekávaný počet prvků v přihrádce je $E[A_j] = \alpha \cdot \odot$

- $E[A_j] = E[\sum_{i \in S} l_{ij}] = \sum_{i \in S} E[l_{ij}] = \sum_{i \in S} P[h(i) = j] = \sum_{i \in S} \frac{1}{m} = \frac{n}{m} \odot$

Lemma

Pokud je hešovací systém silně 2-nezávislý, pak

- $E[A_j^2] = \alpha(1 + \alpha - 1/m)$
 - $E[A_j^2] = E[(\sum_{i \in S} l_{ij})(\sum_{k \in S} l_{kj})] = \sum_{i \in S} E[l_{ij}^2] + \sum_{i, k \in S, i \neq k} E[l_{ij} l_{kj}] = \sum_{i \in S} P[h(i) = j] + \sum_{i, k \in S, i \neq k} E[h(i) = j \text{ a } h(k) = j] = \alpha + \frac{n(n-1)}{m^2} \odot$
- $\text{Var}(A_j) = \alpha(1 - 1/m)$
 - $\text{Var}(A_j) = E[A_j^2] - E^2[A_j] = \alpha(1 + \alpha - 1/m) - \alpha^2$

Hešování se separovanými řetězci: Složitost operace Find

Očekávaný počet porovnání při úspěšné operaci Find

- Celkový počet porovnání při vyhledání všech prvků dělíme počtem prvků
- Předpokládáme silnou 2-nezávislost hešovacího systému
- Celkový počet porovnání při vyhledání všech prvků je $\sum_j \sum_{k=1}^{A_j} k = \sum_j \frac{A_j(A_j+1)}{2}$
- Očekávaný počet porovnání je $1 + \frac{\alpha}{2} - \frac{1}{2m}$
 - $E\left[\frac{1}{n} \sum_j \frac{A_j(A_j+1)}{2}\right] = \frac{1}{2n}(E[\sum_j A_j] + \sum_j E[A_j^2]) = \frac{1}{2n}(n + m\alpha(1 + \alpha - \frac{1}{m}))$

Očekávaný počet porovnání při neúspěšné operaci Find

- Počet porovnání při neúspěšném hledání prvku x je počet prvků $i \in S$ splňující $h(i) = h(x)$
- Tedy chceme spočítat $E[|\{i \in S; h(i) = h(x)\}|]$
- Předpokládáme c -universální hešovací systém
- Použitím linearity střední hodnoty dostáváme
 - $E[|\{i \in S; h(i) = h(x)\}|] = \sum_{i \in S} P[h(i) = h(x)] \leq \sum_{i \in S} \frac{c}{m} = c\alpha$

- $P[E_n] = 1 - \frac{1}{n^{c-1}} \geq 1 - \frac{1}{n^c}$ pro $n \geq 3$

Délky nejdelšího řetězce

Pokud $\alpha = \Theta(1)$ a systém hešovacích funkcí je úplně náhodný, pak délka nejdelšího řetězce $\max_{j \in M} A_j = \Theta\left(\frac{\log n}{\log \log n}\right)$ s velkou pravděpodobností.

Chernoffův odhad

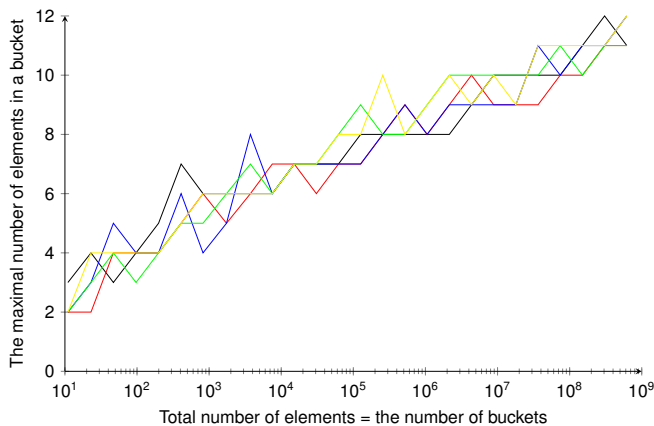
Nechť X_1, \dots, X_n jsou nezávislé náhodné proměnné mající hodnoty $\{0, 1\}$. Označme $X = \sum_{i=1}^n X_i$ a $\mu = E[X]$. Pak pro každé $c > 0$ platí

$$P[X > c\mu] < \frac{e^{(c-1)\mu}}{c^{c\mu}}$$

Důkaz: $\max_{j \in M} A_j = \mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ s velkou pravděpodobností

- Nechť $\epsilon > 0$ a $c = (1 + \epsilon) \frac{\log n}{\mu \log \log n}$. Tedy $c\mu = (1 + \epsilon) \frac{\log n}{\log \log n}$
- Platí $P[\max_j A_j > c\mu] = P[\exists j : A_j > c\mu] \leq \sum_j P[A_j > c\mu] = mP[A_1 > c\mu]$
- Aplikujeme Chernoffův odhad na proměnné I_i pro $i \in S$: $\mu = E[A_1] = \alpha$
- Platí $P[\max_j A_j > c\mu] \leq mP[A_1 > c\mu] < me^{-\mu} e^{c\mu - c\mu \log c}$

Hešování se separovanými řetězci: Nejdelší řetězec: 5 pokusů



- Pravděpodobnost, že množina příhrádek T obsahuje více než $e\alpha d \log n$, se počítá pro náhodnou volbu hešovací funkce $h \in \mathcal{H}$. Proto není možné zvolit zákeřnou množinu příhrádek pro danou hešovací funkci.
- Zjednodušeně řečeno, $\Theta(\log n)$ obsahuje $\Theta(\log n)$ prvků s velkou pravděpodobností.

- $\mathcal{O}(\log n)$ různých prvků je v tabulce velikosti $\Omega(n)$ uloženo v různých příhrádkách s velkou pravděpodobností.
- Posloupnost hledání rozdělíme na podposloupnosti délky $\Omega(\log n)$. Prvky z každé podposloupnosti jsou uloženy v různých příhrádkách s velkou pravděpodobností (narozeninový paradox), a tyto příhrádky nejvýše $\mathcal{O}(\log n)$ prvků.
- Jestliže se hledané prvky mohou opakovat, pak si výsledky posledních $\log n$ hledání pamatujeme v cache.
- Na reálných počítačích by si mohla L1 cache pamatovat prvky z posledních $\log n$ prohledávaných příhrádek.

Důkaz: $\max_{j \in M} A_j = \mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ s velkou pravděpodobností

- Nechť $\epsilon > 0$ a $c = (1 + \epsilon) \frac{\log n}{\mu \log \log n}$. Tedy $c\mu = (1 + \epsilon) \frac{\log n}{\log \log n}$

$$\begin{aligned}
 P[\max_j A_j > c\mu] &< me^{-\mu} e^{c\mu - c\mu \log c} \\
 &= me^{-\mu} e^{(1+\epsilon) \frac{\log n}{\log \log n} - (1+\epsilon) \frac{\log n}{\log \log n} \log\left(\frac{(1+\epsilon) \log n}{\mu \log \log n}\right)} \\
 &= me^{-\mu} e^{(1+\epsilon) \frac{\log n}{\log \log n} - (1+\epsilon) \log n + (1+\epsilon) \frac{\log n}{\log \log n} \log\left(\frac{\mu}{1+\epsilon} \log \log n\right)} \\
 &= me^{-\mu} e^{(1+\epsilon) \frac{\log n}{\log \log n} - (1+\epsilon) \log n + (1+\epsilon) \frac{\log n}{\log \log n} \log\left(\frac{\mu}{1+\epsilon} \log \log n\right)} \\
 &= me^{-\mu} n^{\frac{1+\epsilon}{\log \log n} - (1+\epsilon) + \frac{1+\epsilon}{\log \log n} \log\left(\frac{\mu}{1+\epsilon} \log \log n\right)} \\
 &= \frac{m}{n^{1+\frac{\epsilon}{2}}} e^{-\mu} n^{-\frac{\epsilon}{2} + \frac{1+\epsilon}{\log \log n} + (1+\epsilon) \frac{\log\left(\frac{\mu}{1+\epsilon} \log \log n\right)}{\log \log n}} \\
 &< \frac{1}{\alpha n^{\frac{\epsilon}{2}}} e^{-\mu} n^0 < \frac{1}{n^{\frac{\epsilon}{2}}} \dots \text{pro dostatečně velká } n
 \end{aligned}$$

Protože $-\frac{\epsilon}{2} + \frac{1+\epsilon}{\log \log n} + (1+\epsilon) \frac{\log\left(\frac{\mu}{1+\epsilon} \log \log n\right)}{\log \log n} < 0$ pro dostatečně velká n .

- Tedy $P[\max_j A_j \leq (1 + \epsilon) \frac{\log n}{\log \log n}] > 1 - \frac{1}{n^{\frac{\epsilon}{2}}}$.

