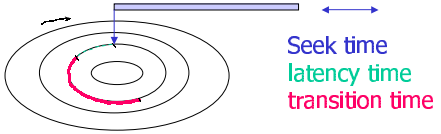
Fyzická organizácia dát

- Organizácia pamätí (vnútorná a vonkajšie pamäte)
  - rotačné, diskové a bubnové pamäte
  - sekvenčné (páskové) pamäte
  - alternatívne pamäte
- Čas prístupu
  - seek time (čas vyhľadania cylindra)
  - latency time (čas pretočenia na začiatok na stope)
  - transmission time (čas prenosu dát)
- Formát blokov a záznamov
  - pevná alebo premenná dĺžka
  - pripichnuté (pinned), nepripichnuté alebo polpripichnuté záznamy

Diskové pamäť



- Čas prístupu
  - seek time (čas vyhľadania cylindra) *cca 10 ms*
  - latency time (čas pretočenia na začiatok na stope) *0.1 ms*
  - transmission time (čas prenosu dát) *cca* $5.10^{-7} \times b$ *s*

Sekvenčné média (pásky)

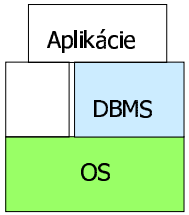
Rewind, locate, read (forward, backward)

Organizácia súborov

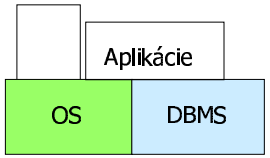
- Operačný systém
- Databázový systém

Možné architektúry

Vyrovnávacie pamäte (buffers)
Odsun blokov:
- v poradí
- ktorý je v pamäti najdlhšie
- s ktorým sa najdlhšie nepracovalo



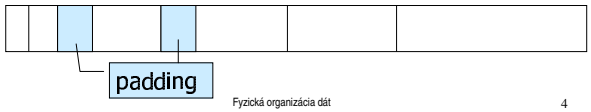
a



b

Údajová štruktúra záznamu

- Stavová informácia
  - used / unused bit
  - deleted bit
- Formát (typ) záznamu
  - informácia o štruktúre záznamu
  - informácia o dĺžke a pozícii jednotlivých položiek (priamo alebo prostredníctvom oddelovačov)
- Výplň (padding, waste)
- Vlastná uložená informácia



Údajová štruktúra bloku

- Hlavička bloku
  - veľkosť bloku, typ bloku
  - smerníky na predošlý a nasledujúci blok
  - počet záznamov, smerníky na jednotlivé záznamy
  - údaj o volných pozíciách a vynechaných záznamoch
- Uložené záznamy
- Volné miesto

Zjednodušenie:

Bloky a záznamy pevnej dĺžky.

Smerníky a ich implementácia

- Interné smerníky
  - radenie blokov súborov
  - smerníky na položky záznamov v záznamoch premennej dĺžky
- Aplikačné smerníky
  - **Absolútne smerníky** - pripichnuté záznamy
  - **Kľúče** - volne pohyblivé záznamy. Musí však byť implementovaná vyhľadávacia štruktúra. Nájdenie záznamu môže vyžadovať viac prístupov k bloku.
  - **Dvojica (b, k)**, kde b je adresa bloku a k je kľúč. Po prinesení bloku hľadanie sa už deje v operačnej pamäti záznamy sa môžu pohybovať v rámci bloku. Polopripichnuté (semi-pinned) adresa pozostáva z dvojice (b, a), kde a je pozícia v adresári bloku.

Pridelovanie pamäti pre bloky

- First fit
- Best fit
- Worst fit (next free in cyclic address space)

Schéma organizácie súborov

- Vyhľadanie záznamu (Search, Look-Up)
- Vloženie záznamu (Insert)
- Vynechanie záznamu (Delete)
- Modifikácia záznamu (Update, modification)
- Vytvorenie súboru (Create), Ukončenie práce so súborom (close)

Organizácia súborov

- Sekvenčné súbory a haldy
- Indexovo - sekvenčné súbory
- Stromovo organizované súbory
- Hašované súbory

Halda (heap) : triviálna schéma absencia organizácie. Bloky tvoria obojsmerne spájaný zoznam. Záznamy sú uložene za sebou ako prichádzali.

