PV062 – Organizácia súborov - Lekcia 1 – informačná teória

- Informácia = **správa**, fakt, ktorý o niečom vypovedá – charakterizuje objekty, udalosti, javy
- Vždy v sebe nesie znalosť, ktorá je pre príjemcu nová – znižuje informačnú neistotu
- Prenos / uchovávanie – na **nosiči** (signál, pamäťové médium, papier) v rôznych **podobách** (písmená, číslice, špecifické symboly, obrázky, zvuky)
- Informácia má význam, len ak jej príjemca je schopný ju interpretovať („dekódovať“)
- Nemusí byť vždy presná alebo pravdivá
- Má nehmotný, abstraktný charakter, ale je vždy spojená s nejakým fyzikálnym javom
- Iniciátor **informačnej teórie**: Claude Shannon
- Správa, ktorá nesie informáciu má svoju **syntax** (kvantitatívna stránka – *koľko* informácie obsahuje daná správa? Aká je miera prekvapenia?) a **sémantiku** (kvalitatívna stránka - obsah)

ENTROPIA: sledujme výsledok procesu výberu jednej alebo viacerých alternatív z nejakej množiny možných alternatív. Napr. zariadenie I produkuje s rovnakou pravdepodobnosťou signály A, B, C, a pýtame sa, *aký signál bude vybraný teraz*, miera neurčitosti je 3. Keď výstupnú hodnotu uvidíme, neurčitosť zmizne. Znížením neurčitosti získavame **informáciu.**

Zariadenie II produkuje znaky 1, 2 a teda vykazuje mieru neurčitosti 2. Kombinované zariadenie I+II produkuje výstupy A1, A2, B1, B2, C1, C2 a má mieru neurčitosti 6, ale *nie je meraná multiplikatívne*. Miera množstva informácie má **aditívny** charakter (keď si prečítam dve knihy, získam množstvo informácie odpovedajúce súčtu množstiev informácie v každej z nich). Je výhodné zaviesť: **f{x} = - log(p(x))**; t.j. sčítavame záporné hodnoty logaritmov p-stí možných výstupov.

Nech zariadenie III produkuje trvale jedinú výstupnú hodnotu. Keďže ju poznáme, nezískame žiadnu informáciu a zariadenie produkuje výstup s pravdepodobnosťou 1, log 1 = 0.

- $I(A) = \log (1/P(A)) = - \log P(A)$ v jednotkách Sh – Shannon
- $I(A) = \log_2 (1/P(A)) = - \log_2 P(A)$ v jednotkách bit

Množstvo informácie v správe X súvisí s **pravdepodobnosťou jej výskytu**. Správa „v rulete padlo číslo 17“ prináša väčšie množstvo informácie ako správa „v rulete padlo nepárne číslo“.

- Menej pravdepodobná správa prináša väčšie množstvo informácie, znižuje väčšiu neurčitosť
- Množstvo informácie je vždy kladné
- Množstvo informácie v skupine nezávislých správ je rovné súčtu množstva informácií obsiahnutých v jednotlivých správach

Neurčitosť zdroja: príjemca musí poznať, aké všetky možné správy môžu byť produkované, ale nevie, akú správu od zdroja obdrží. Prijatím správy je potom táto neurčitosť odstránená. Neurčitosť, entropia **zdroja** správy **X**, H(X) je rovná množstvu informácie v správe obsiahnutej, t.j. H(X) = I(X)

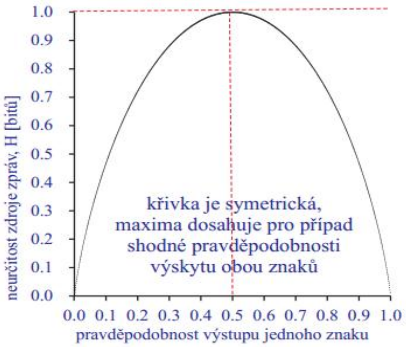
Determinované systémy majú nulovú neurčitosť, správy o ich stavoch nesú nulovú informáciu. Najväčšiu neurčitosť má systém, pri ktorom sú pravdepodobnosti jeho stavov rovnomerne rozdelené. Neurčitosť systému generujúceho správy v závislosti na stavoch, v ktorých sa nachádza závisí

- Na počte stavov systému
- Na pravdepodobnosti výskytu jednotlivých stavov

Ak systém nadobúda *n* možných stavov s pravdepodobnosťami p1, p2, ... pn, pričom súčet týchto pravdepodobností je 1, potom každý stav prispieva do neurčitosti systému svojou neurčitosťou.

Shannonova formula: neurčitosť H(X) zdroja informácií determinuje množstvo informácií zdrojom generované.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$



Entropia je najväčšia, ak sú všetky udalosti rovnako pravdepodobné. Vzrastie, ak sa zvýši počet možných výstupov.

Priemerné množstvo informácie je 1,75 b / symbol. ASCII: 7b/symbol

Informácie musíme pre účely spracovania, zdeľovania, skladovania, prenosu ... efektívne vyjadriť, pričom každý účel kladie individuálne požiadavky na efektivitu vyjadrenia (napr. zdeľovanie – zrozumiteľnosť vyjadrenia, skladovanie – minimálny bitový objem atď.)

- Statické (textové) informácie sa vyjadrujú **diskrétnymi symbolmi**
- Dynamické (napr. zvukové) informácie vyjadrujeme v čase spojito, **analógovo**
- Analógové správy je možné tiež vyjadriť postupnosťami diskrétnych správ nesúcich hodnoty analógových charakteristík nameraných v istých diskrétnych časoch

Lekcia 2 – kódovanie informácií

- **Kódovanie** = proces nahradzovania symbolov *zdrojovej abecedy A* symbolmi *cieľovej (kódovacej) abecedy* A_c
- Použitie: vytváranie dát, šifrovanie, kompresia, samoopravné kódovanie
- **Abeceda** = konečná množina znakov. Pre nás $A_c = \{0,1\}$
- Konečná postupnosť prvkov abecedy tvorí **slovo** S, jeho dĺžka je značená |S|
- Množinu všetkých slov nad abecedou A značíme A*, množinu všetkých nenulových slov A^+
- **Kódové slovo** je postupnosť prvkov A_c použitá pre vyjadrenie postupnosti prvkov A
- Kódovanie je potom zobrazenie K: A → A_c^+
- Kód C daný kódovaním K je trojica C: (A, A_c^+, K); K je obor hodnôt
- Zobrazenie K musí byť *prosté*, aby kód mohol byť **jednoznačne dekódovateľný**

Klasifikácia kódov podľa dĺžok kódových slov:

- Kód s kódovými slovami pevnej (rovnakej) dĺžky – napr. *blokový kód* (ASCII)
- Kód s kódovými slovami **premenlivej** dĺžky – napr. *prefixový, sufixový kód* – má za cieľ **redukovať bity** potrebné pre zobrazenie informácie (častejšie sa vyskytujúce slová sa kódujú na kratšie kódové slová, redšie sa vyskytujúce slová sa kódujú na dlhšie kódové slová)

Stupeň (rate) **kódu** = priemerný počet bitov použitých pre kódovanie symbolov v istom kóde. Priemerná dĺžka kódového slova musí byť väčšia (nanajvýš rovná) ako entropia kódovanej správy. Kód je **optimálny**, keď jeho stupeň *minimálne prevyšuje* (nemôže byť nižší, lebo by sme stratili isté množstvo informácie) množstvo informácie obsiahnutej v symboloch zdrojových správ.