Hešování se separovanými řetězci: Amortizovaná složitost hledání

Lemma: Počet prvků v $\Theta(\log n)$ příhrádkách

Jestliže $\alpha = \Theta(1)$, systém hešovacích funkcí je úplně náhodný, pak v pevných $d \log n$ příhrádkách T je nejvýše $e\alpha d \log n$ prvků s velkou pravděpodobností, kde $d \geq 0$. ① ②

Důkaz

- Nechť $X_i = 1$, pokud $h(i) \in T$, jinak $X_i = 0$, kde $i \in S$
- Počet prvků v příhrádkách T je $X = \sum_i X_i$
- Očekávaný počet prvků v T je $\mu = E[X] = E[\sum_{i \in T} A_i] = |T|E[A_i] = \alpha d \log n$
- Chernoff: $P[X > c\mu] < \frac{e^{(c-1)\mu}}{c^{c\mu}}$
- $P[X > c\mu] < e^{d\alpha \log n (c-1 - c \log c)} = n^{d\alpha(c-1 - c \log c)} = n^{-d\alpha}$ pro $c = e$
- $P[X \leq c\mu] \geq 1 - \frac{1}{n^{d\alpha}}$

Hešování se separovanými řetězci: Amortizovaná složitost hledání

Lemma: Počet prvků v $\Theta(\log n)$ příhrádkách

Jestliže $\alpha = \Theta(1)$, systém hešovacích funkcí je úplně náhodný, pak v pevných $d \log n$ příhrádkách T je nejvýše $e\alpha d \log n$ prvků s velkou pravděpodobností, kde $d \geq 0$.

Lemma (Cvičení)

Jestliže $\alpha = \Theta(1)$ a velikost $S \subseteq U$ je $\mathcal{O}(\log n)$, pak funkce vybraná z úplně náhodného hešovacího systému je perfektní na S s velkou pravděpodobností. ①

Amortizovaná složitost pro hledání $\Omega(\log n)$ prvků (Pätraşcu [19])

Jestliže $\alpha = \Theta(1)$, systém hešovacích funkcí je úplně náhodný a hledáme $\Omega(\log n)$ různých prvků, pak amortizovaná složitost hledání jednoho prvku je $\mathcal{O}(1)$ s velkou pravděpodobností. ②

Amortizovaná složitost pro hledání $\Omega(\log n)$ prvků (Pätraşcu [19])

Jestliže $\alpha = \Theta(1)$, systém hešovacích funkcí je úplně náhodný a máme posloupnost operací Find délky $\Omega(\log n)$, při které můžeme využívat cache $\Theta(\log n)$ prvků, pak amortizovaná složitost hledání jednoho prvku je $\mathcal{O}(1)$ s velkou pravděpodobností. ③ ④

Více-příhrádkové hešování se separovanými řetězci

2-příhrádkové hešování

Prvek x může být uložen v příhrádce $h_1(x)$ nebo $h_2(x)$ a nový prvek vkládáme do příhrádky s menším počtem prvků, kde h_1 a h_2 jsou dvě hešovací funkce.

2-příhrádkové hešování: Délka nejdelšího řetězce (bez důkazu)

Očekávaná délka nejdelšího řetězce je $\mathcal{O}(\log \log n)$.

k-příhrádkové hešování

Prvek x může být uložen v příhrádkách $h_1(x), \dots, h_k(x)$ a nový prvek vkládáme do příhrádky s menším počtem prvků, kde h_1, \dots, h_k jsou hešovací funkce.

k-příhrádkové hešování: Délka nejdelšího řetězce (bez důkazu)

Očekávaná délka nejdelšího řetězce je $\mathcal{O}\left(\frac{\log \log n}{\log k}\right)$.

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry

10 Literatura

Lineární přidávání: Složitost operací Find, Insert a Delete

Předpoklady

- $m \geq (1 + \epsilon)n$
- Pokud příhrádky po smazaných prvcích jen značujeme, pak n je součet počtu prvků a označovaných příhrádek

Očekávaný počet porovnání při operaci Insert je

- $\mathcal{O}\left(\frac{1}{2\epsilon}\right)$ pro úplně náhodný systém (Knuth, 1963 [14])
- konstantní pro $\log(n)$ -nezávislý systém (Schmidt, Siegel, 1990 [22])
- $\mathcal{O}\left(\frac{1}{\epsilon \log \frac{1}{\epsilon}}\right)$ pro 5-nezávislý systém (Pagh, Pagh, Ruzic, 2007 [17])
- $\mathcal{O}(\log n)$ pro 4-nezávislý systém (Pátrašcu, Thorup, 2010 [20]) ①
- $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ pro tabulkové hešování (Pátrašcu, Thorup, 2012 [21])

Lineární přidávání: Analýza

Počet prvků od dané příhrádky do nejbližší volné příhrádky

Jestliže $\alpha < 1$ a systém hešovacích funkcí je úplně náhodný, pak očekávaný počet porovnání klíčů je $\mathcal{O}(1)$. ①

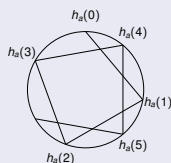
Důkaz

1. Necht $1 < c < \frac{1}{\alpha}$ a $q = \left(\frac{e^c - 1}{e^c}\right)^\alpha$
 - Platí $0 < q < 1$
2. Necht $p_t = P[|\{x \in S : h(x) \in T\}| = t]$ je pravděpodobnost, že do dané množiny příhrádek T velikosti t je zahešováno t prvků. Pak $p_t < q^t$. ②
 - Necht X_i je náhodná proměnná indikující, zda prvek i je zahesován do T
 - Necht $X = \sum_{i \in S} X_i$ a $\mu = E[X] = t\alpha$
 - Platí $c\mu = \alpha t < t$
 - Chernoff: $p_t = P[X = t] \leq P[X > c\mu] < \left(\frac{e^c - 1}{e^c}\right)^\mu = q^{\frac{\mu}{\alpha}} = q^t$
3. Necht b je nějaká příhrádka. Necht p'_k je pravděpodobnost, že příhrádky b až $b+k-1$ jsou obsazeny a $b+k$ je první volná příhrádka. Pak $p'_k < \frac{q^k}{1-q}$. ③
 - $p'_k < \sum_{s=0}^{\infty} p_{s+k} < q^k \sum_{s=0}^{\infty} q^s = \frac{q^k}{1-q}$
4. Očekávaný počet porovnání klíčů je $\sum_{k=0}^m k p'_k < \frac{1}{1-q} \sum_{k=0}^m k q^k = \frac{2-q}{(1-q)^2}$

Lineární přidávání: Proč 2-nezávislý systém nestačí?

Kombinace systému Multiply-shift a Lineárního přidávání

- Multiply-shift: $h_a(x) = (ax \bmod 2^w) \gg (w-l)$
- Označíme $h'_a(x) = \frac{ax \bmod 2^w}{2^{w-l}} = \frac{ax}{2^{w-l}} \bmod 2^l$
- Pak $h_a(x) = \lfloor h'_a(x) \rfloor$
- Jaká je očekávaná složitost vložení prvků $S = \{1, \dots, n\}$?



Vlastnosti

1. Označme $\|x\| = \min\{x, 2^l - x\}$, tj. vzdálenost od $h_a(0)$ po kružnici
2. Platí $\|h'_a(x) - h'_a(y)\| = \|h'_a(x-1) - h'_a(y-1)\| = \|h'_a(x-y)\|$ pro $x \geq y$
3. Platí $\|h'_a(ix)\| \leq i \|h'_a(x)\|$ pro všechna $i \in \{1, \dots, n\}$
4. Jestliže $\|h'_a(x)\| \geq 1$ pro všechna $x \in S$, pak h_a je perfektní na S
5. $P[h_a$ není perfektní] $\leq P[\exists x \in S : \|h'_a(x)\| < 1] \leq \sum_{x \in S} P[\|h'_a(x)\| < 1] \leq \frac{2n}{m}$
6. $P[h_a$ je perfektní na $S]$ $= 1 - P[h_a$ není perfektní na $S] \geq 1 - \frac{2n}{m}$

Cíl

Chtěli bychom ušetřit paměť, a tak prvky budeme ukládat přímo do tabulky.

Operace Insert

Nový prvek x vložíme do prázdné příhrádky $h(x) + i \bmod m$ s nejmenším možným $i \geq 0$.

Operace Find

Iterujeme dokud nenajdeme prvek nebo prázdnou příhrádku.

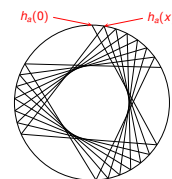
Operace Delete

- Lina varianta: Příhrádku smazaného prvku označujeme, aby následné operace Find pokračovali v hledání
- Varianta bez značkování: Zkontroluje a přesouvá prvky v celém řetězci

1. Existuje 4-nezávislý hešovací systém a posloupnost operací Insert nezávislá na vybrané hešovací funkci taková, že očekávaná složitost je $\Omega(\log n)$.

1. Neúspěšná operace Find musí dojít až k volné příhrádce a též operace Insert, pokud cestou není příhrádka označená operací Delete. Úspěšná operace Find může porovnat méně prvků. Složitost operace Delete se v různých verzích liší, ale v rozumných implementacích dojde nejhůře k nejbližší volné příhrádce. Knuth [14] spočítal očekávanou složitost přesně, ale výpočet je náročný.
2. Zde uvažujeme prvky, které hešovací funkce zobrazí do daných příhrádek, a nikoliv prvky, které se do daných příhrádek dostanou vlivem lineárního přidávání.
3. Tedy příhrádky $b-s$ až $b+k-1$ jsou obsazeny pro nějaké s . Indexy příhrádek počítáme modulo m .

Lineární přidávání: Proč 2-nezávislý systém nestačí?



