5. Schémata organizace souborů.

Soubor

Soubor je logická paměťová jednotka, která představuje pojmenovanou kolekci vzájemně souvisejících informací (dat).

Zpravidla je umístěn na vnějším paměťovém médiu, kde se zobrazuje do alokačních bloků (což mohou být fyzické stránky, sektory atd.). Přístup k celým souborům řeší operační systém, vlastní organizace souborů může být buď standardní nebo proprietární (?).

Struktura (organizace) souboru

- Relativně volná – textová data v řádcích, posloupnost bytů, sekvenčně řazené záznamy pevné i proměnné délky, …
- Rigidně formátovaná – záznamy, které se přistupují pomocí vhodných řídících struktur (indexy, haše), …

Struktura souboru (databázový pohled)

Soubor se skládá ze **záznamů**, což jsou kolekce **atributů** charakterizujících nějaký objekt.

Na fyzickém médiu se pak člení do **stránek** (fyzických bloků).

Kolekce záznamů, která je spolu v nějaké relaci pak tvoří **soubor**.

Pro efektivní řešení dotazů nad soubory a modifikaci souborů se soubory opatřují **indexy**.

Hlavní problém optimální struktury souboru

Hlavním cílem je minimalizace přístupů na typicky pomalé zařízení, na kterém je soubor umístěn.

Statické vs. Dynamické soubory

Soubory mohou být buď statické, nebo dynamické, tj. záznamy se mohou přidávat/měnit/mazat. V druhém případě je návrh optimální souborové struktury složitější.

Přístup k záznamům v souboru

- **Čistě sekvenční** – zejména historicky v době páskových zařízení, doba přístupu závisela lineárně na velikosti souboru.
- **Přímý** – souvisí s objevem disků. Přístup k záznamům může realizován např.

1. **Hašováním** – *adresa záznamu na disku = F(klíčová položka).*
2. **Indexy** – Tabulka se záznamy *{index : adresa na disku}*. Indexy se posléze začaly ukládat do stromů. Dnes jsou hojně používané tzv. **B-stromy** s logaritmickou složitostí vyhledávání.

Schémata organizace souborů

Schéma organizace se dá rozdělit na tři úrovně. Cílem je umožnit optimálně řešit operace nad záznamy souboru nezávisle na konkrétní fyzické paměti.

Schéma organizace souborů = způsob, jak jsou zakódovaná data počítačem uložená do paměti

Logické schéma

Logická paměť se strukturou optimalizovanou tak, aby umožňovala efektivní operace nad soubory. Logická paměť se dělí na logické stránky a obsahuje vlastní data (primární soubor) + pomocné struktury jako indexy atd. (sekundární soubor).

Cílem návrhu optimálního logického schéma je používání struktury, která při přidávání, modifikaci nebo ubírání záznamů v souboru minimalizuje operace se stránkami. Ideální stav je 1 požadavek = 1 přístup.

Návrh zahrnuje operace jako přidávání stránek, vztahy mezi stránkami, operace nad stránkami, plnění stránek, stanovení mezí faktoru naplnění atd.

Fyzické schéma souboru – Zobrazuje logické soubory do paměťových jednotek konkrétního použitého typu fyzické paměti.

Implementační schéma – Popisuje rozmístění dat na konkrétním uchovávajícím médiu. Standardně řeší OS nezávisle na aplikacích

Složitost schéma organizace souborů

- **Prostorová** – kolik fyzických stránek zabere zobrazení našich souborů
- **Časová** – jak budou časově náročné operace nad těmito soubory. Počet I/O operací s fyzickými stránkami pro jednotlivé operace s logickými stránkami (počet načítaných fyzických stránek do RAM a počet zapisovaných fyzických stránek do zařízení).

