

Velke otazky

● **Definujte Splay strom. Vyslovte a dokažte větu o amortizované složitosti operace Splay.**

Definice splay stromu

- binární vyhledávací strom
- samovyvazující se
- používá splay operace (zig/zag ...)

Amortizovaná složitost operace splay:

- amortizovaná cena je nejvýše $3(r'(v) - r(v)) + 1$
- což dává složitost $O(\log n)$

Důkaz

- Celková cena je rovna sumě jednotlivých kroků
- Mejdme hloubky x po 0 až k krocích: $r_0(x), \dots, r_k(x)$
- Cena i -tého kroku je $3r_i(x) - 3r_{i-1}(x)$, plus 1 pokud je to ZIG rotace
- Amortizovaná cena $\leq \sum_{i=1}^t (3r_i(x) - 3r_{i-1}(x)) + 1$
- Amortizovaná cena $\leq 3r_t(x) - 3r_0(x) + 1$
- Amortizovaná cena $\leq 3 \log_n + 1 = O(\log n)$

● **Definujte (a,b)-strom. Popište, jak na něm probíhají operace Find, Insert a Delete. Rozeberte jejich složitost v nejhorším případě.**

Definice

- vyhledávací strom
- všechny vrcholy kromě kořene mají počet synů A až B
- kořen má nejvýše B synů
- Všechny listy jsou ve stejné výšce

- Platí $a \geq 2$ & $b \geq 2a - 1$

Find

- V každém vrcholu najdeme nejmenší větší klic a do jeho dítěte vstoupíme
- Končíme v listu

Insert

- Najdeme správného otce
- Pridáme prvek
- Pokud se nevejde rozdělit děti na dvě části a přidáme nový klic do rodiče
- Opakujeme dokud se klíče nevejdou

Delete

- Najdeme správného otce
- Odebereme prvek
- Pokud je klicu málo, přesuneme klic z bratra, nebo se spojíme s bratrem

Nejhorsí případ - složitost

- všechny $O(\log n)$

 **Formulujte cache-aware a cache-oblivious algoritmy pro transpozici čtvercové matice. Rozeberte jejich časovou složitost a I/O složitost.**

Cache aware

- rozdělíme matici na submatice $d \times d$ tak že $2d^2 \leq B$ (2 submatice se vejdou do cache naraz)
- transponujeme obě submatice
- pokud nejsou na diagonále, tak je prohodíme

Cache oblivious

- Rekursivně dělíme matici na čtvrtiny dokud nemáme matici 1×1
- $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$

- A_{11} a A_{22} se pouze transponují
- A_{12} a A_{21} se transponují a prohodi navzájem


Složitost

Cache aware

- počet podmatic je $\frac{k^2}{d^2}$
- I/O složitost: $O(\frac{k^2}{B})$, (+1 pokud nebude cache zarovnaná)
- časová složitost: $\Omega(k^2)$
- $k \times k$ je velikost matice
- $d \times d$ je velikost podmatice
- B je velikost bloku paměti

Cache oblivious

- předpoklad *tall cache*, neboli do cache se nam vejdou alespoň 4 bloky
- $\frac{k^2}{d^2} * 4d \leq \frac{8k^2}{B} = O(\frac{k^2}{B})$, (+1 pokud k není mocninou dvojky)

 **Definujte c -univerzální a k -nezávislé rodiny hešovacích funkcí. Formulujte a dokažte větu o střední délce řetězce v hešování s řetězcí. Ukažte příklady c -univerzálních a k -nezávislých rodin pro hešování přirozených čísel.**

c -univerzální

- náhodně zvolená $h \in H$ splňuje $P[h(x) = h(y)] \leq \frac{c}{m}$, (pro všechny dvojice $x \neq y \in U$)
- počet hešovacích funkcí $h \in H$ takových že $h(x) = h(y)$ je nejvýše $\frac{c|H|}{m}$, (pro všechny dvojice $x \neq y \in U$)

k -nezávislý

- $k \in \mathbb{N}$; $K = \{1, \dots, k\}$ a $c \geq 1$
- náhodně zvolená $h \in H$ splňuje:

- $P[h(x_i) = z_i \forall i \in K] \leq \frac{c}{m^k}$
- pro všechna po dvou různá $x_1, \dots, x_k \in U$ a všechna $z_1, \dots, z_k \in M$

věta o střední délce řetězce v hešování s řetězci

- Jestliže H je c -univerzální, pak očekávaný počet prvků v přiřadné $h(x)$ pro $x \in U$ je nejvýše $\frac{cn}{m}$

Důkaz

- $E[|\{y \in S : h(x) = h(y)\}|] = \sum_{y \in S} P[h(x) = h(y)] \leq \frac{cn}{m}$

příklady

Multiply mod prime

- $h_{a,b}(x) = ax + b \pmod{p} \pmod{m}$
- 2-univerzální
- (2,4)-nezavisly

Tabulkové hashování

- $h(x) = T_1(x^1) \text{ XOR } \dots \text{ XOR } T_d(x^d)$
- 3-nezavisle

Popište a analyzujte hešování s lineárním přidáváním. (Linear probing)

- INSERT
 - Vkladanou hodnotu zahesujeme a vložíme na index rovný jeho hashu
 - Pokud tam už něco je, tak přidáme k indexu 1 a zkusíme to znovu (opakujeme dokud prvek nevložíme)
- DELETE
 - nelze, pouze vytvoříme pomníček a občas prehashujeme
- Vytváří shluky, což není dobré
- Práteleška na cache
- poloprázdna a bez velkých shluků se chová dobře

● Definujte vícerozměrné intervalové stromy. Rozeberte časovou a prostorovou složitost konstrukce a obdélníkových dotazů (včetně zrychlení kaskádováním).

- Vylepsené intervalové stromy (zlepší složitost dotazu)
- Binární vyhledávací strom podle x
 - Každý vrchol má další BVS podle y
- Prostor: $O(n \log^{d-1} n)$
- Cas:
 - BUILD: $O(n \log^{d-1} n)$
 - INSERT: $O(\log^d n)$
 - DELETE: $O(\log^d n)$
 - FIND RANGE: $O(\log^d n + p)$
 - kaskádování bude v $O(\log^{d-1} n + p)$
 - všem prvkům nahore předpocítáme pozici ve stromu níže

● Definujte suffixové pole a LCP pole. Popište a analyzujte algoritmy na jejich konstrukci.

- Suffixové pole S
 - pro řetězec α délky n
 - udává lexikografické pořadí suffixu daného slova α
 - lze vybudovat v $O(n)$
 - postavíme suffixový strom a pak jej sepišeme do pole
- LCP pole
 - $\text{LCP}(\alpha, \beta) = \max\{k \mid \alpha[:k] == \beta[:k]\}$
 - udává délku nejdelšího společného prefixu α a β
 - lze vybudovat v $O(n)$ se suffixovým polem

Malé otázky

● Popište „nafukovací pole“ se zvětšováním a zmenšováním. Analyzujte jeho amortizovanou složitost.

- zvětší se na 2x při zaplnění
- zmenší se na 1/2 při 1/4 zaplnění
- Amortizovane $O(1)$

● Popište vyhledávací stromy s líným vyvažováním (BB[α] stromy). Analyzujte jejich amortizovanou složitost.

- binární vyhledávací strom
- každý podstrom syna musí být velikosti maximálně α násobek všech dětí jejich otce
- $\frac{1}{2} \leq \alpha \leq 1$
- hloubka je $\theta(\log n)$

● Navrhněte operace Find, Insert a Delete na Splay stromu. Analyzujte jejich amortizovanou složitost.

- vše amortizovane v $O(\log n)$
- zig / zag dvojrotače rotače krom kořene

● Vyslovte a dokažte věty o amortizované složitosti operací Insert a Delete na $(a, 2a-1)$ -stromech a $(a, 2a)$ -stromech.