Príklad jednoznačne dekódovateľného neblokového kódu: Nech zdroj generuje symboly A, C, G, T s pravdepodobnosťami: P(A) = 1/2, P(C) = 1/4, P(G) = 1/8, P(T) = 1/8. Množstvo informácie predstavované jednotlivými znakmi (- $\log_2$ P(i)) je: i(A) = 1 bit, i(C) = 2 bity, i(G) = 3 bity, i(T) = 3 bity. Neurčitosť daného zdroja (množstvo ním produkovanej informácie) je H = 1/2 + 2/4 + 3/8 + 3/8 = 1.75 bit/symbol. 4 príklady kódov pre tento prípad:

- Kódovacia funkcia K **kódu 1** nie je prostá (A = 0 aj C = 0) a kód 1 nie je jednozn. dekódovateľný – ignorujeme ho
- Kódovacia funkcia K kódov 2,3,4 je síce prostá (= sú to **nesingulárne kódy**), ale **kód 2** nie je jednozn. dekódovateľný, (0000 môže byť AAAA, GG, AAG, ...).
  Preto sa pri kódovaní nesingulárnym nejednoznačne dekódovateľným kódom jednotlivé slová musia **oddeľovať** (napr. čiarkou), čo je ale nevýhodné

Znak	Pravd.	Kód 1	Kód 2	Kód 3	Kód 4
A	0,5	0	0	0	0
C	0,25	0	1	10	01
G	0,125	1	00	110	011
T	0,125	10	10	111	0111
stupeň		1,125	1,25	1,75	1,875

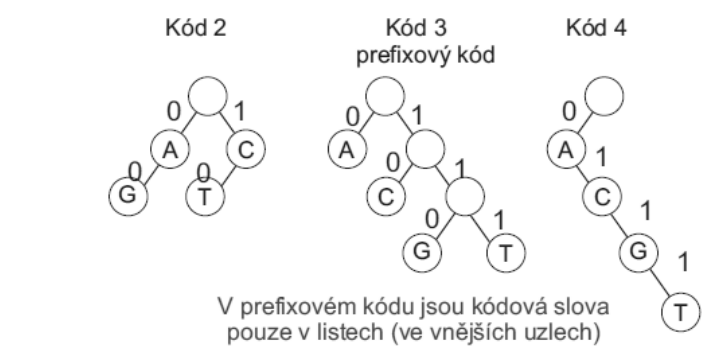
- **Kód 3** je *bezprostredne jednoznačne dekódovateľný*, čo je vysoko žiadúce – jednotlivé kódové slová sa dajú pri analýze zľava doprava rozpoznať okamžite po prečítaní. Jedná sa o **prefixový** kód = žiadne z kódových slov tohoto kódu nie je prefixom iného kódového slova (napr. UTF-8)
- **Kód 4** je tiež nesingulárny jednoznačne dekódovateľný, ale nie *bezprostredne dekódovateľný* – kódové slovo sa rozpozná až pri prečítaní úvodnej 0 ďalšieho kódového slova

Prefixový q-árny kód (s kódovacou abecedou o q prvkoch) s dĺžkami kódových slov d_1, d_2, ..., d_m existuje práve vtedy, keď je splnená **Kraftova nerovnosť:** $\sum_{i=1}^{m} q^{-d_i} \leq 1$

McMillanova veta: Kraftova nerovnosť platí pre ľubovoľný jednoznačne dekódovateľný kód.

- Navrhujeme binární prefixový kód pro kódování cifer
  0, 1, . . . , 9
  ve zprávach s velmi často se vyskytujícími ciframi 0 a 1 a velmi řídce se vyskytujícími ciframi 8 a 9
- Nápad 1, nerealizovatelný
  - ✓ délka kódových slov pro 0 a 1 bude rovna 2 (např. 00 a 01)
  - ✓ délka kódových slov pro 8 a 9 bude rovna 5 (1xxxx)
  - ✓ délka kódových slov pro 2 a 3 bude rovna 3 (1xx)
  - ✓ délka kódových slov pro 4, 5, 6 a 7 bude rovna 4 (1xxx)
  - ✓ $2 \times 2^{-2} + 2 \times 2^{-3} + 4 \times 2^{-4} + 2 \times 2^{-5} = 1,0625$
- Nápad 2, realizovatelný
  - ✓ délka kódových slov pro 0 a 1 bude rovna 2 (např. 00 a 01)
  - ✓ délka kódových slov pro 8 a 9 bude rovna 5 (1xxxx)
  - ✓ délka kódových slov pro 2, 3, 4, 5, 6 a 7 bude rovna 4 (1xxxx)
  - ✓ $2 \times 2^{-2} + 6 \times 2^{-4} + 2 \times 2^{-5} = 0,9375$

- Nápad 3, ještě lépe navržený realizovatelný kód
  - ✓ $2 \times 2^{-2} + 2^{-3} + 5 \times 2^{-4} + 2 \times 2^{-5} = 1$
  - ✓ 0 – 00
  - ✓ 1 – 01
  - ✓ 2 – 100
  - ✓ 3 – 1010
  - ✓ 4 – 1011
  - ✓ 5 – 1100
  - ✓ 6 – 1101
  - ✓ 7 – 1110
  - ✓ 8 – 11110
  - ✓ 9 – 11111



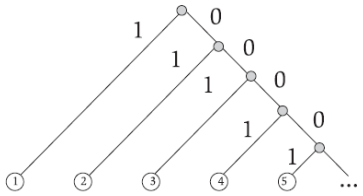
KAŽDÝ PREFIXOVÝ KÓD JE NESINGULÁRNY A JEDNOZNAČNE DEKÓDOVATEĽNÝ.

Unárny kód: C(i) = i – 1 núl nasledovaných jednotkou (alebo i – 1 jednotiek nasledovaných nulou). Je optimálny pre p(i) = $1/2^i$

Binárny kód: čísla sú zapísané v dvojkovej sústave. Nespĺňa vlastnosť jednoznačnej dekódovateľnosti – musíme použiť oddeľovače, alebo ho treba definovať ako blokový kód. Je optimálny pre p(i) = 1/N.

Unární kód:

1	1
2	01
3	001
4	0001
5	00001
6	...



Prvý Eliasov kód pre kódovanie kladných celých čísel: zapíšte číslo binárne, odčítajte 1 od počtu bitov a na začiatok pripojte taký počet núl (resp. zapíšte počet bitov v unárnom kóde pred číslo).

