OZD

Zakl. pojmy

zaznam - log. jednotka dat, ma **atributy** se jmeny a domenami (max. 1 hodnota ∀ atribut. **(homogenni) soubor** - kolekce (multimnozina) zaznamu. Pro soubory s neopakujicimi se zaznamy je **klic** mn. atributu, ktere zaznam jednoznacne identifikuji v souboru. 1 z klicu se oznacuje jako **primarni** . **vyhledavaci klic** je neco jineho - k jeho 1 hodnote se da najit mnozina odp. zaznamu. Jsou 3 druhy vyhledavacich klicu: hodnotovy, hashovany a relativni (primo pozice v souboru). Na souboru jsou definovany operace INSERT, DELETE, UPDATE a FETCH.

log. zaznamu odpovida **fyzicky zaznam** delky R na BUNO mag. disku; ten muze obsahovat dalsi data - oddelovace, ukazatele, hlavicky. Fyz. zaznamy jsou organizovany do **bloku** delky B - hl. jednotkou prenosu mezi RAM a HDD. B/R se nazyva **blokovaci faktor** ($\lfloor b \rfloor$). Zaznamy - pevna nebo promenna delka. Schema organizace souboru - popis log. pametove struktury kde se soubor nachazi. Muze obs. vice log. souboru, ten ktery nese uz. data je **primarni** , jeho delka v poctu zaznamu se znaci N. Dalsi operace nad SOS - BUILD, REORGANIZATION, OPEN, CLOSE.

Dotaz - ∀ fce, co kazdemu svemu argumentu priradi odpoved/mn. zaznamu. Dotazy - na uplnou/castecnou shodu; intervalovou shodu. **vyvazenost struktury** - zajisteni omezeni cesty pri vyhledavani nejakym vyrazem ($O(\log N)$ atp.), rovnomernost naplneni bloku – **faktor naplneni stranek**. Splneni se dosahuje stepenim a slevanim bloku; SOS ktere splnuje - **dynamicke**; ost. - **staticke**.

fyzicka media - hl. mag. pasky, HDD. Pasky - sekvencni pristup, bloky pri vyhledavani cteny a kontrolovany sekvencne. Pro kapacitu dulezita hustota, vhodne setrideni dat podle klice. Nutne meziblokove mezery. HDD - primy pristup. Hlavy, "valce", stopy. Vzdy je mozne cist jen 1 hlavou. Vets. rozdeleny na sektory. Pro rychlost: **seek** (presun na jiny valec), **rotate** (doba 1 pulotacky), **block transfer time** - propustnost sbernice. Nove disky - ruzny pocet sektoru na 1 stopu, slozity vypocet cylindru a hlavy z cisla bloku ⇒ dela elektronika, udavana adresa cylindr-hlava-sektor nemusi mit nic spolecneho se skutecnosti.

1 stopa = 512 K. Nacteni 1 MB: min. $s + r$ pro nalezeni 1. sektoru $+2 \times 2r$ za nacteni stop. Pri cteni z nah. rozmistenych 4 K bloku: $256 \times (s + r + btt)$ - tj. az 100x pomalejsi.

Staticke metody org. souboru

Casovy odhad doby pristupu k zaznamu T_F - slozitejsi: $s+r+btt$ obecne, $r+btt$ pro dalsi ve stejnem cylindru, $T_{RW} = 2r$ REWRITE - přepis (behem 1 otacky disku). Sekvencni cteni - $t' = R \cdot t/(R + W)$, kde W jsou meziblok. mezery, t je transfer rate. Dalsi casy: T_I, T_D T_U, T_Y (reorg.), T_X (read entire file)

Hromada- jenom fyzicky za sebe naplacane zaznamy. Slozitost nalezeni $O(N)$.

Sekvencni soubor

neusporadany - to same co hromada, **usporadany** - zaznamy razeny podle klice. aktualizovane zaznamy se umistuji do zvl. neusp. souboru, REORGANIZATION znova setridi. Nalezeni opet $O(N)$ nebo $O(N/\lfloor b \rfloor)$. U medii s primym pristupem ale $O(\log_2 N)$.

