OZD pro středně znalého matfyzáka

Ladislav Láska

17. ledna 2011

`Organizace a Zpracování Dat I` je předmět, který je sice již pouze povinně volitelný, ale i tak se ho někteří mohou pokusit vystudovat, třeba kvůli krásným 5ti kreditům. Tento text provede středně znalého matfyzáka-informatika velice rychle všemi potřebnými kapitolami pro složení zkoušky. Předpokládá se účast na cvičeních a psychická přítomnost, protože většina algoritmů bude pouze ve zkratce. Vhodná je také znalost datových struktur z přednášek Programování I & II a ADS I & II.

Obsah

1 Úložná zařízení

1.1 Značení

Na přednáškách, v příkladech a na slajdech je dost specifické značení. Slovníček:

	Význam	Poznámka
R	délka fyzického záznamu	
B	velikost bloku	
b	blokovací faktor B/R	říká, kolik záznamů se vejde do bloku

2 RAID

Redundant Array of Inexpensive Disks (RAID) je metoda, kdy se slučuje více disků do jednoho virtuálního tak, aby se dosáhlo vyšší spolehlivosti, rychlosti, nebo obojího.

Poznámka Sice se na přednášce dozvíme, že RAID 0 a 1 je podporován levnými řadiči, ale ona to není úplně pravda. Levné řadiče (hlavně ty na desce) většinou implementují "částečně hardwarový RAID", tj. jenom si pamatují nastavení a umí říct ovladači, aby nastavil softwarový RAID. To se může negativně projevit tak, že CPU bude zatíženo výpočty parit a nebude stíhat jiné oprace (operace zápisu na RAID 5 je relativně náročná na výpočetní výkon). Proto se pro každou solidní aplikaci vyplatí mít speciální řadič, který umí skutečně plný hardwarový RAID (tedy operační systém už uvidí pole jako jeden disk).

2.1 RAID 0 - striping

Není redundantní, pouze rozkládá data na více disků. Data jsou rozkládána na menší bloky a ty jsou střídavě ukládána na různé disky. Chyba jednoho disku může vést ke ztrátě dat na všech!

2.2 RAID 1 - mirroring

Totožná data se zapisují na dva disky. Zápis je pomalý jako pomalejší z disků, čtení může být až 2x rychlejší než u jednoho disku - data lze číst z obou disků.

2.3 RAID 3 a 4 - striping with parity

Prakticky RAID 0, ale má jeden disk na paritu. Umí tedy přeží výpadek až jednoho disku.

2.4 RAID 5 - striping with distributed parity

Data jsou rozložena a stripována jako u RAIDu 0, ale proložena paritou (tj. parity různých bloků můžou být na různých discích). Zápis na paritní disk tolik nebrzdí (typicky se bude parita zapisovat na více disků naráz), je ale náročnější na výpočet.

2.5 RAID 6

Jako RAID 5, ale paritní bloky jsou distribuované dva.

2.6 Vnořená pole RAID

Úrovně RAID se dají samozřejmě kombinovat do stromovitých struktur. Typicky se zapisují jako například RAID 0+1. Pořadí čísel odpovídá vnoření úrovní od pomyslného kořene stromu disků, druhé číslo jeho synům atp.

2.7 Magnetické pásky

Magnetické pásky jsou sice velmi stará technologie, nicméně pro svou spolehlivost a cenu stále hojně používanou zejména pro zálohování. Pro představu páska 1.6TB pro zařízení s přenosovou rychlostí 240MB/s stojí dnes cca 800kč (ke dni 14. 1. 2011), což je vedle ostatních variant bezkonkurenční cena. Nevýhodou samozřejmě je vysoká pořizovací cena mechaniky. Další nevýhodou je, že je stavěná na sekvenční přístup a náhodný je velmi pomalý (převíjení).

2.8 Flash paměti a SSD disky

Flash paměti (SSD jsou flash) jsou nové, rychlé a drahé paměti. Jejich výhoda je taková, že umožňují skutečně náhodný přístup k datům a velké přenosové rychlosti. Nevýhoda je přecejenom nižší kapacita než u klasických disků a také omezení zápisu: je omezen počet zápisů na disk a zápis vyžaduje cyklus read-modify-write na celém vnitřním bloku (což je něco okolo 128kB).