Kombinace systému Multiply-shift a Lineárního přidávání

- Uvažujme prvek $x \in S$ takový, že $\|h'_a(x)\| \leq \frac{1}{2}$
- Prvky $x, 2x, \dots, kx$ patří do příhrádek $0, 1, \dots, \lfloor \frac{1}{2}k \rfloor$, kde $k = \lfloor \frac{m}{2} \rfloor$
- Máme nejvýše x skupin po alespoň k prvcích
- Složitost operace Insert je $\Omega\left(\frac{m}{x}\right)$, platí-li $\|h'_a(x)\| \leq \frac{1}{2}$

Lineární přidávání a hešovací systémy (Pátrašcu, Thorup, 2010 [20])

- Multiply-shift má očekávanou složitost $\Theta(\log n)$ operace Find
- Existuje 2-nezávislý systém s očekávanou složitostí $\Theta(\sqrt{n})$ operace Find

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
 - Universální hešování
 - Perfektní hešování
 - Separované řetězce
 - Lineární přidávání
 - Kukačkové hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry

10 Literatura

Kukačkové hešování: Algoritmus pro operaci Insert

Vložení prvku x do tabulky T

```

1 pos ←  $h_1(x)$ 
2 for  $n$  krát  $\textcircled{1}$  do
3   if  $T[pos]$  je prázdná then
4      $T[pos] \leftarrow x$ 
5     return
6   swap( $x, T[pos]$ )
7   if  $pos == h_1(x)$   $\textcircled{2}$  then
8      $pos \leftarrow h_2(x)$ 
9   else
10     $pos \leftarrow h_1(x)$ 
11 rehash()
12 insert(x)
```

Rehash

- Náhodně vygenerujeme nové hešovací funkce h_1 a h_2 z \mathcal{H}
- Můžeme zvětšit velikost tabulky
- Vložíme všechny prvky do nové tabulky $\textcircled{3}$

Kukačkové hešování: Analýza

Neorientovaný kukačkový graf G

- Vrcholy jsou příhrádky tabulky
- Hrany jsou dvojice $\{h_1(x), h_2(x)\}$ pro všechny prvky $x \in S$.

Vlastnosti kukaččího grafu

- Operace Insert postupuje po cestě z vrcholu $h_1(x)$ do prázdné pozice $\textcircled{1}$
- Nový prvek nemůže být vložen, jestliže komponenta obsahující $h_1(x)$ obsahuje kružnici

Lemma

Nechť $c > 1$ a $m \geq 2cn$. Pro dané příhrádky i a j je pravděpodobnost, že mezi i a j existuje cesta délky k , nejvýše $\frac{1}{mc^k}$.

Složitost operace Insert bez přehesování

Nechť $c > 1$ a $m \geq 2cn$. Očekávaná délka cesty je $\mathcal{O}(1)$.

Počet přehesování

Nechť $c > 2$ a $m \geq 2cn$. Očekávaný počet přehesování při vkládání n prvků do tabulky velikosti m je $\mathcal{O}(1)$. $\textcircled{2}$

- $k = 1$ For one element, the probability that it forms an edge ij is $\frac{2n}{m^2}$. So, the probability that there is an edge ij is at most $\frac{2n}{m^2} \leq \frac{1}{mc}$.
- $k > 1$ There exists a path between i and j of length k if there exists a path from i to u of length $k-1$ and an edge uj . For one position u , the i - u path exists with probability $\frac{1}{mc^{k-1}}$. The conditional probability that there exists the edge uj if there exists i - u path is at most $\frac{1}{mc}$ because some elements are used for the i - u path. By summing over all positions u , the probability that there exists i - j path is at most $m \frac{1}{mc^{k-1}} \frac{1}{mc} = \frac{1}{mc^k}$.

Insert without rehashing:

- Using the previous lemma for all length k and all end vertices j , the expected length of the path during operation Insert is $m \sum_{k=1}^{\infty} k \frac{1}{mc^k} \leq \sum_{k=1}^{\infty} \frac{k}{c^k} = \frac{c}{(c-1)^2}$.

Number of rehashes:

- Using the previous lemma for all length k and all vertices $i = j$, the probability that the graph contains a cycle is at most $m \sum_{k=1}^{\infty} \frac{1}{mc^k} = \frac{1}{c-1}$.
- The probability that inserting rehashes z times is at most $\frac{1}{(c-1)^z}$.
- The expected number of rehashes is at most $\sum_{z=0}^{\infty} z \frac{1}{(c-1)^z} = \frac{c-1}{(c-2)^2}$.

Kukačkové hešování (Pagh, Rodler [18])

Popis

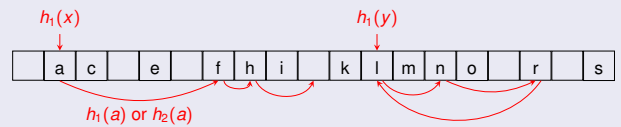
Pro dvě hešovací funkce h_1 a h_2 prvek x musí být uložen v příhrádce $h_1(x)$ nebo $h_2(x)$. V jedné příhrádce může být uložen nejvýše jeden prvek.

Operace Find a Delete

Triviální, složitost $\mathcal{O}(1)$ v nejhorším případě.

Příklad operace Insert

- Úspěšně vložení prvku x do příhrádky $h_1(x)$ po třech realokacích
- Prvek y není možné vložit do $h_1(y)$



- 1 Po n pokusech jsme už určitě v cyklu. Lze ukázat, že v cyklu jsme s velkou pravděpodobností už po $\Omega(\log n)$ krocích.
- 2 Potřebuje najít druhou pozici, ve které prvek x může být uložen.
- 3 Při vkládání prvků do nové tabulky může dojít k Rehash, takže si při implementaci musíme dát pozor, abychom některé prvky neztratili.

- 1 Z příhrádek komponenty tvořící cestu je právě jedna volná, ale nemusí to být příhrádka koncového vrcholu cesty.
- 2 Předpokládáme, že na začátku je tabulka prázdná a chceme vytvořit tabulku velikosti m s n prvky.

Kukačkové hešování: Analýza

Složitost operace Insert bez přehesování

Nechť $c > 1$ a $m \geq 2cn$. Očekávaná délka cesty je $\mathcal{O}(1)$.

Počet přehesování

Nechť $c > 2$ a $m \geq 2cn$. Očekávaný počet přehesování při vkládání n prvků do tabulky velikosti m je $\mathcal{O}(1)$. $\textcircled{1}$

Složitost operace Insert včetně přehesování

Nechť $c > 2$ a $m \geq 2cn$ a hešovací systém je úplně nezávislý. Pak očekávaná amortizovaná složitost operace Insert je $\mathcal{O}(1)$.

Pagh, Rodler [18]

Jestliže $c > 1$ a $m \geq 2cn$ a hešovací systém je $\log n$ -nezávislý, pak očekávaná amortizovaná složitost operace Insert je $\mathcal{O}(1)$.

Pátračcu, Thorup [21]

Jestliže $c > 1$ a $m \geq 2cn$ a použijeme tabulkové hešování, pak časová složitost vytvoření statické Kukačkové tabulky je $\mathcal{O}(n)$ s velkou pravděpodobností.

- Předpokládáme, že na začátku je tabulka prázdná a chceme vytvořit tabulku velikosti m s n prvky.

Cvičení:

- 7.1. Hodíme n míčků do n košů a označíme n_i počet míčků v i -té přihrádce. Dokažte, že $E[\sum_{i=1}^n n_i^2] = \Theta(n)$.
- 7.2. Dokažte, že čekáme-li na událost, která nastane v jednom kroku s pravděpodobností p (nezávisle na ostatních krocích), pak $E[\# \text{ kroků}] = \frac{1}{p}$.
- 7.3. Dokažte, že pomocí c -universálního hešovacího systému je možné efektivně zkonstruovat (jednouúrovňové) perfektní hešování do tabulky velikosti $m = \Theta(n^2)$.
- 7.4. Dokažte, že k sestrojení dvouúrovňového perfektního hešování FKS stačí 2-nezávislý hešovací systém.
- 7.5. Dokažte, že systém $\mathcal{H} = \{h_a(x) = (ax \bmod p) \bmod m; 0 < a < p\}$ je c -universální a najděte nejmenší možné c .
- 7.6. Formálně dokažte, že systém všech náhodných funkcí je 1-universální a k -nezávislý pro všechna k .
- 7.7. Formálně dokažte, že k -nezávislý systém je $(k-1)$ -nezávislý.
- 7.8. Formálně dokažte, že 2-nezávislý systém je c -universální pro nějaké c .
- 7.9. Najděte 1-universální systém, který není 2-nezávislý.
- 7.10. Dokažte, že když $P[h(x_i) = z_i]$ pro všechna $i = 1, \dots, k] \leq \frac{1}{m^k}$ pro všechna $z_1, \dots, z_k \in M$, pak $P[h(x_i) = z_i]$ pro všechna $i = 1, \dots, k] = \frac{1}{m^k}$ pro všechna $z_1, \dots, z_k \in M$.
- 7.11. Proč je hešovací systém Multiply-mod-prime počítá dvě modula? Předpokládejme, že velikost tabulky m je prvočíslo, a uvažujme hešovací systém funkcí $h_{a,b}(x) = ax + b \bmod m$ pro $a, b \in M$. Je tento systém c -universální pro nějaké c nezávislé na m ?