Index-sekvencni soubor

Prim. soubor - sekvencni soubor setrideny podle prim. klice & nad nim (i viceurovnovy) index: cislo bloku & min. hodnota klice v nem; pro vyssi uroven to same ale na blocich indexu nizsi urovne. Nejvyssi uroven - obvykle 1 blok (**master**). Pocet urovni se da spocitat : $x = \lceil \log_p N/\lfloor b \rfloor \rceil$, kde $p = \lfloor B/(V + P) \rfloor$ a V je velikost klice a P velikost pointeru na blok/zaznam. Zarazeni noveho zaznamu - problemy: pridavani do oblasti preteceni & v ni retezeni zaznamu za sebe (∀ ma pointer na dalsi zaznam, nejenom blok, v obl. preteceni). Pro oddaleni nutnosti vkladat do oblasti preteceni lze bloky plnit na min nez 100%.

Indexovany soubor

Prim. soubor + indexy pro ruzne vyhledavaci klice. Indexuji se primo zaznamy, ne jen bloky. Prim. soubor uz nemusi byt setrideny. Index muze byt podobny jako u indexsekvencniho souboru; ale lepsi je, kdyz pro zaznamy se stejnym klicem je ve vsech urovnich az na prvni (nejnizsi) ten klic jen 1x a pak odkazuje na seznam zaznamu. Pro aktualizace se nepredpoklada oblast preteceni, ale zmeny v indexu. $T_F = (1 + x)(s + r + btt)$. Dotazy na kombinovanou castecnou shodu - lze pouzit,

ale narocne; Alternativa: **kombinovany index** pro vice atributu; to ale vyzaduje analyzu na co jsou dotazy nejcastejsi. **clusterovany index** - zaznamy se spolecnou hodnotou 1 atributu jsou blizko sebe (jde jen pro 1 atribut).

bitove mapy - efektivni zpusob indexace pro atributy s malou domenou (max. 100vky) - pro kazdou hodnotu teto domeny vyrobit vektor bitu delky N, kde 1 na i-te pozici indikuje, ze i-ty zaznam ma prave tuto hodnotu atributu. Boolske dotazy potom jako bitove operace nad temito vektory, and, or, not - normalni bitove operace CPU. Vektory bitu lze navic komprimovat - nejsou tak velke, vhodne pro databaze s hodne zaznamy.

indexy v DIS (document information system) - tzv. **invertovane soubory** pro fulltextove hledani - ulozene pro kazde slovo pocet souboru kde se vyskytuje a pointer na soubor souradnic = dvojic dokument, pozice. Dotazy na shodu pro vice slov - mnozinove pruniky (zac. od slova s nejmensim poctem vyskytu v databazi). Ziskani invertovaneho souboru: rozparsovani na slova, setrideni, odstraneni duplicit, vypocet frekvenci slov. **Zipfuv zakon** distribuce slov $f = k/r$, kde k je konstanta, r je poradi cetnosti, f je cetnost. Taky vyuziti komprese; zmenseni - vyhozeni nevyznamnych slov.

Soubor s primym pristupem

zaznamy v primarnim souboru (adresovy prostor) rozptyleny pomoci hash-fce. Casto pouzivane $h = k \mathrm{mod}\ M'$ kde M' je nejblizsi prvocislo mensi nez velikost adr. prostoru. Hash-fce neni prosta, proto vznikaji kolize, reseni: otevrene adresovani (pozice + 1), rehashovani, oblast preteceni. h prosta = perfekni hashovani (ridky vyskyt), jinak se perfektni hashovani oznacuje taky stav, kdy je potreba $O(1)$ pristupu na disk pro nalezeni zaznamu. Implementace - bud hash = cislo stranky; nebo hash = cislo stranky i relativni pozice v ni. Kolize - snaha umistit zaznam pokud mozno do stejne stranky. Ocekavana delka retezce kolizi: $\lambda = M/N$ - pravdepodobnost ze misto je plne, delka $E = (1-\lambda) + 2(1-\lambda)\lambda + k(1-\lambda)\lambda^{k-1}... = (1-\lambda)\sum_{i=0}^{\infty} \lambda^i = (1-\lambda) \cdot 1/(1-\lambda)^2 = 1/(1-\lambda)$.