Výhody:
- Žiadne náklady na správu
- Ľahké vkladanie a vynechanie

Nevýhody:
- Sekvenčné vyhľadávanie podľa kľúča

Kľúčovosekvenčnésúbory

Kľúčovo sekvenčné súbory sú podobné haldám, odlišujú sa však tým, že súbor sa udržuje utriedený podľa primárneho kľúča.

Problémom je vkladanie vyžaduje odsunutie viet od danej pozície. To je neúnosné. Riešenie **bloky pretečenia** (overflow blocks). Prístup k blokom prerušenia:

- Nepriamo cez primárny súbor
- Prístup s posunom cez smerník v bloku

Zhodnotenie: V porovnaní s predošlou organizáciou za minimálne náklady sa získala možnosť vyhľadávania bisekciou alebo interpoláciou.

Indexovo-sekvenčnésúbory

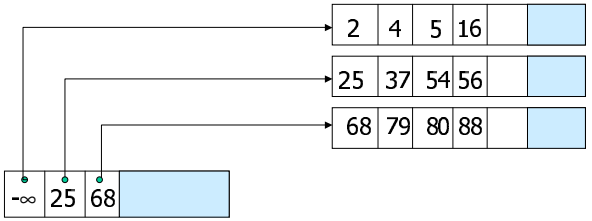
K základnému kľúčovo sekvenčnému súboru sa pridáva pomocná štruktúra index. (**ISAM** - index sequential access method). Index je súbor, ktorého vety postávajú z dvojíc (*kľúč, smerník*). Vlastne stačí začiatok hodnoty kľúča a smerník na blok, v ktorom sú vety s danou hodnotou kľúča. Vyhľadávanie pozostáva z vyhľadávania v indexe a v primárnom súbore (vlastne v bloku a blokoch preplnenia).

Ak je súbor príliš veľký možno zaviesť index na indexový súbor. Indexy vyššej úrovne. **(HISAM** - hierarchical index sequential access method).

Hardwarovo závislé indexovanie - úrovne: bloky, sektory, stopy, cylindre, disky.

Pri narastaní súboru pribúda blokov preplnenia. Efektívnosť metódy klesá. **Periodické reorganizácie**.

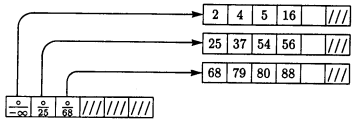
Príklad indexovo-sekvenčnéhosúboru



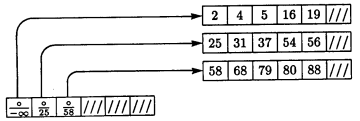
Insert{ 19, 58, 31, 52}

- Pinned versus unpinned records
- Sorted files with pinned records

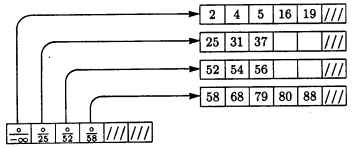
Pôvodná štruktúra index sekvenčného súboru.

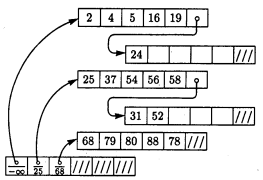


Po inzercií 19, 58 a 31.

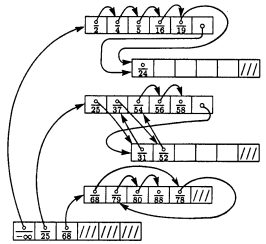


Po následnom vložení kľúča s hodnotou 52.





Vkladanie do súboru s pripichnutými záznamami.



Smerníky umožňujúce udržovanie usporiadania v indexovo sekvenčnom súbore s pripichnutými záznamami.