3 Uložení dat na disku

Následující dva odstavečky pojednávají o způsobech ukládání dat na disk. Je to pouze formalita. Ve skutečnosti všechny následující algoritmy jdou použít na obě metody přístupu s tím, že v případě indexovaného souboru je datová struktura perzistentně uložena zvlášť a odkazuje do datového souboru. V případě "pouze"datového souboru je celá struktura uložena v něm a data jsou součástí uzlů. Oba přístupy mají svá pozitiva a negativa, která se poznají většinou až při obrovských množstvích dat. Jaká to přesně jsou přenechám na čtenáři.

3.1 Indexed Sequential Access Method (ISAM)

ISAM je způsob přístupu k datům, kdy jsou data uložena v sekvenčním souboru (často v pořadí, v jakém byla vložena) a ve zvláštním souboru je držen index (který do hlavního, též primárního, souboru ukazuje).

Poznámka autora: na přednáškách se z toho dělá hrozná věda, míchá se do toho indexový soubor a obecně je to hrozný chaos. Prakticky je indexový soubor a ISAM to samé, jenom se to jinak nazývá a říká se, že je to jiné.

3.2 Datový soubor

Datový soubor je jednoduchá mapa paměti datové struktury na disk. Je to nejjednodušší metoda přístup, bohužel bez cachování není zrovna užitečná.

4 Datové struktury

4.1 Perfektní hashování (Cormack)

Mějme primární hashovací funkci $h(k)$ a sadu sekundárních funkcí $h_i(k, r)$. Zavedeme si dvě tabulky - primární a sekundární. Primární tabulka se indexuje pomocí primární hashovací funkce a jeden její řádek říká, kolik hodnot se stejným primárním hashem je uloženo (sloupec **r**). Sloupec **p** určuje, kde začínají tyto konfliktní záznamy (tento úsek je vždy spojitý!) a sloupec **i** říká, jakou hashovací funkcí h_i lze záznamy rozlišit (tj. udává pozici našeho záznamu v intervalu $(p, p+r-1)$). Následující příklad ukazuje 3 vložené hodnoty: jedna nekonfliktní A a dvě konfliktní BB a BC.

	p	**i**	**r**
0	0	-	1
1			
2	1	5	2
3			

p	**val**
0	A
1	BB
2	BC
3	

Vkládání Při vkládání nalezneme nejdříve příslušný řádek v primární tabulce a pokud je neobsazený, založíme jej (a hodnotu vložíme do sekundární tabulky na první volné místo). Pokud je řádek v primární tabulce obsazený, zvětšíme **r** a nalezneme takové **i**, že bude odlišovat všechny záznamy (včetně těch, které již v tabulce jsou). Následně všechny prvky vložíme do sekundární tabulky (ty co tam už jsou vložíme znovu a původní místa uvolníme) setříděné podle hodnot jejich hashů.

4.2 Perfektní hashování (Larson & Kajla)

Data budeme ukládat do d přihrádek. Navíc každé přihrádce přiřadíme signaturu velikosti b bitů (kde 2^b je počet záznamů ve stránce), ta je na začátku maximální možná. Pořídíme si sadu hashovacích funkcí $H_i(k, d)$ a $S_i(k, b)$, kde H_i bude ukazovat na stránky a S_i počítat signatury.

Vkládání Nejdříve nalezneme i takové, aby H_i ukazovala na stránku p, že $sep(p) > S_i(k)$. Pokud v této přihrádce je volno, záznam vložíme a jsme hotovi. Pokud volno není, stránku musíme štěpit - to uděláme tak, že ze stránky vyjmeme záznamy s nejvyšší signaturou, separátor stránky snížíme na tuto signaturu a všechny je vložíme znovu do databáze (záznam, který jsme vkládali původně, v této stránce již zůstane - rozmyslete si, proč by do této stránky stejně spadnul i "opakovaném"vkládání).

4.3 Rozšiřitelné hashování (Fagin)

Mějme hashovací funkci $h(k)$ vracející binární číslo a adresář velikosti d. Navíc předpokládejme, že $d = 2^i$ pro nějaké i. Stránky odkazované z adresáře mají libovolnou kapacitu, minimálně však 1. Označme $b := \log_2 d$. V adresáři adresujeme prvními b-bity z $h(k)$.

Vkládání Při vkládání nejdříve nalezneme příslušnou stránku (triviálně zahashujeme klíč a v adresáři najdeme pointer). Pokud se do stránky záznam vejde, vložíme ho. Pokud je stránka plná, musíme stránku rozštěpit.