Trideni

Pro trideni neceho co se nevejde do pameti: **trideni slevanim**. Z 2 souboru vezmu 2 prvky, vyberu mensi z nich, zapisu a ze souboru odkud pochazel nactu dalsi; postupne dostavam delsi setridene kusy. Puvodni pouziti - setrideni kousku co se vesly do pameti & slevani na paskach. Dnes pouzitelne i na HDD. **trideni haldou** - halda je vyvazeny bin./n-arni strom, vhodny pro hledani minima/maxima. Da se uchovavat v poli - zleva zarovnany; potomci uzlu i vzdy na pozicich $n(i-1)+2, ..ni+1$ (pro indexovani od 1). Trideni haldou: vytvori se (max.) halda, pak se z ni vybiraji akt. max. prvky, davaji se na konec pole, to co tam bylo se vklada do haldy. Iterace konci az v halde zbyva jen 1 prvek. Celkem $O(n \log n)$ operaci.

N-cestny algoritmus trideni: Mam-li n+1 stranek v pameti: 1.vytvori heap/quicksortem setridene behy velikosti n stranek. Pak v kazdem kroku sleva (max.) n nejkratsich behu, vysledek uklada jako 1 beh. Pro M stránek v souboru: $O(2M\lceil \log_n M/n \rceil)$ - cele projde $\log_n(M/n)$-krat procesem slevani.

Delsi behy muzu vyrobit pomoci haldy: Prubezne odebiram a vypisuju minimalni prvky a nacitam dalsi, pokud je vetsi nez minimum, hodim ho na haldu, pokud je mensi, dam ho do akt. vznikajici haldy na druhem konci pole. Az se vycerpa halda, zacnu novy beh. V nejhorsim pripade stejne jako trideni v pameti, prumerne 2x lepsi, idealne setridi vsechno.

Staticke hashovaci metody

Cormackovo perfektni hashovani

Soubor + adresar. Krom prim. hashovaci funkce h jsou potreba dalsi hash-fce $h_i(k,r) = (k \mathrm{mod}\ (2i + 100r + 1)) \mathrm{mod}\ r$, kde r je velikost oboru hodnot (pocet kolizi), i je nejaky vhodnyparametr - cislo hash fce (hleda se postupnym zkousenim dokud neni vysledek bez kolizi). V adresari: $\forall$ zaznam ulozeny odkaz do prim. souboru, r a i. V prim. souboru klic a data. Pri hledani zahashuju klic h, dostanu se do adresare, pokud je tam $r = 0$, nenasel jsem. Pokud $r \geq 1$, spocitam podle ulozeneho i hash-fci h_i, vezmu odkaz do prim. souboru, prictu vysledek h_i a v prim. souboru zkontroluju jestli jsem nasel co jsem hledal. Pridavani - spocitam h, pokud tam nic neni, najdu volne misto v prim. souboru delky 1 a vlozim, pokud tam uz neco je, najdu volne misto delky $r+1$, zvetsim r a najdu i, aby kolizni od sebe rozhazela a prehazim je na nove misto. r nemusim zvetsovat o 1, ale rychleji, takze najdu pozadovane i lepe (pokud mam blby h_i, nekdy r proste musim zvetsit). Da se pouzit i pro dynamicky pametovy prostor.

Perf. hashovani Larson-Kalja

Mensi pomocna tabulka: jen **separator** (spec. cislo) pro kazdou stranku. Mam M hash-fci, ktere vytvareji pro zaznam posloupnost stranek do kterych ho zkousim vkladat, navic 2. sadu jim jednozn. odpovidajicich fci, ktere pocitaji **signaturu**. $\forall$ stranka ma separator, pokud je separator $<$ signatura zaznamu, OK, jinak tam zaznam nemuzu vlozit. Po preplneni vyhodim zaznamy s nejvetsi signaturou a zmensim separator. Prvek se nemusi podarit vlozit (kdyz mi dojdou obe sady funkci). Pokud vkladam do plne stranky, nemuzu vlozit i kdyz mam mensi signaturu nez je separator - separator pak musim zmensit a ze stranky vyhodit i neco dalsiho - kaskadovani insertu. Vhodne pro malo vkladani a hodne cteni - jen 1 pristup na disk (maly adresar se do pameti vejde).