Standardní organizace souborů (statické)

homogenní soubor – hodnoty položek jsou primitivní typy, všechny záznamy jsou jednoho typu
nehomogenní soubor – hodnoty položek jeho záznamů nejsou primitivní typy nebo záznamy nejsou jednoho typu

Hromada

Nehomogenní soubor se záznamy proměnné struktury (délky). Složitost vyhledávání roste lineárně s počtem záznamů. Může být i efektivní co do zabraného místa.

Neuspořádaný sekvenční soubor

Homogenní soubor podobný hromadě. Složitost vyhledávání je stejná, složitost přidávání záznamů je konstantní – záznam se prostě přidá na konec. Vhodné pro aplikace požadující sekvenční zpracování celého souboru. *Průměrný* počet přístupů na disk při vyhledávání N/2.

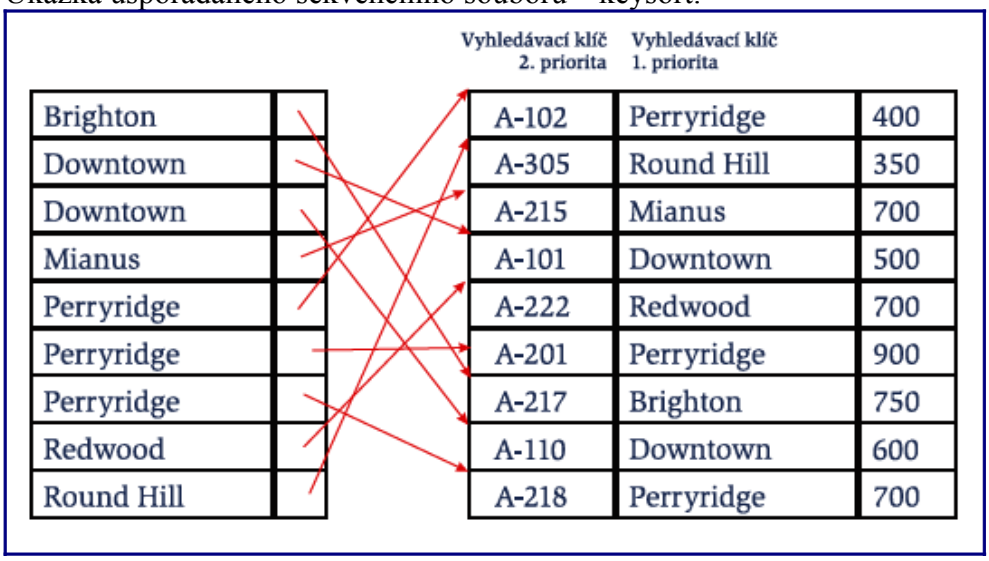
Záznam	Číslo účtu	Pobočka	Stav
0	A-102	Perryridge	400
1	A-305	Round Hill	350
2	A-215	Mianus	700
3	A-101	Downtown	500
4	A-222	Redwood	700
5	A-201	Perryridge	900
6	A-217	Brighton	750
7	A-110	Downtown	600
8	A-218	Perryridge	700