- nechť $\bar{B}(n)$ značí $B(n)$ bez nejvýznamějšího bitu (vždy = 1)
- C_1 : kladné celé n se kóduje zápisem $\bar{B}(n)$ s prefixem = délce binární reprezentace n, $|\,B(n)\,|$ v unárnímu kódu
  - ✓ $n = 38 = 100110_2$, $\bar{B}(38) = 00110_2$, $|\,B(38)\,| = 6$, $\Rightarrow C_1(38) = 00000100110_2$
  - ✓ $n = 5 = 101_2$, $\bar{B}(5) = 01_2$, $|\,B(5)\,| = 3$, $\Rightarrow C_1(5) = 00101_2$
  - ✓ $n = 1 = 1_2$, $\bar{B}(1) = \varepsilon$, $|\,B(1)\,| = 1$, $\Rightarrow C_1(1) = 1_2$

Druhý Eliasov kód je preusporiadaním prvého: každý z bitov, ktorý ostane po odseknutí sa uvedie príznakom 0, na koniec kódového slova sa pridá 1.

- ✓ $C_2(5) = 00011_2$, tj. délkový prefix se rozptýlí do $\bar{B}(n)$

Tretí Eliasov kód sa používa pre veľké čísla:

- Nejprve ilustrace s malým číslem, 50 (110010_2)
  - ✓ Kódové slovo Eliasova kódu C_1 pro 50 je 000001100010_2, $|\,B(50)\,| = 6$ bitů,
  - ✓ Jeho permutací na kódové slovo C_2 vznikne 010000010010_2 délka kódového slova pro 50 v C_2 je 11 bitů
  - ✓ délka kódového slova pro 50 v C_3 je 10 bitů $C_1(6) = 00110$, $C_2(6) = 01001$, $C_3(50) = 0100110010_2$
- a nyní Eliasův kód C_3 pro 1 000 000 (11110100001001000000_2), délka = 20 bitů
  - ✓ Kódové slovo Eliasova kódu C_1 pro 1000000 je $00000000000000000001111010000100100000 0_2$, $|\,B(1\,000\,000)\,| = 20$ bitů, $|\,C_1(1\,000\,000)\,| = 39$ bitů
  - ✓ délku 20 vyjádříme v C_2, $20 = 10100_2$, $|\,B(20)\,| = 5$, $C_1(20) = 000010100_2$, $C_2(20) = 000100001_2$
  - ✓ Kódové slovo Eliasova kódu C_3 pro 1 000 000 je $00010000111101000010010000000_2$ a má délku 28 bitů místo 39 bitů kódu téhož čísla v C_2

Lekcia 3 – kompresia dát

- Je proces identifikácie a odstraňovania redundancie v dátach
    - Bezstratová (lossless) = zobrazenie rovnakého objemu informácie do menšieho bitového priestoru; dekompresiou sa získajú originálne dáta (typicky texty)
    - Stratová (lossy) = odstránenie nepodstatnej informácie v obrázkoch a zvukoch
- Kódovanie správy dĺžky O(X) [b] na správu dĺžky L(X) [b]
- Cieľom je dosiahnutie L(X) << O(X)
- Ideálne L(X) → H(X), kde H(X) je neurčitosť správy
- Menej než H(X) bitmi správu zakódovať nemôžeme, ak má byť kompresia bezstratová
- **Kompresný pomer:** L(X) / O(X), napr. 0.75 značí 25% redukciu dĺžky správy
- **Kompresný faktor:** O(X) / L(X)
- **Kompresný zisk:** 1 – (L(X) / O(X))
- **Redundancia správy:** R(X) = O(X) – H(X)
- Redundancia komprimovanej správy: R‘(X) = L(X) – H(X)
- **Fyzická kompresia** = ignoruje význam komprimov. dát, pozerá sa na ne len ako na sled bajtov
- **Logická kompresia** = zohľadňuje význam komprimovaných dát, často pri stratovej kompresii
- 1. fáza = **modelovanie** = hľadanie a popis redundancie vo vstupných dátach – vznik **modelu**, ktorý definuje symboly, resp. zdrojové komponenty správ.
    - **statický** model = jeho parametre pre kódovanie vstupných dát sú konštantné
      => statická kompresia – procedúra nezávislá na vstupných dátach
    - **adaptívny** model = jeho parametre sa menia podľa toho, ako sa menia charakteristiky kódovaných vstupných dát
      => adaptívna kompresia – rešpektuje vlastnosti vstupných dát, dynamická
- 2. fáza = **kódovanie** = generovanie výstupných (komprimovaných) dát

Kompresné metódy:

- **štatistické:** častejšie sa vyskytujúce vzorky dát sa kódujú kratšími kódovými slovami
    - počas modelovania sa prvkom priraďujú pravdepodobnosti výskytu
    - Shannon-Fanovo, Huffmanovo, aritmetické kódovanie
- **slovníkové:** udržuje sa zoznam často opakovaných vzorkov vstupných dát
    - počas modelovania sa vyčleňujú opakovane sa vyskytujúce vzorky
    - LZ77, LZ78, LZW

Štatistické modely:

- nultého stupňa: P výskytu všetkých znakov sú **rovnaké** – odpovedá mu náhodný text a preto sa príliš často nepoužíva (napr. pre DNA sekvencie, ale nie text v počítači)
- prvého stupňa: P výskytu všetkých znakov sú **rôzne** – najširšie využitie
- n-tého stupňa: model pracuje s P výskytu n-tíc znakov

Základné, intuitívne techniky:

- **Braillovo písmo**: rok 1820 – 6 bodov v obdĺžniku 3x2, t.j. 6 bitov a 2^6 kódových slov
- **Baudotov kód:** 5 bitový
- **MacWrite kódovanie:** 4-bitové kódy pre 15 najpoužívanejších znakov + kód ESCAPE, za ktorým nasleduje na 8 bitoch ASCII znak

- Typ **RLE (Run Length Encoding):** náhrada opakujúcich sa susedných symbolov počtom opakovaní (jeden bajt) a špecifikáciou opakovaného symbolu (druhý bajt) => „bajtový prúd“
    - Používal sa napr. v modemoch: MNP (Microcom Network Protocol)
    - 3 identické bajty bezprostredne za sebou spustia RLE
    - Za ne sa vsunie **repetičný indikátor** (číslo určujúce počet opakovaní znaku)
    - Napr. AAA → AAA0, AAAA → AAA1, AAAAA → AAA2
    - Vylepšenie: in-flight-coding = zohľadňuje sa štatistika výskytu znakov
    - Kompresia neopakovaných znakov je kontraproduktívna!
    - ITU-T T4 – štandard faxových strojov – každý riadok je tvorený striedaním postupností bielych a čiernych pixelov
    - BinHex 4.0
- **Bentleyho kódovanie Move to Front**: dynamicky sa vytvára a udržiava abeceda A, ktorej prvky sa v komprimovanom texte vyskytujú najčastejšie.
    - Príklad: A = a, b, c, d, m, n, o, p
    - Indexy sa označia binárne od 1 do 8: 1, 10, 11, 100, 101, 110, 111, 1000
    - Každá hodnota indexu i sa **prefixuje** počtom núl = $\log_2 i$
    - Teda 1, **0**10, **0**11, **00**100, **00**101, **00**110, **00**111, **000**1000