- Sekvence m Insertu a Deletu na začátku prázdném $(a, 2a)$ stromu vykoná $O(m)$ akcí.

Dukaz

- Cena = pocet zmenenych vrcholu
- Ukazeme ze potencial je mensi roven nule
- Mejme funkci $f(k)$ jakozto zmenu k deti, ktera musi splnovat
 - $|f(i) - f(i + 1)| \leq c$, kde c je libovolna konstanta
 - $f(2a) \geq f(a) + f(a + 1) + c + 1$
 - $f(a - 2) + f(a - 1) \geq f(2a - 2) + c + 1$
- nastavime $c = 2$ a vykouzlime:

k	a-2	a-1	a	...	2a-2	2a-1	2a
f(k)	2	1	0	0	0	1	2

- INSERT prida klic ($O(1)$), zmeni potencial o $O(1)$ a vykona nekolik rozdeleni s amortizovanou slozitosti 0.
- DELETE odebere klic ($O(1)$) a vykona sekvenci spojeni s amortizovanou slozitosti 0.
 - Pripadne pokud si vezme dite bratra, coz ma slozistost $O(1)$ a muze se stat pouze jednou.

Takze amortizovana cena je konstantni pro obe operace, jelikoz potencial je porad nezaporny a zacina v nule.

Tudiz m operaci Insert a Delete provede $O(m)$ modifikaci vrcholu.

● Analyzujte k-cestný Mergesort v cache-aware modelu. Jaká je optimální volba k?

- Optimalni hodnota $K = \lfloor M/2B \rfloor$
- Pocet pruchodu klesne na $\lceil \log_k N \rceil$
- Jeden krok zabere $O(\log K)$ a cely mergesort $O(N \log N)$

● Vyslovte a dokažte Sleatorovu-Tarjanovu větu o kompetitivnosti LRU.

- LRU = least-recently used
- NEPLATI: LRU je k-kompetitivni, neboli $C_{LRU} \leq k * C_{OPT}$

Veta:

- $C_{LRU} \leq \frac{M_{LRU}}{M_{LRU} - M_{OPT}} * C_{OPT} + M_{OPT}$

Dukaz:

- máme epochy $E_0 \dots E_k$
- LRU zaplatí v každé epoche M_{LRU} a v první maximálně M_{LRU}
- v nenulové epoche i :
 - i. různé bloky: OPT platí alespoň bloky které neměl v cache ($M_{LRU} - M_{OPT}$)
 - ii. LRU zaplatí za blok 2x: alespoň M_{LRU} různých bloků
- v nulové epoche:
 - i. LRU i OPT začíná s prázdnou cache: $C_{LRU} = \# \text{ různých bloků} = C_{OPT}$
 - ii. LRU začíná s neprázdnou oproti OPT: neškodí LRU
 - iii. LRU i OPT mají neprázdnou cache: to je ta $+M_{OPT}$

● Popište systém hešovacích funkcí odvozený ze skalárního součinu. Dokažte, že je to 1-univerzální systém ze Z_p^k do Z_p .

- Mejsme d -dimenzionalni vektory nad telesem Z_p
- $h_t(x) = t \times x$

Dukaz

- $P[h_t(x) = h_t(y)] = P[t \times x = t \times y] = P[t \times (x - y) = 0] = P[\sum_{i=1}^k (x_i - y_i)t_i = 0] = P[(x_k - y_k)t_k = -\sum_{i=1}^{k-1} (x_i - y_i)t_i]$
- Posledni krok rika, ze posledni iterace sumy by se musela presne trefit a na to ma pravdepodobnost $1/p$
- Neboli existuje prave jedno t_k takove aby rovnost platila a zaroven $t_k \in Z_p$

● Popište systém lineárních hešovacích funkcí. Dokažte, že je to 2-nezávislý systém ze Z_p do $[m]$.