Uspořádaný sekvenční soubor

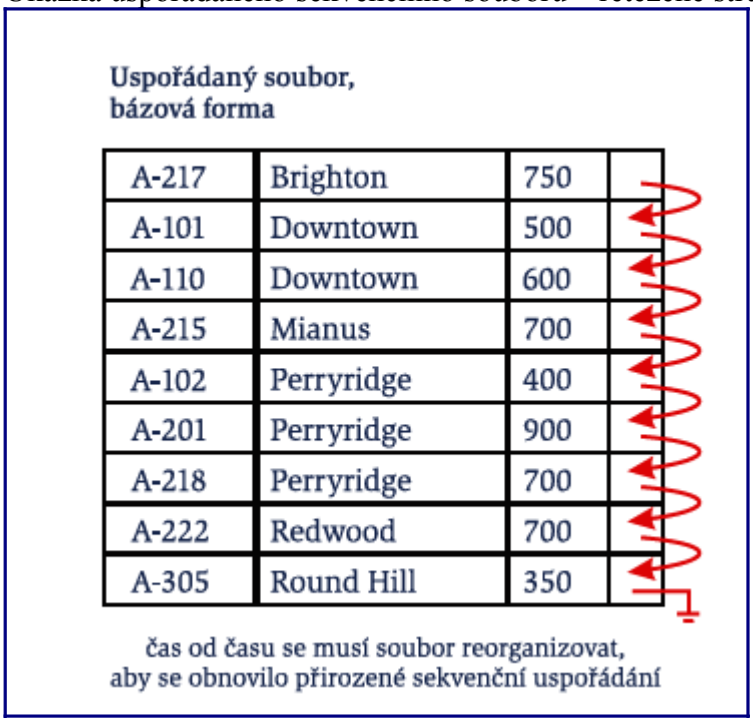
Homogenní soubor uspořádaný podle nějakého vyhledávacího klíče. Data mohou ukládat přímo do souboru a online reorganizovat, nebo se ukládají do neorganizovaného souboru a občas se pak vyvolá setřiďovací operace. **Keysort** – třídí se pouze index, ne primární soubor, vhodné pro velké soubory. **Řetězené struktury** – usnadní vkládání a mazání, jednou za čas se vyvolá reorganizace, aby se obnovilo přirozené sekvenční uspořádání souboru.

Složitost vyhledávání je logaritmická – půlí se interval.

Ukázka uspořádaného sekvenčního souboru – keysort:



Ukázka uspořádaného sekvenčního souboru – řetězené struktury:



Indexované soubory

- třídí se index, primární soubor se záznamy je netříděný

Index-sekvenční soubory

- soubor setříděný podle primárního klíče, ke kterému je vytvořena struktura indexů

- 2 možné přístupy – indexový i sekvenční
- Indexem se vymezuje oblast stránek v primárním souboru, kde záznam může ležet, tato oblast se prohlíží sekvenčně
- buckety pro přesahy, nutná občasná reorganizace

Soubor s přímým přístupem

- algoritmická transformace vyhledávacího klíče na adresu záznamu
- hašování

Indexování v souborech

- procházení přímo záznamů je při operacích velice pomalé – indexy mají řádově menší velikost než samotné záznamy a jejich prohledávání je výrazně rychlejší
- díky menší velikosti je možné indexy držet v operační paměti bez nutnosti přístupu na disk
- často stačí pro vyřešení některých dotazů zpřístupnit pouze malou část záznamů – na základě indexů je možné efektivně provádět filtrování
- forma indexu: { vyhledávací klíč ; ukazatel na záznam }
- typy indexů:
    a. řádné, lineární indexy – v tabulce uspořádané dvojice { vyhledávací klíč ; ukazatel na záznam } podle hodnoty vyhl. klíče
    b. hašované indexy – využití hašovací funkce vyhledávacího klíče
    c. stromové indexy – využití grafové struktury strom
    d. indexové bitové mapy – pozice bitů v bitovém vektoru určují lokality záznamů
- indexy mohou být víceúrovňové – „index do indexu“ – nejnižší úroveň může být přímo primární soubor, výše jsou řídké indexy

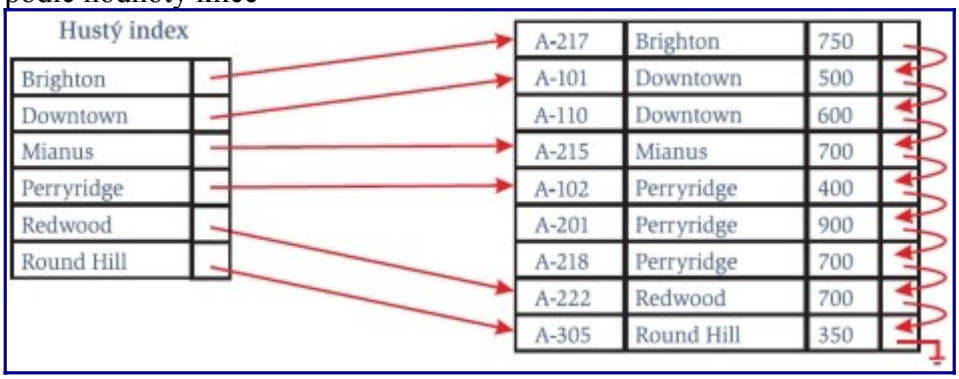
Druhy dělení řádných indexů:

- **Husté indexy** – v indexu je záznam pro každou hodnotu vyhledávacího klíče.
- **Řídký index** – v indexu jsou záznamy pouze pro vybrané klíčové hodnoty. Používá se v případě, že záznamy v souboru jsou setříděné. Případně lze použít kombinaci obou indexů.

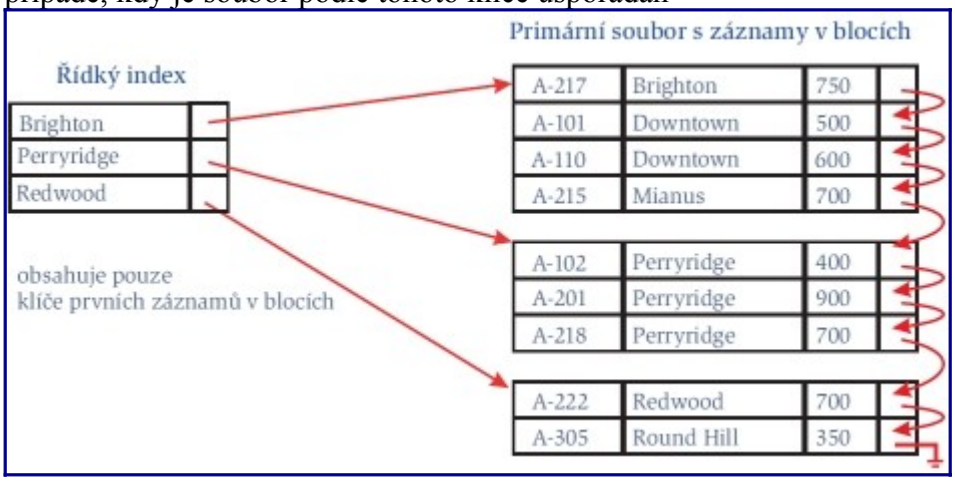
primární index – podle jeho klíče je uspořádán primární soubor se záznamy; může být hustý nebo řídký

sekundární index – určený pro dotazy založené na jiném vyhledávacím klíči než na primárním; musí být hustý

hustý – indexový záznam pro každou hodnotu vyhledávacího klíče. Typicky bývá uspořádán podle hodnoty klíče



řídký – indexový záznam pouze pro některé hodnoty vyhledávacího klíče; použitelné pouze v případě, kdy je soubor podle tohoto klíče uspořádán



Uložení souborů na disku

1. souvislá alokace datových bloků
    - soubor je na disku uložen v souvislé posloupnosti datových bloků
2. alokace datových bloků pomocí zřetězeného seznamu
    - každý datový blok obsahuje data a ukazatel na následující datový blok
3. alokace datových bloků pomocí tabulky
    - alokace datových bloků je založená na zřetězeném seznamu, ale ukazatele na následující datový blok jsou uloženy v tabulce FAT
4. i-nodes
    - i-node je struktura, která obsahuje jak atributy souboru, tak adresy datových bloků, ve kterých je uložen obsah souboru

Dynamické organizace souborů

Motivace: změny dat neznamenají velké (globální) reorganizace, které jsou drahé, ale spíše reorganizace lokální.