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
 - k-d stromy
 - Intervalové stromy
 - Interval trees
 - R-stromy
- 8 Dynamizace
- 9 Bloom Filtry

Intervalový dotaz v \mathbb{R}^1

Statically

Body uložíme do pole

Build: $\mathcal{O}(n \log n)$

Count: $\mathcal{O}(\log n)$

Query: $\mathcal{O}(k + \log n)$

k je počet vyjmenovaných bodů

Dynamicky

Body uložíme do vyhledávacího stromu

Build: $\mathcal{O}(n \log n)$

Insert: $\mathcal{O}(\log n)$

Delete: $\mathcal{O}(\log n)$

Count: $\mathcal{O}(\log n)$

Query: $\mathcal{O}(k + \log n)$

Hešování: Další možnosti

Kvadratické prohledávání

Vložit prvek x do prázdné přihrádky $h(x) + ai + bi^2 \bmod m$ s nejmenším možným $i \geq 0$, kde a, b jsou pevné konstanty.

Dvojitě hešování

Vložit prvek x do prázdné přihrádky $h_1(x) + ih_2(x) \bmod m$ s nejmenším možným $i \geq 0$, kde h_1, h_2 jsou dvě hešovací funkce.

Brentova varianta operace Insert

Jestliže přihrádka

- $b = h_1(x) + ih_2(x) \bmod m$ je obsazena prvkem y
- $b + h_2(x) \bmod m$ je taky obsazena
- $c = b + h_2(y) \bmod m$ je prázdná,

pak přesuneme prvek y do přihrádky c a prvek x vložíme do b . Tímto se zkrátí očekávaná doba hledání.

- 7.12. Dokažte, že tabulkové hešování je 3-nezávislé.
- 7.13. Rozhodněte, zda systém $\mathcal{H} = \{h_a(x) = (x + a \bmod p) \bmod m; 0 \leq a < p\}$ je c -universální pro nějaké c .
- 7.14. Najděte nejmenší c , pro které je systém $\mathcal{H} = \{h_a(x) = (ax + b \bmod p) \bmod m; a, b \in [p]\}$ c -universální.
- 7.15. Rozhodněte, zda systém $\mathcal{H} = \{h_a(x) = (ax + b \bmod p) \bmod m; a, b \in [p], a \neq 0\}$ je 2-nezávislý.
- 7.16. Dokažte, že systém Poly-mod-prime $\mathcal{H} = \{h_{a_0, \dots, a_{k-1}}(x) = (\sum_{i=0}^{k-1} a_i x^i \bmod p) \bmod m; a_0, \dots, a_{k-1} \in [p]\}$ je k -nezávislý.

Intervalový dotaz (Range query)

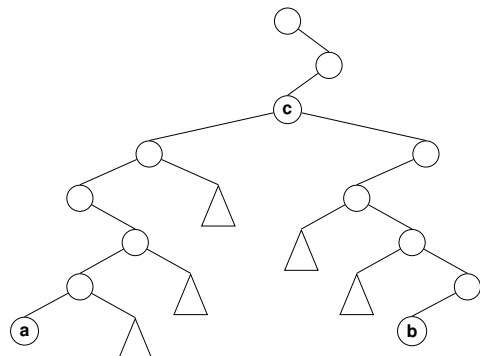
Popis problému

- Daná množina n bodů S v \mathbb{R}^d
- Intervalem rozumíme d -dimenzionální obdélník
- Operace Query: Najít všechny body v daném intervalu
- Operace Count: Určit počet bodů v daném intervalu

Aplikace

- Počítačová grafika, výpočetní geometrie
- Databázové dotazy, např. určit zaměstnance ve věku 20-35 a platem 20-30 tisíc

Intervalový dotaz v \mathbb{R}^1 : Příklad



Vrchol a je nejmenší prvek v intervalu, b je největší prvek v intervalu a c je poslední společný vrchol na cestách z kořene do vrcholů a a b .

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
 - k-d stromy
 - Intervalové stromy
 - Interval trees
 - R-stromy
- 8 Dynamizace
- 9 Bloom Filtry

k-d stromy: Operace Query, Count: Algoritmus

Algoritmus

```

1 Procedure Query (vrchol stromu v, interval R)
2   if v je list then
3     Vypiš v, pokud leží v R
4   else if rozdělující nadrovina vrcholu v protíná R then
5     Query (levý syn v, R)
6     Query (pravý syn v, R)
7   else if R je „vlevo“ od rozdělující nadroviny vrcholu v then
8     Query (levý syn v, R)
9   else
10    Query (pravý syn v, R)

```

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
 - k-d stromy
 - Intervalové stromy
 - Interval trees
 - R-stromy
- 8 Dynamizace
- 9 Bloom Filtry

Intervalové stromy v \mathbb{R}^2 : Paměť

Vertikální pohled

Každý bod p je uložen v právě jednom vrcholu v x-stromu a dále je bod p uložen v každém y-stromu přiřazenému vrcholu na cestě z x-kořene do v .

Horizontální pohled

Každá vrstva x-stromu rozkládá body podle x-ové souřadnice. Proto každý bod je uložen v nejvýše jednom y-stromu z každé vrstvy x-stromu.

Paměťová složitost

Každý bod je uložen v $\mathcal{O}(\log n)$ y-stromech a celková paměťová složitost je $\Theta(n \log n)$.

Popis

- Body uložíme do binárního stromu
- Do kořene uložíme medián podle první souřadnice
- Do levého (pravého) podstromu uložíme body mající první souřadnice menší (větší) než medián
- Vrcholy v první vrstvě pod kořenem se body rozdělují podle druhé souřadnice
- V dalších vrstvách dělíme (cyklicky) podle dalších souřadnic
- Výška stromu je $\log_2 n + \Theta(1)$
- Operace Build v čase $\mathcal{O}(n \log n)$
- Body je též možné ukládat jen do listů a vrcholy pak obsahují jen rozdělující nadroviny

k-d stromy: Operace Query, Count: Analýza

Příklad nejhoršího případu pro \mathbb{R}^2

- Máme množinu bodů $S = \{(x, y); x, y \in [m]\}$, kde $n = m^2$
- Chceme najít množinu všech bodů v intervalu $(1, 2; 1, 8) \times \mathbb{R}$
- V každé vrstvě rozdělující podle y-ové souřadnice musíme prozkoumat oba podstromy
- Výška stromu je $\log_2 n + \Theta(1)$ a v polovině vrstev prozkoumáváme oba podstromy
- Celkem navštívíme $2^{\frac{1}{2} \log_2 n + \Theta(1)} = \Theta(\sqrt{n})$ listů

Příklad nejhoršího případu pro \mathbb{R}^d

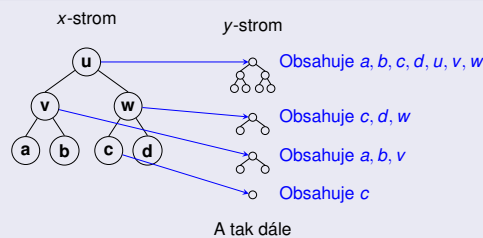
- Mějme množinu bodů $S = [m]^d$, kde $n = m^d$
- Chceme najít množinu všech bodů v intervalu $(1, 2; 1, 8) \times \mathbb{R}^{d-1}$
- V každé vrstvě nerozdělující podle první souřadnice musíme prozkoumat oba podstromy
- V $\frac{d-1}{d} \log_2 n + \Theta(1)$ vrstvách prozkoumáváme oba podstromy
- Celkem navštívíme $2^{\frac{d-1}{d} \log_2 n + \Theta(1)} = \Theta(n^{1-\frac{1}{d}})$ listů

Intervalové stromy v \mathbb{R}^2 (Range tree)

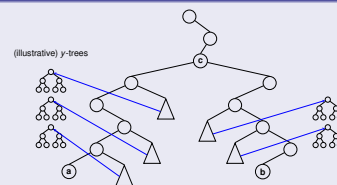
Konstrukce

- Vybudujeme binární vyhledávací strom podle x-ové souřadnice bodů (x-strom)
- Pak pro každý vrchol v vybudujeme jeden binární vyhledávací strom podle y-ové souřadnice obsahující všechny body v podstromu vrcholu v (y-strom)
- Bodu můžou být uloženy ve všech vrcholech nebo jen v listech

Příklad

Intervalové stromy v \mathbb{R}^2 : Operace Query

Příklad



Složitost

$\mathcal{O}(k + \log^2 n)$ protože y-ový dotaz je volán v $\mathcal{O}(\log n)$ y-stromech

Popis

- i -strom je binární vyhledávací strom podle i -té souřadnice pro $i = 1, \dots, n$
- Pro $i < d$ má každý vrchol u i -stromu ukazatel na $(i + 1)$ -strom obsahující právě všechny body z podstromu vrcholu u
- Intervalovým stromem rozumíme všechny výše popsané stromy

Representace

Struktura vrcholu intervalového stromu obsahuje

- key** nadrovina rozdělující prostor mezi syny
- left, right** ukazatel na levého a pravého syna
- tree** ukazatel na kořen $(i + 1)$ -stromu