Štěpení stránky Nejprve se podíváme, kolikrát je stránka uložena v adresáři (rozmyslete si, že odkazy na tuto stránku jsou všechny v souvislém bloku). Pokud se stránky vyskytuje pouze jednou, musíme rozštěpit adresář (a tím zajistíme, že se bude vyskytovat dvakrát). Pokud vícenásobně, najdeme bit ve kterém se stránky naposledy shodují a v něm rozštěpíme. Záznamy do dvou nových stránce přerozdělíme (aby byly na svém místě) a nový záznam přidáme.

Štěpení adresáře Adresář štěpíme tak, že zvětšíme jeho hloubku o jeden bit. Ukazatele na stránky vždy zkopírujeme (rozštěpený řádek ukazuje na ty samé stránky).

4.4 Lineární hashování (Litwin)

Mějme hashovací funkci $h(k)$ vracející binární číslo o d bitech. Mějme také konstantu L, která značí, kolik vložení musíme vykonat mezi štěpeními. Začneme s jednou stránkou a oblastí přetečení. Pro lepší pochopení si zaveďme pojem **expanze** jako úseky, mezi kterými došlo ke zdvojnásobení počtu stránek. Protože při štěpení se zvětší počet stránek o 1 je zřejmé, že po k-té expanzi bude mít adresář 2^k stránek a také že další expanze nastane za 2^k štěpení. Tedy štěpení v jedné fázi bude právě tolik, kolik máme stránek na začátku fáze expanze. Tady je vidět, že stránky můžeme štěpit jednoduše podle pořadí s tím, že novou stránku zařadíme za poslední.

Vkládání Při vkládání najdeme odpovídající stránku a pokusíme se vložit. Pokud je stránka plná, vložíme záznam do oblasti přetečení. Pokud jsme udělali právě L-té vložení od předchozího štěpení, rozštěpíme

Štěpení Štěpení probíhá tak, že stránku, která je na řadě, rozdělíme podle prvního zatím nepoužitého bitu hashovací funkce (tj. zvětšíme rozlišení o jeden bit). Některé záznamy budou "vyházeny"do nové stránky, kterou umístíme za poslední.

4.5 Skupinové štěpení stránek

Poznámka autora: Skupinové štěpení stránek je varianta Litwina (resp. Litwin je speciální případ skupinového štěpení), kde se stránky navíc dělí do skupin. Ve slajdech to není popsané a vypadá to, že se to u zkoušek objevuje. Pokud to někdo ovládá, rád to sem přidám, nicméně já o tom nevím prakticky nic.

4.6 Částečná shoda

Při dotazech na částečnou shodu hledáme podle více atributů, ale nevyžadujeme, abychom znali všechny (lze hledat například pouze zapomocí poloviny). Atributy navíc umíme ohodnotit pravděpodobností, s jakou se podle nich bude hledat. Implementujeme to tak, že rozdělíme celkovou délku klíče na segmenty různé délky, kde každý atribut bude mít svůj segment. Při vyhledávání se pak budeme řídit jenom částí adresy: to ale má jistou nevýhodu, protože možná budeme muset prohledávat nespojitý adresní prostor.

Určení počtu bitů z pravděpodobností Pro každý atribut A_i nechť máme pravděpodobnost P_i a chceme vypočítat jemu příslušící počet bitů d_i. Spočítejme hodnoty d_i:

$$d_i = \frac{d - \sum_{j=1}^{n} \log_2 P_j}{n} + \log_2 P_i \tag{1}$$

Pokud při výpočtu vyjde hodnota $d_i < 0$, příslušný atribut vyškrtneme a výpočet zopakujeme. Pokud naopak nějaké $d_i > d$, výpočet zastavíme a jako klíč použijeme pouze atribut A_i (a tedy $d_i := d$).

4.7 Efektivní prohledávání stránek

Při dotazech na částečnou schodu se nám může snadno stát, že budeme muset prohledávat relativně velký adresní prostor (který navíc nemusí být spojitý). Ten můžeme zmenšit například použitím nějaké heuristiky (deskriptory stránek), která nám řekne, které stránky nemáme prohledávat (ale neřekne nám, které stránky přímo chceme - může nás tedy nechat něco prohledat zbytečně, ale určitě nám nic nezatají), nebo se snažit prohledat stránky efektivněji (Grayovy kódy).

4.7.1 Deskriptory stránek

Každou stránku budeme charakterizovat w-bitovým číslem, tzv. deskriptorem. Deskriptor složíme jako OR všech "deskriptorů"záznamů ve stránce (nějaký hash). Deskriptor se může skládat z hodnot v různých atributech, libovolně rozdělených (například podle pravděpodobnosti jako ve vyhledávání na částečnou shodu), ale je velice vhodné použít hashovací funkci, která má za výsledek spíše méně 1 a více 0 (více dále).