✓ kódování řetězu $a\,b\,c\,a\,p$ a uspořádání abecedy:

$$a \to 1 \qquad A = a, b, c, d, m, n, o, p$$
$$a\,b \to 1\,010 \qquad A = b, a, c, d, m, n, o, p$$
$$a\,b\,c \to 1\,010\,011 \qquad A = c, b, a, d, m, n, o, p$$
$$a\,b\,c\,a \to 1\,010\,011\,011 \qquad A = a, c, b, d, m, n, o, p$$
$$a\,b\,c\,a\,p \to 1\,010\,011\,011\,0001000 \qquad A = p, a, c, b, d, m, n, o$$

✓ dekódování řetězu 10100110110001000 a uspořádání abecedy:

$$1 \to a \qquad A = a, b, c, d, m, n, o, p$$
$$1\,010 \to a\,b \qquad A = b, a, c, d, m, n, o, p$$
$$1\,010\,011 \to a\,b\,c \qquad A = c, b, a, d, m, n, o, p$$
$$1\,010\,011\,011 \to a\,b\,c\,a \qquad A = a, c, b, d, m, n, o, p$$
$$1\,010\,011\,011\,0001000 \to a\,b\,c\,a\,p \qquad A = p, a, c, b, d, m, n, o$$

Štatistické metódy: *prvkom vstupnej abecedy s väčšou P výskytu sa priradia prvky výstupnej abecedy kódované kratšími bitovými reťazcami*. Používajú sa **prefixové kódy** premenlivej dĺžky (Morseovka).

- Príklad: majme abecedu {A, B, C, D} kde P výskytu jej prvkov je: 3/4, 1/8, 1/16, 1/16
- Entropia zdroja:
    - $-(3/4)\log_2(3/4) -(1/8)\log_2(1/8) -(1/16)\log_2(1/16) -(1/16)\log_2(1/16) = 1.19$ b/symbol
- Kódovanie 1 (nezohľadňuje pravdepodobnosť)
    - A = 00, B = 01, C = 10, D = 11, priemer = 2 b/symbol
- Kódovanie 2: prefixový kód, slová premenlivej dĺžky
    - A = 0, B = 10, C = 110, D = 111
    - Priemer = 3/4 * 1 + 1/8 * 2 + 1/16 * 3 + 1/16 * 3 = 1.375 b/symbol = **efektívnejšie**

SHANNON - FANOVO kódovanie: kód je reprezentovaný binárnym stromom vybudovaným nad znakmi vstupnej abecedy zoradenými do neklesajúcej postupnosti podľa ich početnosti. Tento strom vznikne rekurzívnym delením postupnosti na dve časti s početnosťami najbližšími ½ (zhora dole).

HUFFMANOVO kódovanie: vytvára tzv. Huffmanov strom opačným smerom: zospodu nahor. Tieto kódovania sú optimálne, ak P výskytu jednotlivých znakov sú zápornými mocninami 2. Potom sú entropie jednotlivých znakov celočíselné a odpovedajú dĺžke kódu.

ARITMETICKÉ kódovanie: ak P výskytu jednotlivých znakov NIE sú zápornými mocninami 2, dochádza k redundancii, ktorú je možné znížiť použitím aritmetického kódovania – nekóduje jednotlivé symboly, ale celú správu – výsledkom je číslo z intervalu [0,1). **Viz. príklady, slidy: 47 – 147.**

Lekcia 4 – súborové organizácie

- **Databáza** = základný nástroj pre dlhodobé uchovávanie a sprístupňovanie dát
    - uchováva dáta na vonkajšej pamäti vo forme súborov
- **Súbor** = pomenovaná kolekcia súvisiacich informácií – záznamov
    - homogénne (obsahuje záznamy jedného / neštruktúrovaného typu) a nehomogénne
    - statické (nemenné, len na čítanie) a dynamické (menia sa, zápis)
- **Záznam** = bázová dátová jednotka reprezentujúca nejaký objekt (osobu, predmet, ...)
    - charakterizovaná svojimi vlastnosťami – **atribútmi**
    - atribúty sú uložené v poliach (**položkách**) záznamu
    - s každým atribútom súvisí jeho **dátový typ** (integer, float, string, bool) určujúci obor hodnôt atribútu a množinu povolených operácií nad ním
    - Kolekcia položiek + definícia ich dátových typov = **typ záznamu:**
        - Logický záznam = len hodnoty atribútov vytvárajúce zoznam
        - Fyzický záznam = hodnoty atribútov vytvárajúce zoznam + **definícia dĺžok** atribútov (môže byť konšt. alebo variabilná)

Súbory sa dlhodobo uchovávajú na energeticky nezávislých vonkajších pamätiach. Sprístupňovanie

- súboru ako celku riešia **adresárové služby** implementované ako služby OS
- jednotlivých záznamov riešia **organizácie súborov** implementované v knižniciach (volajú OS)
    - sprístupnenie záznamu: pomocou dotazu nad súborom: jednorozmerný alebo viacrozmerný (ortogonálny) na úplnú/čiastočnú (intervalovú) zhodu

Cieľ návrhu súborových štruktúr: minimalizácia počtu prístupov na disk – ideálne **1 prístup** na 1 operáciu so záznamom. Možnosti prístupu: sekvenčný + **priamy** = indexovanie alebo hashovanie.

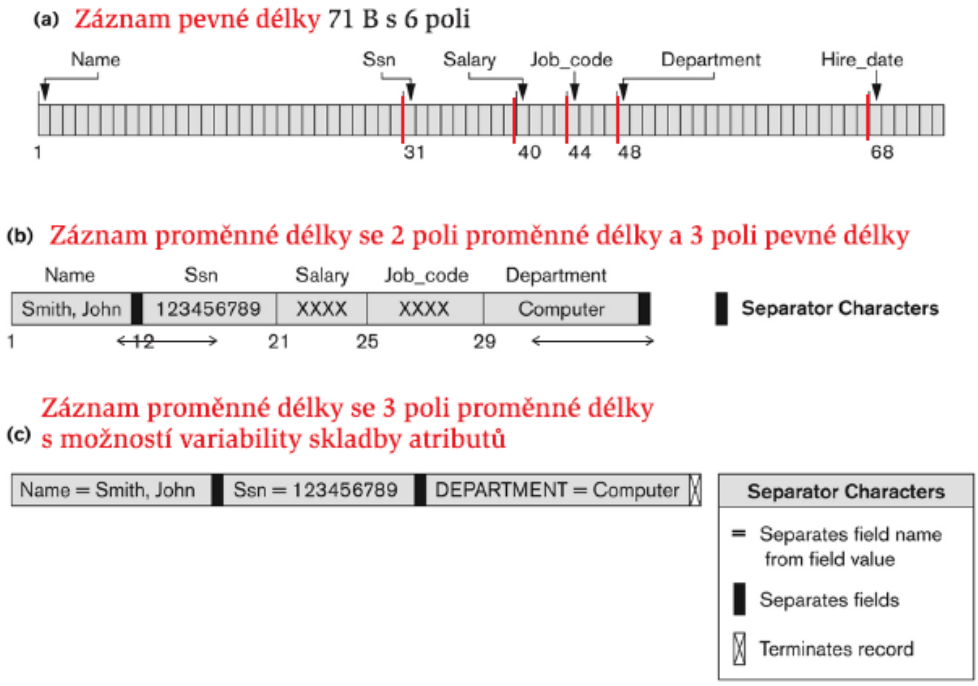
INDEX: tvorený dvojicami {hodnota_kľúča, adresa_na_disku} (ukazateľ na záznam). Implementácia:

- pomocou AVL stromu – nevýhodné pre disky, lebo každý uzol obsahuje len 1 záznam + rastie zhora nadol => zložitá reorganizácia pri vyvažovaní stromu
- pomocou **B stromu**, kde závislosť doby prístupu je $\log_k N$; k = arita uzlu stromu, N = celkový počet záznamov v súbore. B strom rastie zdola nahor, čo umožňuje jednoduchšiu reorganizáciu pri vyvažovaní stromu

Kľúč = časť záznamu, ktorá ho identifikuje. **Primárny** kľúč identifikuje záznam jednoznačne; nemal byť obsahovať dáta, ktoré sa môžu meniť (tzv. dataless key). **Sekundárny** kľúč je vyhľadávací, pomocný. Typy kľúčov:

- **hodnotový** = je určený hodnotou
- **relatívny** = pozícia záznamu vzhľadom k začiatku súboru
- **hashovaný** = algoritmicky transformovaný

Záznamy môžu byť pevnej a premenlivej dĺžky (= nutné ukončenie oddeľovačom):



Blok (**fyzická stránka**) vonkajšej pamäte je samostatne manipulovateľná (adresovateľná) jednotka.

- Neblokovaný záznam: blok obsahuje práve 1 záznam
- Blokovaný záznam: blok obsahuje celistvý počet záznamov
- Prerastený záznam: záznamy sú zapisované do blokov bez ohľadu na hranice blokov (súbor je niekedy rozdelený do viacerých blokov – nutný ukazateľ na ďalší blok)

Hierarchická abstrakcia organizácie súborov v 3 úrovniach - **schémach**:

- **Logická** – hypotetická logická pamäť členená na stránky – obsahuje primárne súbory (dáta) a sekundárne súbory (indexy) => umožňuje operácie nad záznamami v pamäti
- **Fyzická** – zobrazenie logických stránok do fyzických stránok konkrétnej pamäte
- **Implementačná** – rozmiestnenie a alokácia fyzických stránok v pamäti

Lekcia 5 – hašovanie (hashing, randomizing, direct addressing)

Hašovacia funkcia m = h(k) rieši **prístup k záznamom súboru** v čase O(1). Nezamieňať s kontrolným súčtom, fingerprintom alebo kryptografickou hashovacou funkciou.

- h = hašovacia funkcia, generátor *rovnomerne rozložených* hodnôt
  - ak je prostá, hašovanie je **perfektné**
  - ak je prostá a priestor cieľových adries (obor hodnôt) má rozsah rovný počtu hodnôt kľúča, hašovanie je **minimálne a perfektné**
  - ak nie je prostá, musí existovať schéma riešenia **kolízií**
  - kolízia = dva alebo viac kódov kľúčov sa zobrazujú na rovnakú adresu umiestnenia
  - kolidujúce adresy sa uložia do tzv. **kapsy** (viz. ďalej), alebo sa hľadá iné miesto
- k = numerická hodnota
  - nenumerický **vyhľadávací kľúč** je vopred prevedený na **kód kľúča**
  - tento kód kľúča potom určuje pozíciu (adresu) v relevantnej dátovej štruktúre

- m = adresovaná hodnota
    - tabuľka s indexmi záznamov súborov (súbory s hašovanými indexmi)
    - pamäť obsahujúca záznamy súborov (súbory s priamym prístupom)

Požadované vlastnosti hašovacej funkcie:

- rýchla: realizovaná niekoľkými inštrukciami
- deterministická: jej hodnoty závisia len na hodnotách kľúčov – t.j. vypočítava sa z hodnôt všetkých alebo väčšiny bitov kľúča
- referenčne transparentná
- náhodná, ale rovnomerná

Príklady hašovacích funkcií:

- generovanie **absolútnej** (závislá na zariadení) / **relatívnej** (číslo bloku v rámci súboru) **adresy**
- **modulo**: m = k mod M, ale dáva zlé výsledky, ak M nie je prvočíslo, preto modifikácia:
- m = (a*k + b) mod M (M = počet adries)
- **mid-square:** k = 246**158**34, 158^2 = 2**496**4, h(k) = 496
- zmena číselnej sústavy – **radix transformation**: k = 38652_{10}, h(k) = 38652_{11} (= 55354_{10})

Stratégia riešenia kolízií pri vkladaní / hľadaní záznamu: (ovplyvňuje časovú zložitosť: O(n) až O(1))

- **otvorené adresovanie** – riešenie kolízií prebieha v rámci rovnakého adresového priestoru – pomocou nejakého vyhľadávacieho algoritmu v ňom nájdem iné prázdne miesto
    - **lineárne hľadanie:** k východzej adrese pridáva +1 a kontroluje, či už je tam voľno (pri vkladaní), resp. či sa tam nachádza hľadaný záznam; môže spôsobovať zhluky
    - **kvadratické:** pripočítava kvadrát z aktuálneho kroku
    - **dvojité:** keď nenájde prázdne miesto, použije druhú hashovaciu funkciu h2 a následne postupuje o krok h1(k) + h2(k)
    - typicky sa používa pri hashovaní vo vnútornej pamäti
- **uzavreté adresovanie** – nehľadám iné prázdne miesto, ale použijem **kapsy**
    - kapsa = základná jednotka pre ukladanie dát veľkosti jedného bloku
    - ukladáme do nej kolidujúce prípady, ktoré potom prehľadávame sekvenčne
    - ak sa jej kapacita vyčerpá, naviažeme na ňu pretokovú kapsu – **bucket**

STATICKÉ hašovanie – pre statické súbory; zmena súboru (delete, insert) vyvolá zmenu v rozložení kľúča a hashovacia funkcia začne generovať viac kolízií. **DYNAMICKÉ hashovanie** – pre dynamické súbory – rešpektuje zmeny súboru. Rozmiestňuje záznamy do kapsy *rovnomerne*. Patrí sem:

Rozšíriteľné (extendible) hašovanie – Fagin: primárny súbor s dátami sa doplňuje sekundárnym, ktorý sa nazýva indexový **adresár**, čo je *pole ukazateľov na kapsy* => kapsy nie sú adresované priamo.