Dotazy na castecnou shodu

Predp. ze zaznam ma n atributu, vezmu n hash-fci, kazda generuje d_i-bitovy vysledek (signaturu atributu). Pro adresovy prostor $M = 2^d$ stránek $\sum d_i = d$ (-zretezeni: signatura zaznamu). Castecna shoda - dotaz kde bity nekterych atributu jsou nahrazeny "?" - "s-bitova signatura" pro s bitu zadanych - takovy dotaz je $2^{d-s}\times$ drazsi. Prumerna cena dotazu je pro pravdepodobnost P_i dotazu na atribut i :

$$\sum_{q \in \mathcal{P}(\{1,..n\})} (P_q \cdot \Pi_{i \notin q} 2^{d_i})$$

. Idealni rozvrzeni d_i: $d_i = (d - \sum_{j=1}^{n} \log_2 P_j)/n + \log_2 P_i$. Extremy upravit na 0, d; Pokud je 2^{d_i} vetsi nez domena A_i, upravit na $A_i = \lceil \log_2 |A_i| \rceil$ a prepocitat ostatni.

Urychleni dotazu - **deskriptory stranek**- deskriptor zaznamu: $w = w_1 + .. + w_n$ -bitovy retezec: pro kazdy atribut $A_1, ..A_n$ ex. fce T_i, ktera kazde jeho hodnote priradi w_i-bitovy retezec s prave 1 jednickou (muze byt zadano jinak, ex. vzorce pro idealni w_i i pocet jednicek k nim). Deskriptor stranek je "||" deskriptoru vsech zaznamu ve strance. Pri hledani vyrobim deskriptor dotazu, kde jsou "?", tam dam same nuly, pokud ma dotaz nekde v deskriptoru "1" a stranka "0", nemusim tam hledat. Lze udelat i 2urovnove - deskriptory segmentu.

Dalsi moznost - **Grayovy kody** - snazi se resit nesouvislost oblasti stranek se zaznamy vyhovujicimi dotazu (pr. 10???1010). Jde o jiny zpusob kodovani binarnich cisel - 2 po sobe jdouci cisla se lisi vzdy jen v 1 bitu. Max. zisk - o 50% mene shluku. Nikdy neda horsi vysledky nez binarni. Konverze na Graye: $G[n] = B[n]$, $G[i] = B[i+1]\text{xor } B[i]$.

Dynamicke hashovani

Rozsiritelne hashovani - Fagin

Pomocna struktura - adresar s pointery na stranky (nektere mohou byt v adresari vickrat), hash. fce s rovnomernym rozdelenim. Vzdy pouziju jen prvnich d bitu hash-fce, minimum kolik potrebuju. Podle nich urcim zaznam v adresari, z nej najdu stranku. Pro stranku se muze pouzivat min bitu, proto na ni muze ukazovat vic zaznamu v adresari. Kdyz se stranka naplni, rozstepim ji na 2 podle dalsiho bitu hash-fce. Pokud uz stranka pouziva stejne bitu jako adresar, musim zvetsit o 1 bit i adresar (a zdvojnasobit jeho velikost), s ostatnimi strankami nic nedelam. Pri vypousteni zaznamu lze stranky slevat, pokud si pamatuju jejich vyuziti (muzu slevat jen stranky ktere se lisi jen v poslednim bitu).

Linearni hashovani - Litwin

Nema adresar, ma oblast preteceni. Data vkladana do prim. stranek, po kazdych L vlozeni se vynucene stepi urcena stranka (v poradi podle cisel stranek $0, 1, 00, 01, 10, 11, 000$). Preteceni pristupne pointry z prim. stranek. Podle hashe (jeho konec musi odpovida cislu stranky) se rozdeluji pri stepeni prvky. Hledani: kdyz mam aktualne n stranek, vezmu poslednich $\lfloor \log_2 n \rfloor + 1$ bitu, pokud je to $> n$, zanedbam horni bit, mam cislo stranky. Kdyz tam neni a stranka je pretecena, podivam se jeste do obl. preteceni. Pri stepeni je nutne vratit do rozstepenych stranek i zaznamy z obl. preteceni. Problem: vice zaplnene stranky na konci, tedy jeste nerozstepene. Reseni: bud nerovnomerne rozdeleni hash-fce, nebo skupinove stepeni stranek.