Vyhledávání Nejdříve vypočteme masku dotazu - prostě z klíčů, které chceme najít spočítáme stejný hash jako v deskriptoru (pro klíče, které nás nezajímají, vezmeme 0). Pokud v příslušných bitech nastane situace, že v masce dotazu je 1 a v deskriptoru je 0, stránka určitě neobsahuje co chceme a můžeme ji přeskočit. Zde je vidět, že pokud by byly deskriptory samé 1, nic se nedozvíme (a protože se deskriptory vytvářejí OR-em, může se to stát snadno).

4.7.2 Grayovy kódy

Grayův kód je způsob zápisu čísla, velmi podobný binárnímu zápisu, kde se po sobě jdoucí čísla liší právě v jedné pozici v binárním zápisu. Tedy například:

dec	bin	Gray
0	000	0000
1	001	0001
2	010	0011
3	011	0010

Takovéto uspořádání má za následek, že pokud prohledáváme množinu stránek, nebude tolik roztroušená (tedy budeme moci prohledávat spojitý adresní prostor). To sice není zaručený efekt, ale v průměrném případě se vyplatí. Navíc platí, že oproti binárnímu kódování si můžeme pouze pomoct (tj. v nejhorším případě je shluk při Grayových kódech nejvíce takový, jako při použití binárního kódování).

4.8 B-strom

Definice B-strom řádu m je strom kde platí invarianty:

1. Každý uzel má n synů, kde $m/2 \leq n \leq m$.
2. Hodnoty v uzlu jsou setříděny a rozdělují mezní hodnoty všech svých synů.
3. Cesta z každého listu do kořene je stejně dlouhá.

Tedy jde o vyváženou datovou strukturu shlukující více dat v jednom uzlu. To má za následek šetření IO operacemi při používání stromu (zejména při vkládání a mazání), zejména pak pokud jsou data ve stromu uložena přímo (což si nejsem vědom, že by bylo chytré dělat při větších než malých objemech).

Operace Operace hledání je triviální z definice. Vkládání a mazání již triviální není, ale každý absolvent předmětu Programování nebo ADS by ho měl zvládat. Připomeňme tedy, že se vkládá do listů, maže se odkudkoliv a hlídá se přetečení/podtečení uzlu. Pokud taková situace nastane, nejdříve se snažíme převést vrchol sousedovi, nebo si od něj naopak půjčit. Pokud to nelze, uzel štěpíme (rozdělíme na dva) a jeho středovou hodnotou oddělíme dva nové uzly v jejich otci. Alternativně při mazání můžeme vrcholy slučovat tak, že si půjčíme jejich společný oddělovač z otce.

Zamykání Zamykání v B-stromech lze řešit různě, nicméně nejefektivnější je zamykání s preventivním štěpením: při každém průchodu uzlem zjistím, zda v něm lze provést operaci vložení/mazání (podle toho, jakou právě chci udělat pod ním) a případně ho preventivně rozštěpím/sloučím, aby udělat šla. Tím zajistím, že v danou chvíli mi stačí mít uzamčené nejvýše dva uzly a zbytek stromu je přístupný ostatním.

Redundantní verze Předchozí strom nazveme neredundantní, protože každý klíč v něm byl obsažen právě jednou. Jednoduchou modifikací však můžeme získat redundantní verzi, kde hodnoty v uzlech rozdělují své syny neostře - a tedy se můžou vyskytnout vícekrát. To využijeme k tomu, že všechny skutečné hodnoty uložíme v listech a hodnoty v uzlech budou jenom pro vyhledávání

4.9 B+-strom

B+-strom je jednoduchá variace na redundantní B-strom, kde navíc jsou listy stromu spojeny ukazateli do spojového seznamu, aby k nim byl možný rychlý sekvenční přístup.

4.10 B*-strom

Při snaze většího průměrného naplnění byl vytvořen B*-strom, cože je varianta B-stromu, kde je požadováno naplnění každého uzlu až 2/3. Aby se taková struktura dala udržovat, je při vkládání a mazání nutno počítat se sousedními uzly a sdílet s nimi prvky. Pokud nelze již sdílet, dochází ke slučování/štěpení - a to tak, že se 3 uzly sloučí do 2 nebo naopak.

Poznámka autora. B-stromy se používají relativně zřídka (filesystémy HFS a Reiser4) a většinou se říká prostě "B-stromy".*