- Adresár sa buduje z hodnôt, ktoré sú výsledkom aplikácie hašovacej funkcie na kľúč.
- Môže dynamicky rásť/skracovať sa o mocniny dvojky – dĺžka poľa je 2^d, kde d je tzv. **globálna HĺBka** – určuje počet záznamov.
- Cieľová položka adresára obsahuje **adresu kapsy** (t.j. adresu bloku vonkajšej pamäti obsahujúcej kapsu) obsahujúcu záznam s cieľovou hodnotou vyhľadávacieho kľúča.
- Každý záznam nemusí mať vlastnú kapsu (ak sa vojde viac záznamov do jednej) => určuje sa aj lokálna hĺbka kapsy. Väčšinou nie sú nutné pretokové kapsy – radšej sa vytvorí nová v pamäti – iba ak by sa prekročila vopred stanovená hranica počtu kapies.

Nájdenie kapsy obsahujúcej záznam s vyhľadávacím kľúčom k:

1. vypočítaj h(k)
2. zisti index i z globálnej hĺbky adresy kapies
3. použi bity z prvých i **horných** rádov h(k) ako index do tabuľky adries kapies a získaj ukazateľ kapsy, ktorú prehľadaj sekvenčne
4. neúspech = záznam v súbore neexistuje

Vloženie záznamu s kľúčom k do súboru:

1. nájdi odpovedajúcu kapsu s indexom j podľa prechádzajúcich bodov 1,2,3
2. ak je v bloku dát kapsy j miesto, zapíš doňho záznam
3. ak nie, **rozštiep** blok dát na dva; uprav ukazatele kapies, (môže sa stať, že na jeden blok dát odkazuje viac ukazateľov) - viz. príklad, slidy: 40-45

Klady: výkon hashovania s rastom súboru neklesá, nie sú nutné pretokové kapsy.
Zápory: adresár kapies môže byť veľký => veľké nároky na pamäť, potreba ukazateľov.

Lineárne hashovanie – Litwin: odstraňuje adresár; za cenu zníženia pamäťových nárokov zvýši výpočtové – manipuluje s pretokovými kapsami.

- Počet kapies udržuje taký, aby boli naplnené cca z 80%. Ako rastie adresový priestor, rastie aj počet bitov hashovanej hodnoty používanej pre rozmiestňovanie záznamov; n kapies je možné identifikovať $\log_2$ n bitmi. Tieto bity sa vždy berú z **najnižších** rádov h(k).
- Počiatočne sú záznamy súboru rozmiestnené v M kapsách hašovacou funkciu h(k) = k mod M. Každá kapsa má individuálne pretokové oblasti – keď začne pretekať, iná kapsa sa začne štiepiť, aby spravila miesto (viz. príklad, slidy: 50-54).

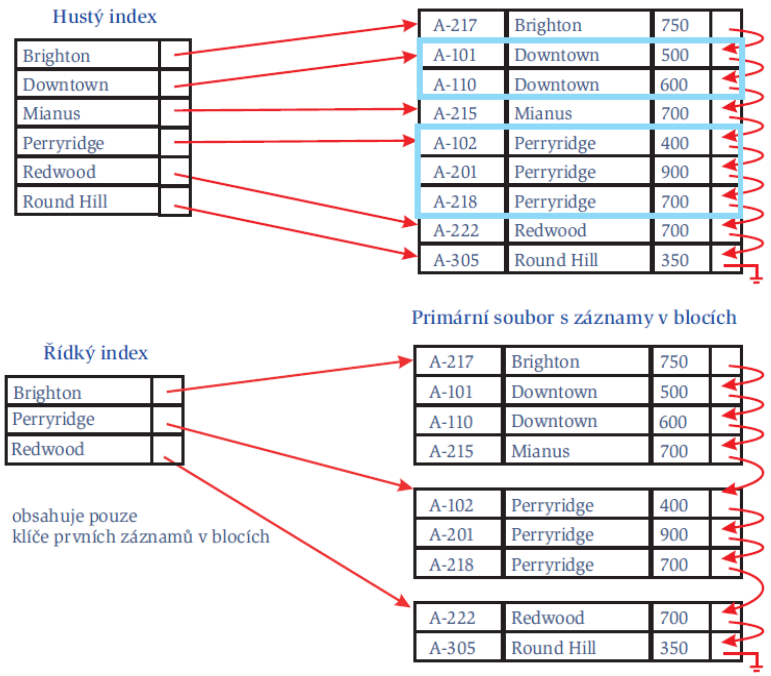
Klady: nemusí sa vytvárať adresár, adresový priestor sa rozširuje pri každom štiepení.

Lekcia 6 – indexové organizácie

- Sekvenčné súbory - neusporiadané:
    - operácie INSERT v čase O(1); FIND v čase O(n)
    - FIND pri blokovacom faktore *b* (počet záznamov na blok) v O(n/*b*) = stále lineárna
    - priemerný počet prístupov na disk pri vyhľadávaní záznamu je n/2
- Usporiadané:
    - **keysort** – triedi sa len kľúč, ktorý je zviazaný so súborom => FIND v O(log n)
    - po INSERT nutná reorganizácia, ale nie zakaždým – zmeny v súbore sa ukladajú do **súboru aktualizácií**, čo je neusporiadaný sekvenčný súbor, ktorý sa občas (v dávkach) zatriedi do primárneho súboru

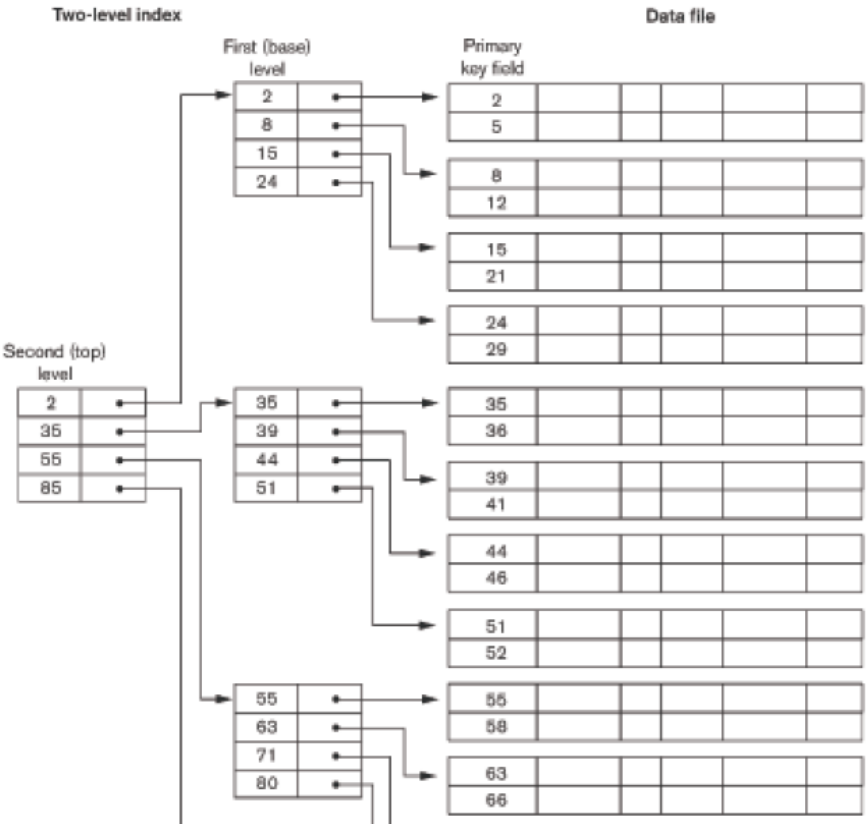
Indexovanie: mechanizmus pre vyhľadanie hodnoty záznamu, ktorého kľúč vyhovuje podmienke
Index = samostatný usporiadaný sekundárny súbor