Intervalové stromy v \mathbb{R}^d : BuildAlgoritmus (Body M jsou v poli setříděné podle poslední souřadnice)

```

1 Procedure Build (množina bodů  $M$ , dimenze stromu  $d$ , aktuální souřadnice  $i$ )
2   if  $|M| = 1$  then
3     return nový list obsahující jediný vrchol  $M$ 
4   if  $i = d$  then
5     return kořen stromu vytvořený ze setříděného pole
6    $v \leftarrow$  nový vrchol
7    $v.tree \leftarrow$  Build ( $M$ ,  $d$ ,  $i + 1$ )
8    $v.key \leftarrow$  medián  $i$ -tých souřadnic bodů  $M$ 
9    $M_l, M_r \leftarrow$  rozděl  $M$  na body mající  $i$ -tou souřadnici menší a větší než  $v.key$ 
10   $v.left \leftarrow$  Build ( $M_l$ ,  $d$ ,  $i$ )
11   $v.right \leftarrow$  Build ( $M_r$ ,  $d$ ,  $i$ )
12  return  $v$ 

```

Intervalové stromy v \mathbb{R}^d : Query $\langle a_1, b_1 \rangle \times \dots \times \langle a_d, b_d \rangle$

```

1 Procedure Query (vrchol  $v$ , aktuální souřadnice  $i$ )
2   if  $v = NIL$  then
3     return
4   if  $v.key \leq a_i$  then
5     Query ( $v.right$ ,  $i$ )
6   else if  $v.key \geq b_i$  then
7     Query ( $v.left$ ,  $i$ )
8   else
9     Query_left ( $v.left$ ,  $i$ )
10    Query_right ( $v.right$ ,  $i$ )
11 Procedure Query_left (vrchol  $v$ , aktuální souřadnice  $i$ )
12  if  $v = NIL$  then
13    return
14  if  $v.key < a_i$  then
15    Query_left ( $v.right$ ,  $i$ )
16  else
17    Query_left ( $v.left$ ,  $i$ )
18    if  $i < d$  then
19      Query ( $v.right.tree$ ,  $i + 1$ )
20    else
21      Vypiš všechny body v podstromu vrcholu  $v.right$ 

```

Intervalové stromy v \mathbb{R}^d : Dynamizace pomocí $BB[\alpha]$ -stromů $BB[\alpha]$ -strom

- Binární vyhledávací strom
- Počet listů v podstromu vrcholu u označme s_u
- Podstromy obou synů každého vrcholu u mají alespoň αs_u listů

Operace Insert (Delete je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací Build (složitost: amortizovaná analýza)

Amortizovaná časová složitost operací Insert a Delete: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace Build má s_u listů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$
- Při jedné operaci Insert/Delete se prvek přidá/smaže v $\Theta(\log n)$ podstromech
- Amortizovaný čas vyvažování při jedné operaci Insert nebo Delete je $\mathcal{O}(\log n)$

Asymptotické velikosti stromů a počty vrcholů, jsou-li stromy jsou vyvážené

	1-stromy	2-stromy	3-stromy	$(d - 1)$ -stromy	d -stromy
Počet stromů obsahující daný bod	1	$\log n$	$\log^2 n$	$\log^{d-2} n$	$\log^{d-1} n$
Celkový počet vrcholů	n	$n \log n$	$n \log^2 n$	$n \log^{d-2} n$	$n \log^{d-1} n$
Celkový počet stromů	1	n	$n \log n$	$n \log^{d-3} n$	$n \log^{d-2} n$

Paměť

- Indukcí: každý bod je uložený v $\mathcal{O}(\log^{i-1} n)$ i -stromech
- Celková paměťová složitost je $\mathcal{O}(n \log^{d-1} n)$

Intervalové stromy v \mathbb{R}^d : Analýza operace BuildVytvoření d -stromů

- d -stromy vytváříme v lineárním čase (tj. konstantní čas na vrchol)
- Počet vrcholů ve všech d -stromech je $\Theta(n \log^{d-1} n)$
- Časová složitost vytvoření všech d -stromů je $\mathcal{O}(n \log^{d-1} n)$

Vytvoření i -stromu pro $i = 1, \dots, d - 1$ (nepočítaje $(i + 1)$ -stromy, ...)

- Počet vrcholů ve všech i -stromech je $\Theta(n \log^{i-1} n)$
- Nechť n_T je počet vrcholů v i -stromu T
- Výbudování samotného stromu T trvá $\mathcal{O}(n_T \log n_T)$
- Výbudování všech i -stromů trvá

$$\sum_{i\text{-strom } T} n_T \log n_T \leq \log n \sum_{i\text{-strom } T} n_T = \log n \cdot n \log^{i-1} n = n \log^i n$$

Časová složitost operace Build

$$\mathcal{O}(n \log^{d-1} n)$$

Intervalové stromy v \mathbb{R}^d : Query

Složitost

- V každém stromě přistoupíme k nejvýše dvěma vrcholům z každé vrstvy
- Z každého navštíveného i -stromu pokračujeme do $\mathcal{O}(\log n)$ $(i + 1)$ -stromů
- Počet navštívených i -stromů je $\mathcal{O}(\log^{i-1} n)$
- Vypsání všech bodů v podstromu trvá $\mathcal{O}(k)$, kde k je počet nalezených bodů
- Celková složitost je $\mathcal{O}(k + \log^d n)$

Intervalové stromy v \mathbb{R}^d : Dynamizace pomocí $BB[\alpha]$ -stromůPoužití $BB[\alpha]$ -stromů v intervalových stromech

- Binární vyhledávací stromy implementujeme pomocí $BB[\alpha]$ -stromů
- Vyžaduje-li $BB[\alpha]$ -strom vyvážení, pak přebudujeme všechny přiřazené stromy

Složitost operace Insert a Delete

- Navštívených i -stromů je $\mathcal{O}(\log^{i-1} n)$ a v každém navštívíme $\mathcal{O}(\log n)$ vrcholů
- Složitost bez přebudování je $\mathcal{O}(\log^d n)$; analyzujeme přebudování
- Uvažujme libovolný vrchol u , který leží v i -stromu
- Přebudování vrcholu u trvá $\mathcal{O}(s_u \log^{d-i} s_u)$
- Přebudování vrcholu u může nastat po $\Omega(s_u)$ po přidání/smazání do vrcholu u
- Amortizovaná cena přidání/smazání do vrcholu u je $\mathcal{O}(\log^{d-i} s_u) \leq \mathcal{O}(\log^{d-i} n)$
- Amortizovaný čas operace Insert a Delete je $\sum_{i=1}^d \mathcal{O}(\log^{i-1} n) \mathcal{O}(\log n) \mathcal{O}(\log^{d-i} n) = \mathcal{O}(\log^d n)$

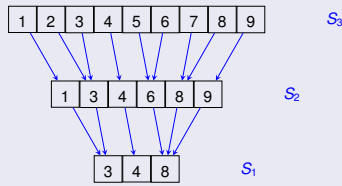
Kaskádování (Fractional cascading)

Motivační problém

Dány množiny $S_1 \subseteq \dots \subseteq S_m$, kde $|S_m| = n$, vymyslete datovou strukturu pro rychlé vyhledání prvku $x \in S_1$ ve všech množinách S_1, \dots, S_m . ☹

Kaskádování

Všechny množiny jsou seříděné a navíc každý prvek v poli S_i má ukazatel na stejný prvek v poli S_{i-1} . ☺



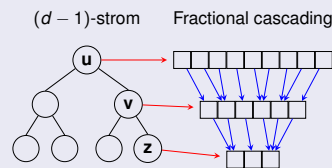
Složitost hledání ve všech m množinách

$\mathcal{O}(m + \log n)$

Intervalové stromy v \mathbb{R}^d s kaskádováním v poslední dimenzi

Použití

- Každému $(d-1)$ -stromu je přiřazena je kaskáda místo d -stromů
- Každý prvek v kaskádě musí mít dva ukazatele do pole nižší úrovně (pro levého a pravého syna vrcholu v $(d-1)$ -stromu)



Složitost operace Query

- Dotaz v jednom $(d-1)$ -stromu trvá $\mathcal{O}(\log n)$ včetně vyhodnocení kaskády
- Dotazů v $(d-1)$ -stromech je $\mathcal{O}(\log^{d-2} n)$
- Složitost operace Query je $\mathcal{O}(k + \log^{d-1} n)$

Další známá vylepšení intervalových stromů

Popsaný postup

Query: $\mathcal{O}(k + \log^{d-1} n)$

Paměť: $\mathcal{O}(n \log^{d-1} n)$

Chazelle [3, 4]

Query: $\mathcal{O}(k + \log^{d-1} n)$

Paměť: $\mathcal{O}\left(n \left(\frac{\log n}{\log \log n}\right)^{d-1}\right)$

Chazelle, Guibas [5] pro $d \geq 3$

Query: $\mathcal{O}(k + \log^{d-2} n)$

Paměť: $\mathcal{O}(n \log^d n)$

Interval trees

Vstup

Množina stromů $S = \{I_1, \dots, I_n\}$, kde $I_i = \langle a_i, b_i \rangle$.

Rekurzivní konstrukce binárního stromu

Nechť

- m je medián koncových bodů $a_1, b_1, \dots, a_n, b_n$,
- $S_m = \{I_i; a_i \leq m \leq b_i\}$ jsou intervaly obsahující m ,
- $S_l = \{I_i; b_i < m\}$ intervaly menší než m and
- $S_r = \{I_i; m < a_i\}$ intervaly větší než m .

Kořen stromu obsahuje

- dvě pole intervalů S_m seříděných podle levého a pravého konce intervalů,
- interval tree intervalů S_l v levém podstromu a
- interval tree intervalů S_r v pravém podstromu. ☹

Složitost

- Doba konstrukce $\mathcal{O}(n \log n)$. ☺
- Paměť $\mathcal{O}(n)$. ☺

- 1 Triviální řešení získá složitost $\mathcal{O}(m \log n)$, kterou bychom chtěli zlepšit.
- 2 Prvky $S_i \setminus S_{i-1}$ ukazují na své předchůdce nebo následovníky.