1. B-strom
2. dynamický haš soubor
    - založený na rozšiřitelném hašování nebo lineárním hašování

Hašování

hašovací funkce (obecně) = převedení libovolně dlouhého vstupu na výstup pevné délky
hašovací funkce (ve smyslu organizace souborů) = funkce řešící přístup k záznamům souboru s konstantní složitostí
kolize = situace, kdy je pro více záznamů spočítána stejná adresa; obvykle se řeší pomocí bucketů – každé paměťové místo má předepsanou kapacitu záznamů, ve které se následně vyhledává lineárně

Hašovací tabulka je datová struktura, která spojuje *klíče* s *hodnotami*.

Perfektní hašovací funkce – hašovací funkce h, která je prostá.

Požadované vlastnosti hašovací funkce:

- je deterministická
- je rychlá
- vypočítává se z hodnot všech nebo alespoň většiny bitů klíče
- pokrývá cílový prostor rovnoměrně

Složení typické hašovací funkce:

1. generování kódu klíče z hodnoty klíče
2. zobrazení kódu klíče na adresu umístění…
    a. indexu relevantního záznamu v hašované tabulce indexů
    b. relevantního záznamu přímo v hašovaném souboru se záznamy (přímý přístup)

Statické hašování
používá se u souborů, které procházejí jen minimem změn
případné změny mohou negativně ovlivnit efektivitu hašování – pokud se sejde na stejné adrese více záznamů než je kapacita bucketu, vyhledávání v rámci těchto záznamů má lineární složitost

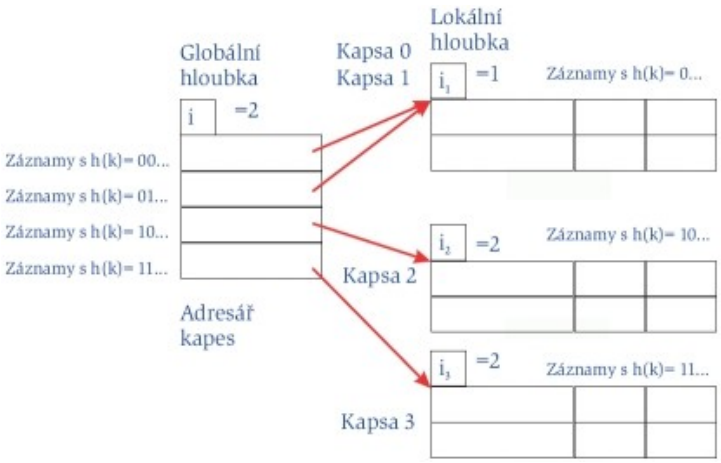
Dynamické hašování
k výpočtu adresy se používá pouze prvních *i* bitů z výstupu hašovací funkce; toto *i* se dynamicky mění – pokud je potřeba více adres, tak se zvyšuje, naopak při malém počtu se *i* zmenšuje
používá se u souborů s proměnným počtem záznamů

buckety jsou naplněné rovnoměrně – pokud jsou plné, tak se štěpí, pokud jsou prázdné, tak se spojují

Druhy:

Rozšiřitelné hašování

- hašovací funkce rozmisťuje záznamy do kapes (kapsa = záznamy s jistou podmnožinou klíčů, vymezuje je hašovací funkce)
- jako index v adresáři kapes se používá dynamicky určovaný prefix výsledku hašovací funkce h(k)
- délku indexu vymezuje globální hloubka adresy kapsy
- štěpení
    - globální hloubka > lokální hloubka – na blok ukazuje více ukazatelů, jednoduše se rozdělí na dva
    - globální hloubka = lokální hloubka – musí vzniknout nová kapsa a také se zdvojnásobí rozměr adresáře kapes



Lineární hašování

- řeší nedostatky rozšiřitelného hašování za cenu vyšší režie dané manipulací s přetokovými kapsami
- počet kapes udržuje tak, aby byly naplněny z např. 80 %
- rozdíl oproti rozšiřitelnému hašování: nemusí se vytvářet ani udržovat adresář kapes, přesto počet kapes roste lineárně.