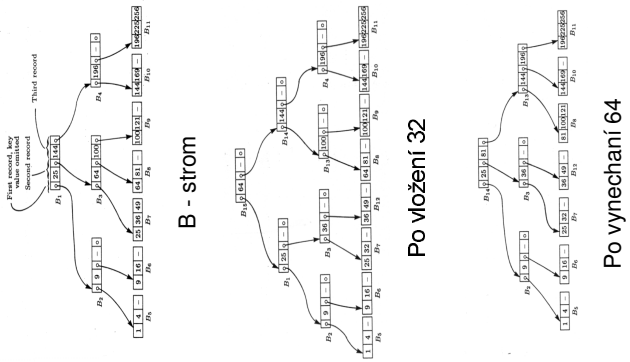
Stromovésúbory:B -stromy

Vyhľadávacie stromy (AVL - stromy, 2 - 3 stromy) nie sú pre viacúrovňovú pamäť celkom vhodné lebo každý uzol obsahuje iba jeden záznam.

B(m) - strom:

- Každý uzol okrem koreňa obsahuje *k* záznamov. Kde *[m/2]* ≤ *k* ≤ *m*. ([2m/3] ≤ *k* ≤ *m*)
- Koreň ak nie je listom obsahuje aspoň *1* a najviac *m záznamov.*
- Vnútorný uzol s *k* záznam ma *k+1* synov.
- Všetky listy sú na tej istej úrovni.

B* stromy majú vo vnútorných uzloch kľúče (index) a záznamy iba v listoch. Je možné aj rôzne ohraničenie na vnútorný uzol a list.



Stratégieindexovania

- Hustý (dense) index: Indexy ukazujú na všetky záznamy.
- Riedky (sparse) index: Indexy ukazujú len na niektoré záznamy. Ostatné hľadáme sekvenčne od najbližšieho predošlého záznamu.

Vo všetkých predošlých prípadoch išlo riedky index.
B stromy vyžadujú nepripichnuté záznamy.

Ak používame sekundárne indexy vzniká potreba hustého indexu a pripichnutých záznamov.

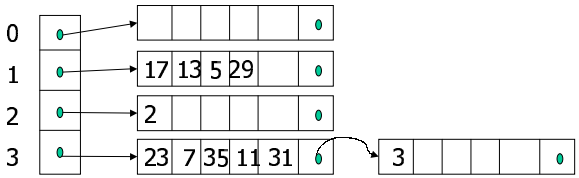
Záznamy možno zmeniť na nepripichnuté ak sekundárny index zostrojíme pre primárne kľúče. Invertovaný súbor obsahuje len záznamy tvaru dvojíc:
<index, primárny kľúč záznamu v pôvodnom súbore>.
Redukuje sa tým významne potreba reorganizácie.

Hašovanie - transformáciak ľúča

Vytvorenie čísla z kľúča:
Nech $m = log_2 M$, kde je M počet priečinkov (buckets).

- Polámanie na úseky dĺžky m ich XOR alebo súčet *mod M.*
- Nejakých m bitov zo stredu mocniny
- Kľúč k berieme ako postupnosť bitov $h(k) = k \bmod M$.
- $h(k) = [M.(k.A \bmod 1)]$, kde $0 < A < 1$. (Fibonacciho transformačná funkcia $A = (\sqrt{5} - 1)/2$.)
- Polynomiálna hašovacia funkcia
- $h(k) = [(M*(k-a)/(b-a)]$, kde $a \le k < b$. „Hašovacia" funkcia zachovávajúca usporiadanie.

Štruktúra hašovaného súboru



adresár základné bloky bloky preplnenia

Technické problémy
- prázdne buckety
- pripichnuté záznamy
- vynechanie

Faktor naplnenia: $\alpha = N/M$
Ak $\alpha < b$ a adresár je v hlavnej pamäti očakávaný prístup k jednému dvom blokom.

Pokročilé dátové štruktúry a ich analýzy

- AVL stromy
- Analýza hašovania
- k– d stromy
- Reorganizácia - princíp dynamizácie
- Dynamizácia hašovania
  - dynamické hašovanie
  - rozšíriteľné hašovanie
  - lineárne hašovanie
- Podpora dotazov
  - čiastočná zhoda
  - intervalová zhoda
  - relačné operácie

AVL(1)stromy (Adelson, Velskij, Landis)

AVL(1) stromy sú binárne vyhľadávacie stromy, v ktorých výška ľavého podstromu a pravého podstromu v každom uzle sa líši najviac o jedna. *(Výška = najdlhšia cesta od koreňa k listu.)*

Úplný binárny strom o výške h má 2^h listov a $2^{h+1}-1$ uzlov.