Intervalové stromy v \mathbb{R}^d s kaskádováním v poslední dimenzi

Paměťová složitost

- Místo d -stromu T s s_T vrcholy máme pole velikosti s_T
- Paměťová složitost je $\mathcal{O}(n \log^{d-1} n)$

Složitost operace Build

- Vybudování $(d-1)$ -stromu s s_u vrcholy včetně kaskádování trvá $\mathcal{O}(s_u \log s_u)$
- Složitost vybudování i -stromů pro $i < d$ se nemění
- Složitost operace Build je $\mathcal{O}(n \log^{d-1} n)$

Operace Insert a Delete (Cvičení)

- Je možné efektivně přidávat a mazat body?
- Je možné reprezentovat kaskády tak, aby bylo možné efektivně hledat, přidávat i mazat body?

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
 - k-d stromy
 - Intervalové stromy
 - Interval trees
 - R-stromy
- 8 Dynamizace
- 9 Bloom Filtry

Literatura

- 1 If S_l or S_r is empty, then there is no left or right child, respectively.
- 2 There are at most n end-points smaller than m , so S_l contains at most $\frac{n}{2}$ intervals. Therefore, the time complexity satisfies the recurrence formula $T(n) \leq 2T(\frac{n}{2}) + \Theta(n)$.
- 3 Every interval is stored in exactly one node. If S_m is empty, then n is even and both S_l and S_r contains $\frac{n}{2}$ intervals. There are at most $n-1$ such nodes. Therefore, the tree has at most $2n-1$ nodes.

Popis problému

Pro daný interval $Q = (a_q, b_q)$ mají všechny intervaly mající průnik s intervalem Q .

Rekurzivní algoritmus

```

1 if  $a_q \leq m \leq b_q$  then
2   Vypíšeme všechny intervaly  $S_m$ 
3   Rekurzivně zpracujeme oba podstromy
4 else if  $b_q < m$  then
5   Pomocí pole intervalů  $S_m$  seříděných podle levého konce najdeme všechny
   intervaly mající průnik s  $Q$ 
6   Rekurzivně zpracujeme levý podstrom
7 else
8   Pomocí pole intervalů  $S_m$  seříděných podle pravého konce najdeme všechny
   intervaly mající průnik s  $Q$ 
9   Rekurzivně zpracujeme pravý podstrom

```

Složitost

$\mathcal{O}(k + \log n)$

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
 - k-d stromy
 - Intervalové stromy
 - Interval trees
 - R-stromy
- 8 Dynamizace
- 9 Bloom Filtry

R-strom: Hledání

Bodový a intervalový dotaz

- Hledáme objekty ležící v daném bodě nebo mající neprázdný průnik s daným objektem (obdélníkem) O .
- Rekurzivně z vrcholu pokračujeme do synů, jejichž ohraničující obdélník má neprázdný průnik s O .

Hledání nejbližšího objektu k danému bodu

- Vzdálenost bodu p a objektu $O \subseteq \mathbb{R}^n$ je $\text{dist}(p, O) = \min_{x \in O} \|p - x\|$.
- Jestliže existuje objekt O^* a pro ohraničující obdélník O vrcholu u R-stromu platí $\text{dist}(p, O^*) < \text{dist}(p, O)$, pak v podstromu vrcholu u neexistuje objekt bližší k bodu p než je O^* .

R-strom: Hledání nejbližšího objektu

Algoritmus založený na prioritní frontě vrcholů, které zbývá projít

```

1 Necht  $O^*$  je dosud nejbližší nalezený objekt (na začátku libovolný)
2 Prioritní fronta  $F \leftarrow$  kořen
3 while  $F$  je neprázdná do
4    $u \leftarrow \text{Pop}(F)$ 
5   if  $u$  je list then
6     if  $\text{dist}(p, O^*) > \text{dist}(p, O)$ , kde  $O$  je objekt vrcholu  $u$  then
7        $O^* := O$ 
8       Z fronty  $F$  smažeme všechny vrcholu, pro jejichž obdélník  $O$  platí
        $\text{dist}(p, O^*) \leq \text{dist}(p, O)$ 
9     else if  $\text{dist}(p, O^*) > \text{dist}(p, O)$ , kde  $O$  je obdélník vrcholu  $u$  then
10      for  $v$  syn  $u$  do
11        Push( $F, v$ )

```

Cvičení:

- 8.1. Dokažte, že paměťová složitost intervalového stromu je $\Omega(n \log^{d-1})$.
- 8.2. Zkuste dokázat časovou složitost vytvoření intervalového stromu pomocí Master theorem
- 8.3. Jak určit počet bodů v daném intervalu?
- 8.4. Analyzujte složitost operací Insert a Delete pomocí potenciálové metody.
- 8.5. Jak dlouho trvá postavit intervalový strom s kaskádováním?
- 8.6. Je možné zkombinovat dynamizaci intervalových stromů pomocí BB[α]-stromů s kaskádováním?

R-strom

Základní pojmy

- Obdélníkem v R^n rozumíme n -rozměrný interval, tj. n -rozměrný kvádr, jehož každá hrana je paralelní s některou osou.
- Ohraničujícím obdélníkem množiny $M \subseteq R^n$ rozumíme nejmenší obdélník obsahující S .

Cíl

- Pro grafický software (vektorový editor, CAD, GIS, ...) vytvořit datovou strukturu, která bude umět efektivně vyhledávat objekty splňující různé podmínky podle požadavků uživatele.
- Pro zjednodušení uvažujeme jen ohraničující obdélníky jednotlivých objektů.

R-strom

- Každý vnitřní vrchol (kromě kořene) R-stromu má a až b synů.
- Všechny listy jsou ve stejné hladině.
- Každý objekt (resp. jeho ohraničující obdélník) je uložen jednom listu.
- Každý vnitřní vrchol si pamatuje ohraničující obdélník obsahující všechny objekty ve svém podstromu.

R-strom: Hledání nejbližšího objektu

Algoritmus založený na průchodu do hloubky

```

1 Necht  $O^*$  je dosud nejbližší nalezený objekt
2 Procedure NearestNeighbour (vrchol  $u$ , bod  $p$ )
3   if  $u$  je list then
4     Zkontroluje, jestli  $u$  obsahuje objekt bližší k  $p$  než  $O^*$ 
5   else
6     Vytvoříme seznam  $L$  všech synů vrcholu  $u$ 
7     Seznam  $L$  seřídíme vhodných třídících kritérií, např.  $\text{dist}(p, O)$ 
8     for  $v \in L$  do
9       if Obdélník  $O$  vrcholu  $v$  splňuje podmínky  $\text{dist}(p, O^*) \geq \text{dist}(p, O)$  then
10        NearestNeighbour( $v, p$ )

```

R-strom: Hledání průsečíků

Varianty

- Nalezení všech dvojic objektů s neprázdným průnikem
- Spatial join: Pro dané dvě množiny objektů R a S najít všechny dvojice objektů $r \in R$ a $s \in S$ s neprázdným průnikem

Algoritmus

```

1 Procedure Intersections (vrchol  $u$ , vrchol  $v$ )
2   for  $u'$  syn  $u \oplus$  do
3     for  $v'$  syn  $v$  do
4       if Obdélníky vrcholů  $u'$  a  $v'$  mají neprázdný průnik then
5         if  $u'$  a  $v'$  jsou listy then
6           Vypíšeme objekty ve vrcholech  $u'$  a  $v'$ 
7         else
8           Intersections( $u', v'$ )

```

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 **Dynamizace**
- 9 Bloom Filtry
- 10 Literatura

Deamortizace dynamického pole

Cíl

Vkládání prvku do dynamického pole umíme v amortizovaně v konstantním čase. Jak vložit prvek do dynamického pole tak, aby složitost byla konstantní i v nejhorším případě?

Postup

- Na začátku máme n prvků v hlavním poli velikosti $2n$ a alokujeme si prázdné pomocné pole velikosti $4n$
- Při každé operaci Insert zkopírujeme dva prvky do pomocného pole
- Po n operacích Insert jsou všechny prvky zkopírované do pomocného pole, a proto můžeme hlavní pole zrušit, pomocné nazvat hlavním a alokovat nové pomocné pole velikosti $8n$

Složitost

Paměťová složitost $\mathcal{O}(n)$ a časová $\mathcal{O}(1)$ v nejhorším případě. ①

Dynamizace přebudováním

Jednou za čas přebudujeme celou datovou strukturu

- Aplikace pro datové struktury omezené kapacity (pole, halda, hešovací tabulky)
- Vyčištění datové struktury od smazaných prvků
- Změna vnitřních parametrů

Složitost

- Nechtě složitost přebudování je $T(n)$ a přebudováváme po $\Theta(n)$ operacích
- Přebudování stojí $\mathcal{O}\left(\frac{T(n)}{n}\right)$ amortizovaně na operaci
- Předpokládáme $T(\mathcal{O}(n)) = \mathcal{O}(T(n))$

Částečné přebudování

Například BB[α]-stromy

R-strom: Insert

Idea

Nové objekty vkládáme podobně jako do (a,b)-stromu, ale

- v každém vrcholu musíme rozhodnout, do kterého syna pokračovat, a
- musíme rozhodnout, jak rozdělit syny při štěpení.