Skupinove stepeni stranek

Rozsireni Litwina pro lepsi vyuziti stranek; stranky vzdy v s skupinach po g, mam inicializacni hash-fci $hash$ do $0..(g \cdot s_0) - 1$. s_0 se casem zvetsuje, g je konstanta. Stepim take po L vlozeni, vzdy g stranek do $g+1$, na to mam nezavisle hash-fce $h_0, ...h_1$ do $0..g$. Az dostanu s skupin po $g+1$ strankach, preorganizuju stranky zpatky do skupin po g (muzu pridat nejake prazdne aby to vychazelo). Pri hledani

- pocitaji se vsechna prehashovani uplne od zacatku - docela HW narocne, ale proti rychlosti disku se vyplati.

Spiralova pamet

Podobne lin. hashovani, ale pro exponencialni rozdeleni klicu; misto rozdeleni 1 stranky na starou a novou tu starou zahodi a prida 2 nove. Klice nejprve rovnomerne rozdeleny hash-fci $G(c,k) :< c, c+1 >$ ($c \in \mathbf{R}_0^+$) , pak prepocteny do $< b^c, b^{c+1} >$ a zaokrouhleny na konkretni cisla stranek ($\lfloor b^G \rfloor$). Pri zmene c se meni velikost prostoru. Pri expanzi se nove c voli tak, aby znicilo stranku ktera se stepi (nejnizsi cislo) - $c_{n+1} := log_b(f+1)$ (kde f je cislo 1. stranky). Aby se mi neposouvaly vsechny stranky porad dal od 0, prvni zahozene strance dam nove cislo a znova ji pouziju – prepocitavani log. stranek na fyzicke se pak dela rekurzivne (if $\lfloor$log. stranka/$b\rfloor$ < $\lfloor$(1+log. stranka)/$b\rfloor$ vezmi rekurzivne fyz. stranka := fyz. stranka($\lfloor$log.stranka/$b\rfloor$), else vrat fyz. stranka = $\lfloor$log. stranka/$b\rfloor$.)

Hash. fce zachovavajici usporadani

Kvuli urychleni intervalovych dotazu - hashovaci metody, zachovavajici usporadani klicu. **lin. hashovani pro intervalove dotazy** Vychazi z Litwina, vyuziva nejvyznamejsi bity k urceni stranky. **castecne linearni hashovani zachovavajici usporadani** - vychazi ze znalosti rozdeleni klicu, deli domenu klicu na nestejne dlouhe intervaly aby vyvazovalo nerovnomernost rozdeleni. Nelze ale porad reorganizovat, takze se upravuji jenom prilehle intervaly pri vkladani a odebirani. Nehodi se pro dynamicky rostouci domenu klicu. Muze byt provedeno viceurovnove.

Stromy

B-Stromy

B-strom radu m: koren ma min. 2 potomky, pokud neni sam listem; kazdy uzel ma $\lceil m/2 \rceil$ - m potomku a vzdy o 1 mene dat. zaznamu, pro podstrom $U(p_i)$: $\forall k \in U(p_i) : k > k_i$, $\forall k \in U(p_{i-1}) : k \leq k_i$; vsechny **vetve** (cesty od korene k listu) jsou stejne dlouhe. Nekdy se data presouvaji az do listu (**redundantni B-stromy**); nekdy jsou z uzlu na data jen pointery, listy muzou mit jinou (jednodussi) dat. strukturu. Pro implementaci vhodne zapamatovani cele aktualne prochazene vetve v nejakem bufferu.

Vyhledavani - jednoduchy pruchod do hloubky podle klicu. Insert - pokud se zaplni stranka, rozstepi se na 2 & pointery na ne se daji do nadrazeneho uzlu, pokud se nevejdou, stepi se dal, treba az do korene. Delete - opacne; v pripade podteceni stranky zkousim pujcit data od sousedu/slevat. V redundantnich stromech nemusim pri odstraneni dat odstranovat klic ve vnitrnich uzlech (lze podle toho hledat i kdyz to tam neni). Vylepseni - vyvazovani stranek - pri preteceni stranky se nejdriv divam jestli neni volno v sousednich, pokud ano, prerozdelim a upravim klice - zarucuje lepsi zaplneni, ale je pomalejsi. Podobne je mozne vyvazovat pocty se sousedy v pripade vyhazovani (i kdyz nemerguju).