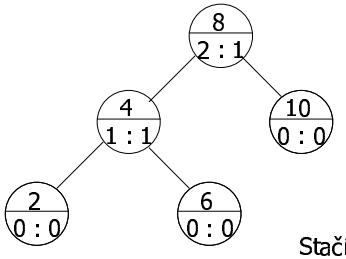
Pre výšku h AVL(1) stromu s n uzlami platí:
$lg(n+1) \le h < 1.4404\, lg(n+2) - 0.328$

Konvencia:
lg = $\log_2$
ln = $\log_e$

Dôkaz: Najmenšiu výšku dosiahneme ak AVL strom je úplný binárny strom. Označme $|T|$ počet uzlov stromu T. Aby AVL strom T_h výšky h mal minimálny počet uzlov, musí v každom platiť rekurentná rovnica: $|T_h| = |T_{h-1}| + |T_{h-2}| + 1$.

Jej riešením je $|T_h| = F_{h+2} - 1 > \frac{1}{\sqrt{5}}\phi^{h+2} - 2$, kde $\phi = \frac{1}{2}(1+\sqrt{5})$

NarábaniesAVL - stromami



Vyváženie sa poruší vložením uzlov 1, 3, 5, 7. Iné vloženia vyváženie napravia.

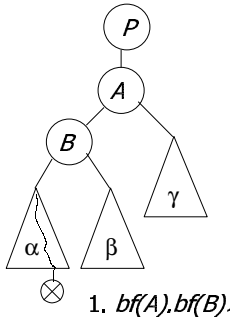
Faktor balansovania
$bf = h_R - h_L$; $bf \in \{-1, 0, 1\}$

Stačí v každom uzle pamätať *bf*.

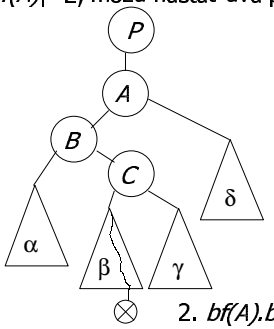
Vkladanie najprv realizujeme bez ohľadu na balansovanie. Pri hľadaní vhodného miesta si zapamätáme do premennej *a* smerník na posledný uzol s nenulovým *bf* alebo koreň, ak uzol s nenulovým *bf* neexistuje.

Dokončenievloženia.

Spätne od vloženého listu po uzol *A*, na ktorý ukazuje *a* aktualizujeme *bf*. Ak v tomto uzle bf nadobudol prípustnú hodnotu skončíme. Ak $|bf(A)|=2$, môžu nastať dva prípady:

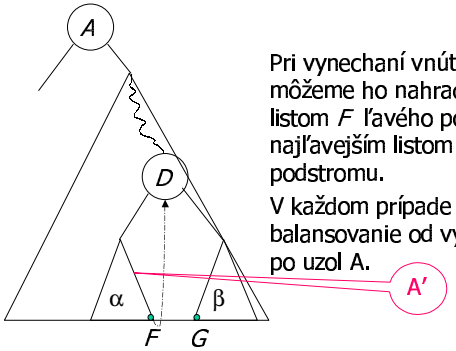


1. $bf(A).bf(B)>0$



2. $bf(A).bf(B)<0$

VynechaniezAVLstromu



Pri vynechaní vnútorného uzla *D* môžeme ho nahradiť najpravejším listom *F* ľavého podstromu alebo najľavejším listom *G* pravého podstromu.

V každom prípade stačí opraviť balansovanie od vynechaného listu po uzol A.