- **Hustý index** = pre každú hodnotu vyhľadávacieho kľúča existuje aspoň jeden indexový záznam. Ten odkazuje na kapsu, ktorá obsahuje všetky záznamy v primárnom súbore s konkrétnou hodnotou vyhľadávacieho kľúča.
- **Riedky index** = záznamy primárneho súboru sú zoradené do poradia daného kľúčom. Obsahuje len vybrané hodnoty kľúča – napr. kľúče prvých záznamov v blokoch (**separátory**).



Riedky index umožňuje efektívne riešiť operácie v rámci daného bloku vymedzeného indexom.

- **Primárny index** = riedky index k súboru, ktorý je najprv *usporiadaný* podľa istého kľúča
- **Sekundárny index** = hustý index určený pre riešenie odpovedí na dotazy založené na inom vyhľadávacom kľúči než na kľúči určujúcom primárny index; môže ich byť viac
- Ku každému indexu 1. úrovne je možné vypracovať (riedky) index 2. úrovne, ktorý ukazuje na bloky indexu 1. úrovne. Počet úrovní je možné bez obmedzenia zvyšovať.



Implementácia indexov:

- pomocou **tabuliek** (riadne / hashované indexy)
    - na miesto záznamu v primárnom súbore pristupujeme pomocou ukazateľa združeného s hodnotou vyhľadávacieho kľúča v tabuľke dvojíc <kľúč, ukazateľ>
    - zrýchlenie vyhľadávania v tabuľke sa dosahuje
        - triedením pri riadnych (lineárnych) indexoch
        - hashovaním pri hashovaných indexoch
- pomocou **bitovej mapy** – ku každej hodnote vyhľadávacieho kritéria sa vypracuje bitová mapa (vektor) – umožňuje klásť dotazy na bázi viacerých vyhľadávacích kľúčov (použitie: SQL)
    - umiestnenie záznamu zistíme podľa pozície konkrétneho bitu v bitovej mape
    - hodnota bitu určuje, ktoré záznamy spĺňajú vyhľadávacie kritérium
    - takéto indexy sú použiteľné, len ak relevantné atribúty nadobúdajú málo hodnôt

		gender	rating			
0	Novák	m	1	A-217	Brighton	750
1	Krejčí	f	2	A-101	Downtown	500
2	Petrů	m	2	A-110	Downtown	600
3	Petr	m	1	A-215	Mianus	700
4	Staudek	m	2	A-102	Perryridge	400
5	Smith	f	1	A-201	Perryridge	900
6	Boháčková	f	1	A-218	Perryridge	700
7	Uhrinová	f	3	A-222	Redwood	700
8	Szabo	m	1	A-305	Round Hill	350

1	100101101
2	011010000
3	000000010

Bitová mapa „rating"

m	101110001
f	010001110

Bitová mapa „gender"

- pomocou **stromu** – ukazatele na pozíciu záznamu v súbore sú zviazané s hodnotami vyhľadávacieho kľúča (ako pri tabuľke), ale hodnoty kľúča sú v uzloch stromu
    - môžu byť len v listoch stromu (B+ stromy)
    - alebo vo všetkých uzloch (B stromy)
    - jediná podmienka: strom musí byť vyvážený

Index-sekvenčné súbory = indexujú sa len bloky (**stopy, valce, zväzky**), po nájdení správneho bloku sa ďalej prehľadáva sekvenčne.

- Index stôp na počiatku každého valca
- Index valcov na počiatku každého zväzku
- Index zväzkov (master index) na počiatku prvého zväzku – pri otvorení súboru ide do RAM

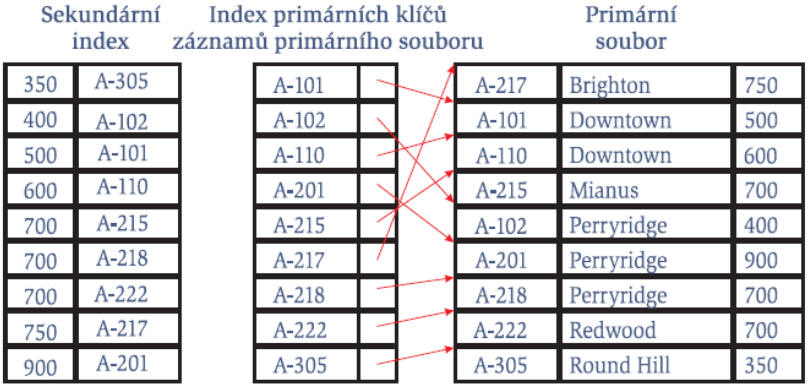
Takýto typ súboru obsahuje:

- *primárne dáta* triedené podľa vyhľadávacieho kľúča
- *primárny index* – viacúrovňový, hierarchicky usporiadaný:

    - úroveň 0 = primárne dáta
    - úroveň 1 = riedky index blokovaných primárnych dát
    - úroveň 2 = riedky index blokovaných indexových párov úrovne 1 atď.
- a *oblasť pretečenia* ktorá sa využíva pri dynamických súboroch na riešenie operácie INSERT – funguje presne ako súbor aktualizácií pri usporiadanom sekvenčnom súbore
    - s pridávaním dát do súboru sa prístup k záznamom spomaľuje – pretokové oblasti sa prehľadávajú sekvenčne, indexy sa viac zanorujú => pri rastúcom súbore **klesá výkon**
    - reorganizácia súboru je časovo náročná a behom nej sú dáta nedostupné
    - zrušenie záznamu môže zanechať dieru v pamäti
    - celkovo nevhodné pre dynamické súbory a súbory, ktoré musia byť k dispozícii 24/7
    - alternatíva: viz. lekcia 7

Indexované súbory umožňujú: klásť na súbor viac dotazov + logicky vyčleniť podmnožiny záznamov v súbore, ktoré spĺňajú zadanú podmienku bez nutnosti sekvenčného prechádzania celého súboru