B* stromy Vylepseni, na zakl. vyvazovani stranek. Koren ma min. 2 potomky, ost. uzly minimalne $\lceil (2m-1)/3 \rceil$ potomku, vsechny vetve jsou stejne dlouhe. Stepeni - odkladane dokud nejsou sourozenci plni, potom se stepi bud 2 do 3 (jen s 1 sourozencem), nebo 3 do 4 (oba) uzlu. Odebirani - podobne slevani 3 do 2 (4 do 3) – stepeni a slevani jde zeslozitit jeste na vic stranek.

odlozene stepeni - pouziva stranku preteceni, vklada znova az kdyz ta se naplni. Stranka preteceni muze byt 1 pro 1 listovy uzel, nebo ji muze sdilet nejaka skupina listu – stepi se az kdyz jsou vsechny listy i preteceni zaplnene ⇒ pokud ma strom vic nez 1 uroven, ma vsechny listy zaplnene (za predpokladu nepouziti delete). S odebiranim - musim i slevat & stepit skupiny - jejich velikost neni pevna.

prefixove stromy - pro redundantni stromy; klice jsou co nejkratsi retezce nutne k odliseni listu, nikoliv cele hodnoty, ktere se nachazeji az v listech. Pri vkladani a stepeni stranek se nejakou heuristikou hleda nejkratsi prefix, ktery by 2 vznikajici stranky oddelil. Mazani a slevani - zadna zmena. Dalsi zkraceni - u potomku se neopakuje predpona klice, kterou ma rodic – to ale hodne zvysi naroky na CPU

B+ stromy - pro intervalove dotazy - zrychleni - zretezeni vzdy uzlu v 1 urovni (a nebo jenom listu). Jeste lepsi - zretezit obema smery.

promenna delka zaznamu - modifikace pro zaznamy ruzne delky: nestepit podle poctu zaznamu, ale na zhruba 1/2 podle velikosti. Podminka existence uzlu: soucet delek zaznamu v nem je $\geq B/2$ kde B je delka uzlu(stránky) (pro B* stromy $2B/3$). Problemy: dlouhe klice maji tendenci propadavat ke koreni, tim se zmensuje arita stromu; muze se 1 stranka stepit i na 3 (pokud vkladam zaznam delsi nez 1/2 stranky); vlozenim zaznamu muze dojit ke zmenseni stromu (jak, to se ve skriptech nepise :( ) (- nejde

vyrobit nezavisle insert a delete, reseni: univerzalni alg. nahrazovani retezce retezcem, insert&delete jsou jeho spec. prip.). Reseni snizovani arity stromu: minimalizace delky klicu (nalezeni klice min. delky, ktera navic splnuje min. naplneni) - pro B* stromy docela slozite.

Vicerozmerne dat. struktury

Vicerozmerne B-stromy

Mam pole atributu, z nej odkazy na seznamy stromu pro jednotl. atributy. Pro prvni atribut mam jen 1 strom. Z kazdeho uzlu toho stromu pro kazdy klic je odkaz na cely strom 2. atributu pro dany klic (podobne pro 2→3 a dalsi atributy). Stromy stejneho atributu jsou ve spojaku, zacinajicim v poli atributu. Muzu hledat vsechny zaznamy pro ktere znam klice vsech atributu nebo jejich podmnozinu (slozitejsi, nutne projit vic stromu). Je ale jednodussi oproti kombinovanemu indexu (netreba mnozinovych operaci).

Vicerozmerna mrizka

Bere mnozinu moznych atributu jako vicerozmerny prostor, ktery deli na hyperkrychle odpovidajici strankam na disku. Nad kazdym rozmerem je vyhledavaci strom s delicimi hodnotami. Jednotlive hyperkrychle (kapsy) muzu podle potreby rozdelovat a skladat dohromady (napr. heuristikou: stridani os). Funguje docela efektivne.