Analýzahašovania

Predpokladajme, že hašujeme do blokov veľkosti 1. Budeme počítať pravdepodobnosť nájdenia voľného miesta na i-tý pokus (vkladanie, neúspešné vyhľadanie).

$$p_1 = \frac{M-n}{M}$$

$$p_2 = \frac{n}{M}\frac{M-n}{M-1}$$

$$p_3 = \frac{n}{M}\frac{n-1}{M-1}\frac{M-n}{M-2}$$

.
.
.

$$p_i = \frac{n}{M}\frac{n-1}{M-1}\frac{n-2}{M-2}\cdots\frac{n-i+2}{M-i+2}\frac{M-n}{M-i+1}$$

$$E_{n+1} = \sum_{i=1}^{n+1} i\, p_i = \frac{M+1}{M-n+1}$$

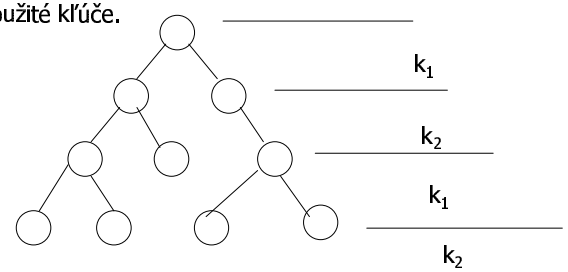
$$E = \frac{1}{N}\sum_{n=1}^{N} E_n = \frac{M+1}{N}\sum_{n=1}^{N}\frac{1}{M-n+2} = \frac{M+1}{N}\left(H_{M+1} - H_{M-N+1}\right)$$

$$\alpha = \frac{N}{M+1} \Rightarrow E = -\frac{1}{\alpha}\ln(1-\alpha)$$

H_n je *n*-té harmonické číslo a platí $H_n \approx \ln(n) + \gamma$

Mnohocestnévyh ľadávaciestromy — k-dstromy

Pre vyhľadávanie podľa viacerých kľúčov môžeme uvažovať, že pri vyhľadávaní striedame cyklicky všetky použité kľúče.



Reorganizáciaa dynamizácia

Uvažujeme, že dátová štruktúra veľkosti n sa dá vytvoriť v čase $t(n) \leq O(n \log n)$. Operácie v nej sa dajú vykonávať v čase $O(\log n)$ a počas jej životnosti sa vykoná $O(n)$ operácii. Potom ju treba reorganizovať.

Ak reorganizácia zväčší štruktúru na veľkosť qn. Potom si pri $q>1$ štruktúra zachová asymptotickú efektívnosť pri periodických reorganizáciách.

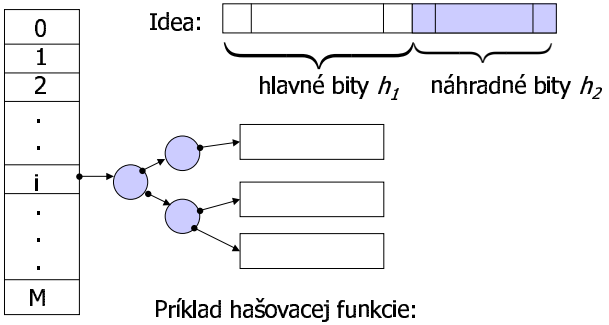
$$\lim_{n\to\infty} \frac{\sum_{i=0}^{n} q^{i} \log(q^{i})}{nq^{n}} = konst$$

Amortizovaná cena operácií

$$\sum_{i=0}^{n} i q^{i} = q \sum_{i=0}^{n} i q^{i-1} = q \frac{\partial}{\partial q}\left(\frac{q^{n}-1}{q-1}\right) = \frac{(n-1)\left(q^{n+1}-q^{n}\right)}{(q-1)^{2}}$$

$$\lim_{n\to\infty} \frac{(n-1)\left(q^{n+1}-q^{n}\right)\log q}{n(q-1)^{2}q^{n}} = \lim_{n\to\infty} \frac{q^{n}(n-1)}{nq^{n}(q-1)} = \lim_{n\to\infty} \frac{(n-1)\log q}{n(q-1)} = \frac{\log q}{q-1}$$

Dynamickéhašovanie (P.A. Larson)