Tyto volby neovlivňují korektnost struktury, ale efektivitu vyhledávání.

Volba syna

Zvolíme takového syna s obdélníkem O , že po vložení nového objektu musíme O nejméně zvětšit.

Štěpení

Rozdělit vrchol u na vrcholu u_1 a u_2 tak, aby součet velikostí obdélníků v u_1 a u_2 byl nejmenší.

- Vyzkoušet všechny kombinace (může být neefektivní)
- Kvadratické štěpení
- Lineární štěpení

Základní pojmy

Základní pojmy

Semi-dynamizace: Naučit datovou strukturu vkládání nových prvků

Úplná dynamizace: Naučit datovou strukturu i mazání prvků

Persistence: Zapamatování si změn v datové struktuře a vyhledávání podle stavu v daném čase

Deamortizace: Modifikace operace mající amortizovanou složitost tak, že upravená operace garantuje složitost v nejhorším případě

- Předpokládáme, že alokace a dealokace bloku paměti trvá $\mathcal{O}(1)$.

Obecná semi-dynamizace

Vyhledávací problém

Vyhledávací problém je zobrazení $f : U_Q \times 2^{U_X} \rightarrow U_R$, kde

- U_Q je universum dotazů
- U_X je universum prvků (uvažujeme jen konečné podmnožiny prvků)
- U_R je universum výsledků

Rozložitelnost

Vyhledávací problém je rozložitelný, jestliže existuje zobrazení $\sqcup : U_R \times U_R \rightarrow U_R$ vyčíslitelná v konstantním čase taková, že pro všechna disjunktní $A, B \subseteq U_X$ a všechny dotazy $q \in U_Q$ platí $f(q, A \cup B) = f(q, A) \sqcup f(q, B)$.

Příklady

- Hledání nejbližšího bodu v \mathbb{R}^d je rozložitelné a máme $U_Q = U_X = U_R = \mathbb{R}^d$
- Rozhodnutí zda bod leží v konvexním obalu bodů je nerozložitelné pro $d \geq 2$

Uvažujeme statickou datovou strukturu se složitostmi

- $B_S(n)$ - čas na build
- $Q_S(n)$ - čas na dotaz
- $S_S(n)$ - paměť
- Předpokládáme, že $Q_S(n)$, $\frac{B_S(n)}{n}$ a $\frac{S_S(n)}{n}$ jsou neklesající

Konstrukce

- Prvky rozdělíme do bloků B_0, B_1, \dots takových, že B_i obsahuje 0 nebo 2^i prvků ①
- Pro každý neprázdný blok vytvoříme jednu statickou datovou strukturu ②
- Časová složitost $\sum_{i: B_i \neq \emptyset} B_S(2^i) \leq B_S(n)$ ③
- Prostorová složitost $\sum_{i: B_i \neq \emptyset} S_S(2^i) \leq S_S(n)$

Dotaz

- Dotaz položíme ve všech blocích a výsledky zkombinujeme funkcí \sqcup
- Složitost $\log n + \sum_{i: B_i \neq \emptyset} Q_S(2^i) \leq \mathcal{O}(Q_S(n) \log n)$ ④

Obecná semi-dynamizace

Insert

- Vytvoříme nový blok B_0 s datovou strukturou obsahující jen nový prvek
- Jestliže máme dvakrát blok B_{i-1} , pak je zrušíme a vytvoříme B_i v čase $B_S(2^i)$ ①
- Analýza: B_i vytvoříme jednou za 2^i operací Insert
- Amortizovaná cena vytvoření B_i je $\frac{B_S(2^i)}{2^i} \leq \frac{B_S(n)}{n}$
- Musíme předplácet vytváření všech $B_i \Rightarrow$ amortizovaná složitost je $\mathcal{O}\left(\frac{B_S(n)}{n} \log n\right)$

Deamortizovaná semi-dynamizace

- Rebuild rozložíme mezi více operací a pokaždé provedeme kousek
- Každý blok máme nejvýše třikrát a navíc jeden rozpracovaný
- Jestliže blok B_{i-1} máme dvakrát, tak začneme vytvářet B_i , který vytvoříme po 2^{i-1} krocích
- Při každé operaci Insert provedeme $\frac{B_S(2^i)}{2^{i-1}} \leq 2 \frac{B_S(n)}{n}$ instrukcí na vytvoření B_i pro každé $i \leq \log n$
- Složitost operace Insert je $\mathcal{O}\left(\frac{B_S(n)}{n} \log n\right)$ v nejhorším případě

Úplná dynamizace

Předpoklad

Datová struktura musí umět efektivně prvky hledat a označovat smazané prvky (v čase $M_S(n)$). ①

Jednodušší varianta operace Delete bez binární verze pro operaci Insert

- Pokud máme v datové struktuře víc označených prvků než neoznačených, pak provedeme Rebuild. ②
- Rebuild nastane nejdříve po $\frac{n}{2}$ operacích Delete a trvá $B_S(n)$
- Amortizovaná složitost je $\mathcal{O}\left(M_S(n) + \frac{B_S(n)}{n}\right)$

Kombinace operací Insert a Delete

- Jestliže v bloku B_i je většina prvků označena, pak z B_i vytvoříme B_{i-1} ③
- Každá operace Delete musí předplatit $\frac{B_S(n)}{n}$
- Amortizovaná složitost operace Delete je $\mathcal{O}\left(\log n + M_S(n) + \frac{B_S(n)}{n}\right)$ ④

Obecné kaskádování (Fractional cascading)

TBD

https://en.wikipedia.org/wiki/Fractional_cascading
<http://mj.ucw.cz/vyuka/1516/ds2/>: přednáška 20.4.

- ① Rozdělení je jednoznačně dáno binárním zápisem čísla n a bloků je nejvýše $\log n + 1$.
- ② Prvky nemáme uloženy v jedné datové struktuře, ale máme $\mathcal{O}(\log n)$ instancí datové struktury.
- ③ $\sum_{i: B_i \neq \emptyset} B_S(2^i) = \sum_{i: B_i \neq \emptyset} 2^i \frac{B_S(2^i)}{2^i} \leq \sum_{i: B_i \neq \emptyset} 2^i \frac{B_S(n)}{n} = n \frac{B_S(n)}{n} = B_S(n)$
- ④ $\log n + \sum_{i: B_i \neq \emptyset} Q_S(2^i) \leq \log n + \sum_{i: B_i \neq \emptyset} Q_S(n) \leq \mathcal{O}(Q_S(n) \log n)$

- ① Pokud neumíme efektivně vyjmenovat všechny prvky v datové struktuře, pak si navíc pamatujeme seznam prvků v každém bloku. Alternativně můžeme rovnou vytvořit B_i , jestliže před insertem byl B_i nejmenší neprázdný blok.

- ① Například hešování s lineárním přidáváním.
- ② Maximální poměr počtu označených a neoznačených prvků můžeme nastavit na libovolnou konstantu.
- ③ Pokud blok B_{i-1} existuje, tak vytvoříme B_i spojením nesmazaných prvků z původního B_i a všech prvků z B_{i-1} .
- ④ Alternativně můžeme přebudovat celou datovou strukturu, jestliže je většina prvků označena.

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

Problém

- Máme více dat než se nám vejde do paměti
- Chtěli bychom alespoň umět rozhodnout, zda prvek leží v seznamu
- Povolíme malou chybu odpovědi na vyhledávací dotaz

Bloom filtr

- Jestliže Bloom filtr prohlásí, že daný prvek v seznamu není, pak v seznamu doopravdy nesmí být
- Jestliže Bloom filtr prohlásí, že daný prvek v seznamu je, pak v seznamu není s pravděpodobností nejvýše p
- Pravděpodobnost se nepočítá přes volbu prvků, ale přes volbu hešovací funkce

Bloom filtry: Vylepšení

Vylepšení

- Použijeme k bitových polí B_1, \dots, B_k velikosti m
- Pro i -té bitové pole vygenerujeme úplně nezávislou hešovací funkci $h_i : U \rightarrow [m]$
- Pro každý prvek $x \in S$ a každé i nastavíme $B_i[h_i(x)] = 1$
- Ostatní bity B_1, \dots, B_k jsou nulové

Operace Find(x)

- Jestliže existuje i takové, že $B_i[h_i(x)] = 0$, pak prvek x v seznamu S určitě není
- Za předpokladu, že $x \notin S$, odhadneme pravděpodobnost, že $B_i[h_i(x)] = 1$ pro všechna i
- $P[\forall i \exists x_i \in S : h_i(x) = h_i(x_i)] \leq \left(\frac{n}{m}\right)^k$
- Volbou $m = \lceil en \rceil$ a $k = \lceil \ln \frac{1}{p} \rceil$ dostaneme $\left(\frac{n}{m}\right)^k \leq p$

Paměť

Potřebujeme přibližně $1,88 \cdot n \log_2 \frac{1}{p}$ bitů

Počítací filtry (Counting filters: Fan, Cao, Broder, 2000)

Cíl

Chceme umět mazat prvky v Bloom filtru

Popis

- Místo pole bitů B použijeme pole čítačů C
- $C[j]$ je počet dvojic (x, h_i) takových, že $x \in S$ a $h_i(x) = j$
- Find: Odpovíme ano, jestliže $C[h_i(x)] > 0$ pro všechny hešovací funkce h_i
- Insert/Delete: Zvýšíme/Snížíme hodnoty čítačů $C[h_i(x)]$ o jedna pro všechny hešovací funkce h_i
- Musíme zvolit dostatečně velké čítače, aby k přetečení docházelo s dostatečně malou pravděpodobností