- $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$
- Mejsme linearni system L , jez je $(2,1)$ -nezavisly v Z_p a po zmoduleni do m je $(2,4)$ -nezavisly podle dukazu nize

Dukaz

- snazime se dokazat $P[h_{a,b}(x) = i \wedge h_{a,b}(y) = j] \leq 4/m^2$, kde $x, y \in [p]$ a $i, j \in [m]$ a nahodnou dvojici (a, b)
- obe strany posoudime nezávisle a pak jejich pravdepodobnost pronasobime
- pravdepodobnost jedne strany je maximalne $2p/mp$
- pravdepodobnost obou zaroven je maximalne $(2p/mp)^2 = 4/m^2$
- tudiz je stale 2-nezávisly

● Sestrojte k-nezávislý systém hešovacích funkcí ze Z_p do $[m]$.

- použijeme polynomiální hashování (poly-mod-prime)
- $h_a(x) = \sum_{i=0}^{k-1} a_i x^i$
- $P_k = \{h_t | t \in Z_p^k\}$
- takovýto systém je $(k,1)$ -nezávislý

Prevod do m

- $P_k \bmod m$
- je $(k,4)$ -nezávislý
- pokud $p \geq 2km$, tak je $(k,2)$ -nezávislý

● Sestrojte 2-nezávislý systém hešovacích funkcí hešujících řetězce délky nejvýše L nad abecedou $[a]$ do $[m]$.

- použijeme poly-mod-prime
 - $h_{a,b,c}(x_1, \dots, x_d) = (b + c * \sum_{i=0}^{d-1} x_{i+1} * a^i \bmod p) \bmod m, (p > m)$
- řetězec doplníme zprava nulami do délky L
- při výpočtu hashe můžeme ignorovat prázdné znaky, neboť $0 * \text{cokoliv}$ je 0 a součet to nezmění

● Popište a analyzujte Bloomův filtr.

- Umi insert, neumí delete a find dává false-positive
- paměťově úsporný

- Insert vklada na pozici zahashovane hodnoty
- False positive je n/m (n je pocet prvku, m je velikost nasi datove struktury)
- Multi-band (k hash funkci)
 - $k = \lceil \log 1/\epsilon \rceil$, kde ϵ je pravdepodobnost false-positive
 - potrebna pamet je $m \lceil \log 1/\epsilon \rceil$, (m muze byt treba $2n$)

● Ukažte, jak provádět 1-rozměrné intervalové dotazy na binárním vyhledávacím stromu.

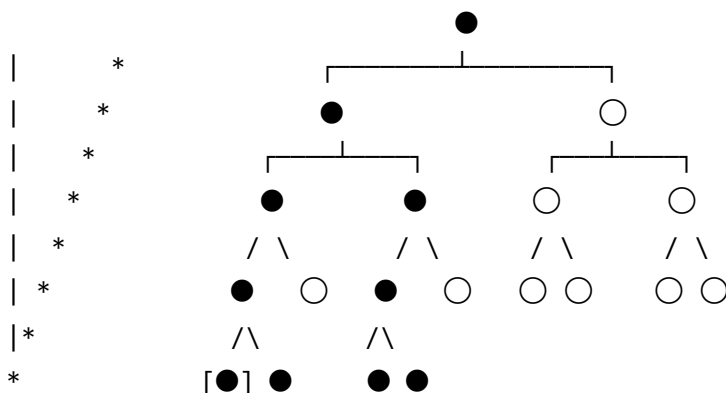
- najdeme levý vrchol, najdeme pravý vrchol a všechny vrcholy / podstromy mezi nimi vypíšeme
- $O(\log n + p)$, kde p je počet vrácených vrcholů

● Definujte k-d stromy a ukažte, že 2-d intervalové dotazy trvají $\Omega(\sqrt{n})$.

- Binární vyhledávací strom
- na i -té hladině porovnáme i -tou dimenzi
- Build trvá $O(n \log n)$

intervalový dotaz maximálně \sqrt{n} :

- varovný příklad:
 - máme dotaz na 2d strom a hledáme všechny prvky na ose y
 - na každé sudé úrovni se musí procházet oba synové
 - počet navštívených listů je $2^{(\log n)/2} = \sqrt{n}$



● Ukažte, jak dynamizovat k-rozměrné intervalové stromy (stačí Insert).