Idea:

hlavné bity h_1 náhradné bity h_2

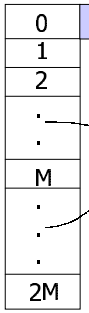
Príklad hašovacej funkcie:

$h(K) = K \bmod p;$ $h_1(K) = h(k) \text{ div } 2^q$

$h_2(K) = h(k) \bmod 2^q$

Rozšíriteľnéhašovanie (Fagin, Pipenger, Nievergelt a Strong)

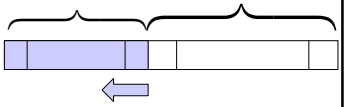
Čo keby sme zmenili poradie hlavných a náhradných bitov?



a $a \geq d$

a - hĺbka adresára

d - úroveň bloku



Počet záznamov v dátovom bloku b.

Hysteréza

Činnosť pri preplnení:

if $a = d$ **then** zdvojnásobiť adresár;

rozdel blok;

Ak počet záznamov v asociovaných blokoch poklesne pod únosnú mieru môžeme bloky opäť spojiť .

Lineárnehašovanie (W. Litwin)

Stále priveľa réžie s adresárom! Stačí jeden bit – existuje blok, neexistuje blok.

Nekonečná trieda hašovacích funkcií:

$$h(q, K) = K \bmod (2^q \times p)$$

Platí:

$$K \bmod 2n = \begin{cases} K \bmod n \\ (K \bmod n) + n \end{cases}$$

Ak blok neexistuje automaticky hľadáme na adrese o *n* menšej.

Ak uveríme sile štatistiky pri preplnení budeme rozdeľovať bloky v poradí. Potom si stačí pamätať len adresu posledného bloku. V takomto prípade treba však znovu ošetrovať preplnenia napr. reťazením v nezaplnených blokoch.

Dotazynačiastočnúzhodu

Problém: Záznamy obsahujú *n* položiek. Ako najlepšie nájsť všetky vety z danými hodnotami *k* položiek.

- indexy (riedky a hustý prienik)
- k-d stromy
- mnohorozmerné hašovanie

	0	1
0	(0,4) (4,6)	(4,3) (6,1) (6,7)
1	(5,0) (5,6) (7,2)	(1,1)

$h_1 = k_1 \bmod 2$
$h_2 = k_2 \bmod 2$

Zložitosť: $M^{1-k/n}$

Výhodné je použiť distribučné hašovacie funkcie.

Dôležité je udržovať hašovací (adresný) priestor približne v tvare n-rozmernej kocky.

Aj mnohorozmerné hašovanie sa dá dynamizovať.

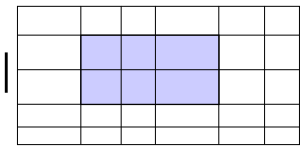
Intervalovázhoda - Rangequeries

Potrebujeme hašovaciu funkciu zachovájúcu usporiadanie.

Predpokladajme: $K \in \langle a, b)$ a $h(K) = \left\lfloor \frac{(K-a).M}{b-a} \right\rfloor$

Táto funkcia je dobrá, ale nemusí rozdeľovať, záznamy do blokov rovnomerne. Ak poznáme kumulatívnu distribúciu Φ hodnôt kľúča na intervale $\langle a, b)$ môžeme vylepšiť hašovaciu funciu:

$$h(K) = \frac{n.\Phi(K)}{M}$$



Existujú aj stromy pre podporu intervalových dotazov (Willard).

V grafických dátach quad-tress a oct-trees.

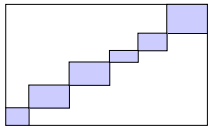
Relačnéoperácie

Zjednotenie rozdiel, projekcia: jednoduché, odstránenie opakovaných záznamov vyžaduje triedenie resp. je tak zložité ako triedenie.

Možnosti spájania dátových štruktúr (merge-able trees).

Prirodzené spojenie, kartézsky súčin:

- Vložené cykly
- Utridenie a zlučovanie (merge join)
- hašovaním (hash join)



Pri hašovaní podľa spajacích položiek spájať sa môžu iba záznamy s rovnakými hodnotami hašovacej funkcie. Na tie už môžeme použiť vložené cykly.