B stromy a jejich varianty

strom = souvislý orientovaný acyklický (jednoduchý) graf
B strom = m-ární vyhledávací strom s omezujícími podmínkami (viz dále)
B+ strom = redundantní varianta B stromu

Vyhledávací B stromy a jejich varianty se používají pro efektivní ukládání a zpětné zpřístupňování dat. Nejčastěji jsou používány v databázích. Na B+ stromech jsou také založeny například souborové systémy XFS, ReiserFS či NTFS.

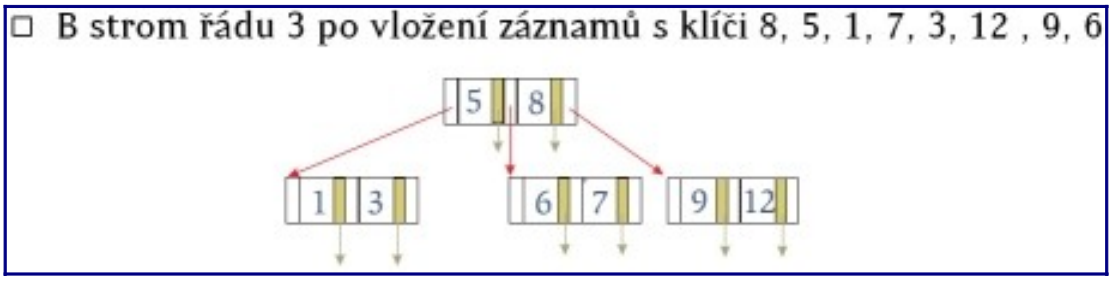
Jiný typ indexu
Sekvenční uspořádání není nutné
Garance I/O pro přístup (vyváženost)

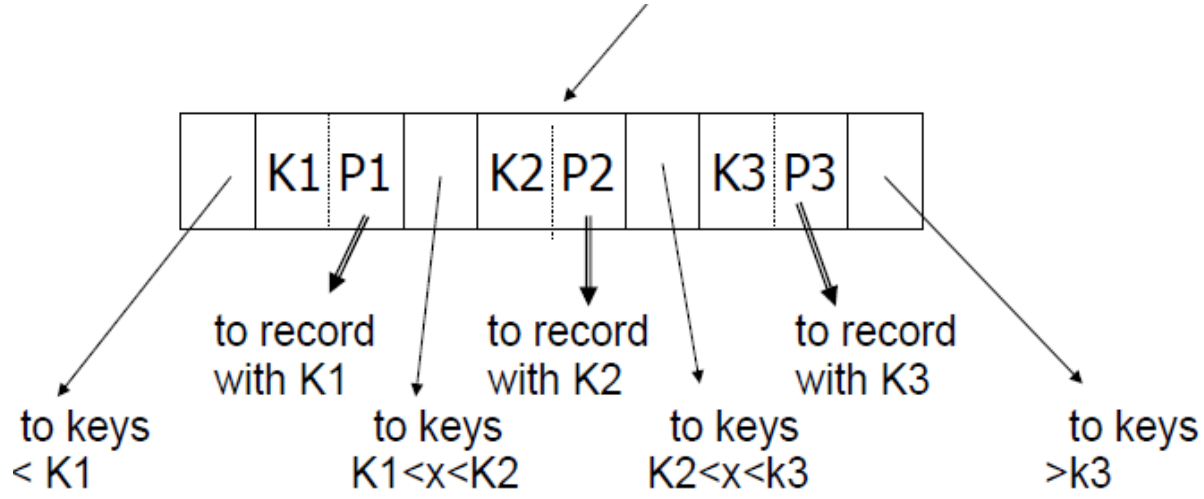
B strom

- Neduplikovat klíče

→odkazy na záznamy i ve vnitřních uzlech
Podmínky pro klíče v podstromech jsou jiné
+ Ve vyhledávání rychlejší (ne vždy)
- Různé struktury vnitřního a listového uzlu
- Mazání je obtížnější na implementaci