Volba velikostí čítačů

- Hodnotu j -tého čítače odhadneme $P[C[j] \geq s] \leq \frac{\binom{nk}{s}}{\binom{m}{s}} \leq \left(\frac{enk}{sm}\right)^s$
- Volbou $k = \frac{m}{n} \ln 2$ dostáváme $P[\max_j C[j] \geq s] \leq m \left(\frac{e \ln 2}{s}\right)^s$
- Pro 4-bitový čítač dojde k přetečení pro $C[j] = 16$ s pravděpodobností $P[\max_j C[j] \geq 16] \leq 1,37 \cdot 10^{-15} m$

Bloomier filtry (Chazelle, Kilian, Rubinfeld, Tal, 2004)

Konstrukce pole R

- Najdeme hešovací funkce h_1 a h_2 takové, že kukaččí graf neobsahuje kružnice
 - Pro $m \geq 2cn$, kde $c > 2$, jsme dokázali, že očekávaný počet pokusů k nalezení takových hešovacích funkcí je $O(1)$
- V každé komponentě zvolíme jeden vrchol j , kterému nastavíme $R[j] = 0$
- Strom projdeme do hloubky z vrcholu j
- Pro každou hranu odpovídající prvku x , jestliže pozice $R[h_1(x)]$ je již nastavena, pak nastavíme $R[h_2(x)] = R[h_1(x)] \oplus v(x)$

Analýza

- Paměť: $4nr$ bitů (lze vylepšit použitím většího počtu hešovacích funkcí)
- Složitost operace Find: $O(1)$ v nejhorším případě
- Očekávaná složitost operace Build: $O(n)$ pokud umíme s prvky U pracovat v konstantním čase
- Paměťová složitost operace Build: $O(n(r + \log n))$ bitů

Popis

- Máme univerzum $U = \{0, 1, \dots, u-1\}$ všech prvků
- Chceme uložit podmnožinu $S \subseteq U$ velikosti n
- Použijeme bitové pole B velikosti $m = \lceil \frac{n}{p} \rceil$
- Uvažujeme 1-universální hešovací funkci $h : U \rightarrow [m]$
- Pro každý prvek $x \in S$ nastavíme $B[h(x)] = 1$, ostatní bity B jsou nulové

Operace Find(x)

- Jestliže $B[h(x)] = 0$, pak prvek x v seznamu S určitě není
- Jestliže $B[h(x)] = 1$, pak se může stát, že existuje prvek $y \in S$ takový, že $h(x) = h(y)$
- Pravděpodobnost, že $B[h(x)] = 1$, i když $x \notin S$, je $P[\exists y \in S : h(x) = h(y)] \leq p$

Paměť

Potřebujeme $\lceil \frac{n}{p} \rceil$ bitů, aby pravděpodobnost chybné odpovědi byla nejvýše p .

Bloom filtry (Bloom, 1970)

Bloom, 1970

- Použijeme úplně nezávislé hešovací funkce $h_1, \dots, h_k : U \rightarrow [m]$ a jedno bitové pole B velikosti m
- Pro každý prvek $x \in S$ a pro každou hešovací funkci nastavíme $B[h_i(x)] = 1$
- Ostatní bity B jsou nulové

Operace Find(x)

- Jestliže existuje i takové, že $B[h_i(x)] = 0$, pak prvek x v seznamu S určitě není
- Jestliže $B[h_i(x)] = 1$ pro všechny hešovací funkce h_i , pak odpovíme, že x v seznamu S je, i když odpověď může být špatná

Analýza (bez důkazu)

- Pro dané n a m je pravděpodobnost chyby nejmenší pro $k = \frac{m}{n} \ln 2$
- Chceme-li chybu s pravděpodobností nejvýše p , pak zvolíme $k = \log_2 \frac{1}{p}$ a $m = 1,44 \cdot n \log_2 \frac{1}{p}$

Pagh, Pagh, Rao (2005)

Ukázali variantu Bloom filtrů, které stačí $n \log_2 \frac{1}{p}$ bitů

Bloomier filtry (Chazelle, Kilian, Rubinfeld, Tal, 2004)

Cíl

- Chceme si pamatovat r -bitové číslo $v(x)$ pro každý prvek $x \in S$
- Jestliže $x \in S$, pak operace Find musí vrátit $v(x)$
- Jestliže $x \notin S$, pak operace Find může vrátit cokoliv
- Operace Find nemusí umět poznat, jestliže x leží v S

Modifikace kukaččího hešování

- Použijeme pole r -bitových čísel R velikosti m
- a dvě úplně nezávislé hešovací funkce $h_1, h_2 : U \rightarrow [m]$
- Operace Find vrátí $R[h_1(x)] \oplus R[h_2(x)]$

Obsah

- 1 Amortizovaná analýza
- 2 Splay strom
- 3 (a,b)-strom a červeno-černý strom
- 4 Haldy
- 5 Cache-oblivious algorithms
- 6 Hešování
- 7 Geometrické datové struktury
- 8 Dynamizace
- 9 Bloom Filtry
- 10 Literatura

- [1] R Bayer and E McCreight.
Organization and maintenance of large ordered indexes.
Acta Informatica, 1:173–189, 1972.
- [2] Mark R Brown and Robert E Tarjan.
A representation for linear lists with movable fingers.
In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 19–29. ACM, 1978.
- [3] Bernard Chazelle.
Lower bounds for orthogonal range searching: I. the reporting case.
Journal of the ACM (JACM), 37(2):200–212, 1990.
- [4] Bernard Chazelle.
Lower bounds for orthogonal range searching: part ii. the arithmetic model.
Journal of the ACM (JACM), 37(3):439–463, 1990.
- [5] Bernard Chazelle and Leonidas J Guibas.
Fractional cascading: I. a data structuring technique.
Algorithmica, 1(1-4):133–162, 1986.
- [6] Michael L Fredman, János Komlós, and Endre Szemerédi.
Storing a sparse table with 0 (1) worst case access time.
Journal of the ACM (JACM), 31(3):538–544, 1984.
- [7] Michael L Fredman and Robert Endre Tarjan.
Fibonacci heaps and their uses in improved network optimization algorithms.
Journal of the ACM (JACM), 34(3):596–615, 1987.
- [8] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran.
Cache-oblivious algorithms.
In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 285–297, 1999.
- [9] Leo J Guibas, Edward M McCreight, Michael F Plass, and Janet R Roberts.
A new representation for linear lists.
In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 49–60. ACM, 1977.
- [10] Leo J Guibas and Robert Sedgwick.
A dichromatic framework for balanced trees.
In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 8–21. IEEE, 1978.
- [11] Scott Huddleston and Kurt Mehlhorn.
A new data structure for representing sorted lists.
Acta informatica, 17(2):157–184, 1982.
- [12] Donald B Johnson.
Priority queues with update and finding minimum spanning trees.
Information Processing Letters, 4(3):53–57, 1975.

- [13] Donald E. Knuth.
Optimum binary search trees.
Acta informatica, 1(1):14–25, 1971.
- [14] Donald Ervin Knuth.
Notes on "open"addressing.
<http://algo.inria.fr/AofA/Research/11-97.html>, 1963.
- [15] Kurt Mehlhorn.
Sorting presorted files.
In *Theoretical Computer Science 4th GI Conference*, pages 199–212. Springer, 1979.
- [16] Jürg Nievergelt and Edward M Reingold.
Binary search trees of bounded balance.
SIAM journal on Computing, 2(1):33–43, 1973.
- [17] Anna Pagh, Rasmus Pagh, and Milan Ruzic.
Linear probing with constant independence.
In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 318–327, 2007.
- [18] Rasmus Pagh and Flemming Friche Rodler.
Cuckoo hashing.
Journal of Algorithms, 51(2):122–144, 2004.

- [25] Daniel Dominic Sleator and Robert Endre Tarjan.
Self-adjusting binary search trees.
Journal of the ACM (JACM), 32(3):652–686, 1985.
- [26] Jean Vuillemin.
A data structure for manipulating priority queues.
Communications of the ACM, 21(4):309–315, 1978.

- [19] Mihai Patrăşcu.
Better guarantees for chaining and linear probing.
<http://infoweeekly.blogspot.cz/2010/02/better-guarantees-for-chaining-and.html>.
blogspot, February 2, 2010.
- [20] Mihai Pătraşcu and Mikkel Thorup.
On the k-independence required by linear probing and minwise independence.
In *International Colloquium on Automata, Languages, and Programming*, pages 715–726, 2010.
- [21] Mihai Pătraşcu and Mikkel Thorup.
The power of simple tabulation hashing.
Journal of the ACM (JACM), 59(3):14, 2012.
- [22] Jeanette P Schmidt and Alan Siegel.
The spatial complexity of oblivious k-probe hash functions.
SIAM Journal on Computing, 19(5):775–786, 1990.
- [23] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan.
Chernoff-hoeffding bounds for applications with limited independence.
SIAM Journal on Discrete Mathematics, 8(2):223–250, 1995.
- [24] Daniel D Sleator and Robert E Tarjan.
Amortized efficiency of list update and paging rules.
Communications of the ACM, 28(2):202–208, 1985.