- amortizovane $O(\log^d n)$
- jeden insert přida
 - 1 vrchol v 1. dimenzi
 - insert do stromu 1. dimenze trva $O(\log n)$, (včetně lineho vyvazovani)
 - bohužel se občas musí postavit celý strom v 2. dimenzi což trva $O(n \log n)$
 - $O(\log n)$ vrcholu v 2. dimenzi
 - insert do stromu 2. dimenze trva $O(\log n)$
 - takže v 2. dimenzi máme celkovou složitost $O(\log^2 n)$
 - ... v dalších dimenzích

● Ukažte, jak použít suffixové pole a LCP pole na nalezení nejdelšího společného podřetězce dvou řetězců.

- spojíme řetězce α , " $\#$ " a β , kde $\#$ je nový symbol do řetězce $\alpha\#\beta$
- sestrojme pro tento řetězec suffixové pole a LCP
- hledáme dvojici po sobě jdoucích indexů i a j , takovou ze:
 - i je před znakem $\#$ a j je za znakem $\#$, nebo přesně obráceně
- vrátíme dvojici která bude mít největší LCP
- složitost $O(n)$

● Ukažte, jak paralelizovat (a,b)-strom.

- použijeme top-down balancování
- INSERT:
 - držíme zámek na aktuálním vrcholu a jeho rodiči
 - pokud je potřeba tak vrchol rozdělíme
 - poté odemkneme rodiče
 - zamkneme dítě které potřebujeme a pokračujeme tam
- DELETE:
 - stejně jako insert
 - ale zamykáme i bratra, kdyby náhodou bychom jej potřebovali připojit

- tady muze nastat deadlock, tak musime vybírat systematicky, treba jen presne o jedna leveho bratra
- pro odstraneni ne-listu se muze stat, ze potrebujeme zamknout root pri hledani vhodneho potomka
 - radeji vytvorit pomnik misto mazani

● Navrhnete a analyzujete bezzámkovou implementaci zásobníku.

- Mejsme zásobník, který má hodnotu, atomicky ukazatel na následníka a atomicky ukazatel na hlavu

PUSH

```

1 | Repeat:
2 |   $h <- stack.head
3 |   n.next <- h
4 |   if CAS(stack.head, h, n) = h: return

```

POP

```

1 | Repeat:
2 |   h <- stack.head
3 |   s <- h.next
4 |   if CAS(stack.head, h, s) = h: return h

```

- muze nastat *live/lock*, ale velmi nepravdepodobne

● Popište atomická primitiva a jejich vlastnosti. Vysvětlete problém ABA a jeho řešení.

- Atomicita znamená, že ostatní danou věc vydí jako nezapočatou, nebo dokončenou
- primitiva:
 - Read and write - normální paměť RAM
 - Exchange - výměni hodnoty atomického registru a lokální paměti
 - Test and set bit - nastaví bit na hodnotu a vrátí původní
 - Fetch and add - připočte číslo a vrátí původní hodnotu

- Compare and swap (CAS) - dostane old a new, pokud je v registru old, tak ho vymeni za new a puvodni hodnotu vrati
- Load linked and store conditional (LL/SC) - po precteni se prida watcher, který pri zapisu povoli zapsat jen pokud jej nikdo nemenal

ABA

- problem s bezzamkovým zásobníkem, kde může nastat přidání do zásobníku
- CAS sice uvidí tu samou hodnotu, ale ta byla 2x změněna mezitím
- vyřeší se použitím LL/SC místo CAS
 - nebo double CAS

příklad:

Vlakno 1	Vlakno 2	
		head -> A -> B -> null
	h <- stack.head	h=A
	s <- h.next	s=B
x <- POP		x=A
		head -> B -> null
y <- POP		y=B
		head -> null
PUSH(x)		PUSH(A)
		head -> A -> null
	if CAS(stack.head, h, s) = h: return h	h=A
		head -> B -> null

chyba ... kde se nám vzalo B ???