- Zřetězenílistůjižnelze využít
- každý uzel až na kořen a listy má aspoň $\lceil m/2 \rceil$ potomků
- kořen má aspoň 2 potomky, pokud není jediným uzlem stromu
- uzel s ($g \leq m$) potomky obsahuje ($g - 1$) vyhledávacích klíčů
- všechny listy jsou na stejné úrovni, v listu je max. $m - 1$ klíčů
- klíče se v celém stromu vyskytují právě jednou, záznamy jsou umístěny přímo v uzlech s klíči nebo jsou z nich adresované
- vlevo od klíče je ukazatel na podstrom obsahující klíče menší než klíč v uzlu, vpravo od klíče je ukazatel na záznam a vedle něj ukazatel na podstrom obsahující klíče větší než klíč v uzlu
- operace přidání, vyjmutí i vyhledávání prvku v B stromě probíhají v **logaritmickém**
- **čase**



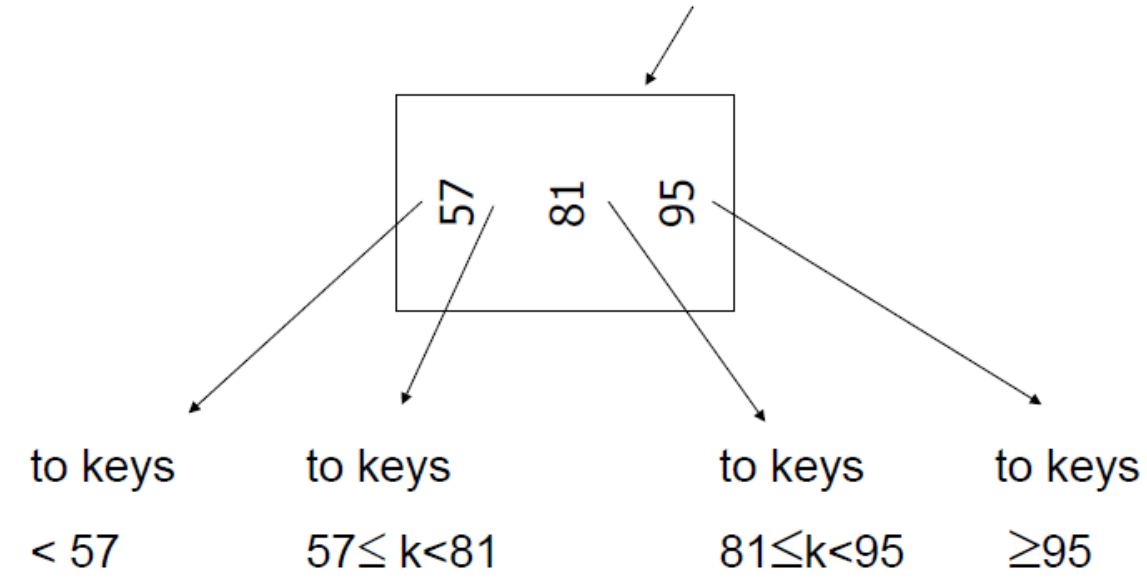


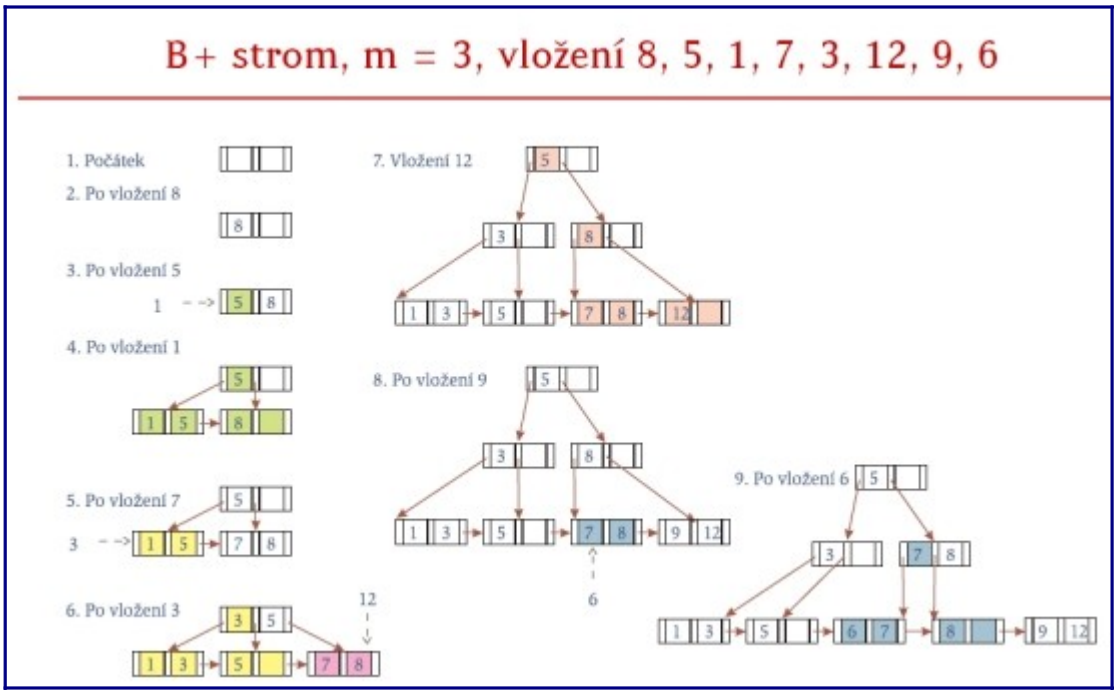
B+ strom

- N – arita, štiepenie (počet ukazovatelov, max počet uložených kľúčov kľúčov (+-1))
- záznamy s daty jsou adresovány pouze z listů
- vnitřní uzly B+ stromu hrají roli indexu k listům (vytvářejí víceúrovňový řídký index listů)
- ve vnitřních uzlech se hraniční klíče mohou opakovat
- Výhody oproti B stromům:
    - uzly jsou menší o ukazatele dat, strom jde více do šířky, průchod stromem je rychlejší
    - vkládání a rušení záznamů je snadnější, jednodušší je celá implementace B+ stromu
    - každý list obsahuje odkaz na následující list (tedy **listy jsou zřetězené**) – umožňuje velice **rychlé procházení** celým stromem.

B+-strom: vkládání
Růst od listu ke kořenu
Nalézt listový uzel a vložit klíč(s odkazem na záznam)
Popř. aktualizovat rodiče
Případy:
a)Bez reorganizace (snadné) - V listu je volnémísto
b)Štěpení listu
c)Štěpení vnitřního uzlu
d)Nový kořen
B+-strom: mazání
Nalézt listový uzel a smazat klíč (s odkazem na záznam)
Popř. smazat list, …
Případy:
a)Bez reorganizace (list není„podplněn“)
b)Přesun do souseda a smažuzel
c)Přesuny mezi sousedy (bez mazániuzlu)
d)Ad (b) a(c) pro vnitřníuzly

- Vnitřní uzel pro *n=4*





B* strom – Obdoba B stromu, která pracuje s uzly naplněnými minimálně do 2/3, místo 1/2 jako u klasického B stromu.

B# strom – Obdoba B+ stromu s povolenou rotací hodnot mezi sousedními uzly.
Trie – Klíče jsou uchovávány na cestách z kořene k externímu uzlu a nikoliv jako celky v jednotlivých uzlech. (Trie **není** *m*-ární vyhledávací strom.)