- primárny súbor s dátami je obvykle netriedený
- vypracujú sa **husté** indexy – vyhľadávacie kľúče – pre všetky možné dotazy
- súbor je vždy usporiadaný podľa **jedného kľúča**
    - podľa primárneho kľúča sa vytvorí triedený primárny index a ten definuje aj usporiadanie súboru
    - sekundárne kľúče by znamenali sekvenčné prechádzanie => nepoužívajú sa
- pri modifikácii primárneho súboru sa musí modifikovať aj index
- **priame indexovanie:** index viaže s hodnotou kľúča ukazateľ záznamu v súbore
- **nepriame indexovanie:** sekundárny index viaže s hodnotou kľúča hodnotu primárneho kľúča



Lekcia 7 – hierarchické indexy (viz. wikipedia: list of data structures)

Index-sekvenčná organizácia nestačí – radšej organizácia súboru s indexom na bázi (B) **stromu:**

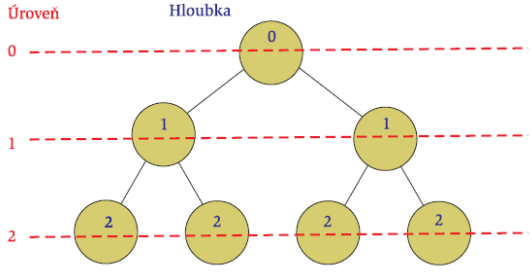
- sekundárny súbor s indexom sa pri vkladaní a rušení záznamu reorganizuje pomocou malých lokálnych zmien v strome
- teda nie je nutné robiť zmeny v rozsiahlom primárnom súbore
- nevýhody: vyššia **priestorová** náročnosť – index je uložený na vonkajšej pamäti
  dodatočná **časová** náročnosť pri štiepení a zlievaní uzlov grafu
- napriek tomu **výhody** indexov budovaných na bázi B stromu **prevažujú** (použitie: NTFS)

Graf G = (V, E)

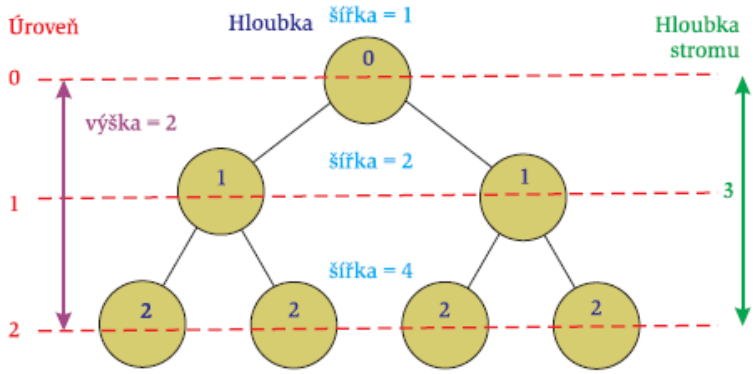
- V je konečná neprázdna množina **uzlov**
- E je množina **hrán** prepojujúcich uzly z množiny V
- **Orientovaný graf** obsahuje orientované hrany - usporiadané dvojice uzlov (v, w)
- **Neorientovaný graf** je zvláštny prípad orientovaného grafu, kde ku každej orientovanej hrane (v, w) existuje aj orientovaná hrana (w, v) a teda na poradí nezáleží
- Uzol v je **susedný** s uzlom w, keď v grafe existuje hrana (v, w)
- **Arita** = počet hrán, ktoré vystupujú z uzlu
- **Paralelné hrany** začínajú a končia v rovnakom uzle
- **Cesta** = postupnosť uzlov, medzi ktorými vedie postupnosť hrán
- **Dĺžka cesty** = počet hrán, ktoré cesta obsahuje, t.j. počet uzlov – 1
- **Cyklus** = cesta, ktorá začína a končí v rovnakom uzle
- **Slučka** = cyklus tvorený jedinou hranou
- Cesta je **jednoduchá**, keď sa žiadny z uzlov na ceste neopakuje
- **Ohodnotený graf** obsahuje ohodnotené hrany – majú priradenú istú **váhu**
- **Súvislý graf** = z každého uzla existuje aspoň jedna cesta do ľubovoľného iného uzla
- **Acyklický graf** = žiadna cesta v grafe nie je cyklom
- **Jednoduchý graf** je orientovaný a acyklický, neobsahuje násobné (paralelné hrany)
  - Označuje sa ako **DAG** – directed acyclic graph
- **Strom** = neorientovaný súvislý acyklický graf – pokiaľ nie je povedané inak, chápeme ho ako **koreňový strom** = uzly na každej ceste sú radené do vzťahu **rodič-potomok**
  - Každý potomok má práve jedného rodiča
  - Jediný vrchol, ktorý nemá rodiča – **koreň**
  - Vrchol, ktorý nemá potomka – **list**
  - Uzol, ktorý má aspoň 1 potomka a nie je koreň = **vnútorný**
  - Cesta z koreňa do listu = **vetva** stromu
- **Podgraf** = podmnožina hrán grafu
- **Kostra** = podgraf, ktorý obsahuje všetky uzly ako pôvodný graf, ale nemá niektoré hrany
- **Podstrom** = časť stromu tvorená jedným jeho uzlom a všetkými jeho potomkami

Charakteristiky stromu:

- **Hĺbka uzlu** hu = dĺžka cesty od koreňa k uzlu
- **Úroveň** = množina uzlov s rovnakou hĺbkou
  Každá úroveň *d* obsahuje najviac k^d uzlov,
  kde k je arita stromu
- **Hĺbka stromu** K = počet úrovní stromu



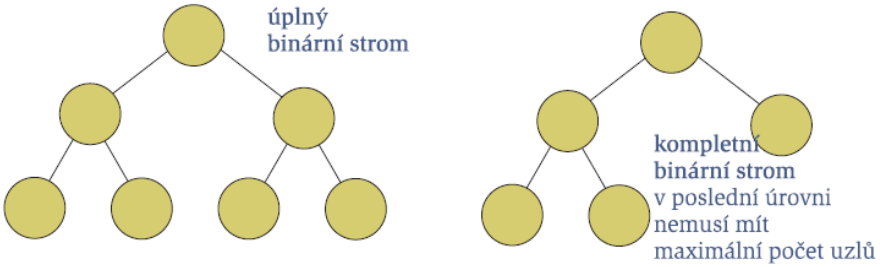
- **Šírka stromu** na istej úrovni = počet uzlov na danej úrovni
- **Výška stromu** h = K – 1 je maximálna hĺbka uzlu v stromu
    - Strom má **najmenšiu možnú výšku**, keď na každej úrovni (okrem poslednej) má plný počet uzlov – žiadúca vlastnosť, lebo zaisťuje minimálne dĺžky ciest k uzlom



- **Vyvažovanie** stromu = rovnomerné rozkladanie uzlov v jednotlivých úrovniach, aby sa dosiahla najmenšia možná výška pri danej arite a počte uzlov
- Strom je **usporiadaný**, keď sú všetci priami potomkovia každého uzlu zoradení

Binárny strom = konečná množina uzlov, ktorá je buď prázdna alebo obsahuje koreň a dva disjunktné binárne podstromy – ľavý a pravý.

- Počet uzlov úplného binárneho stromu o K úrovniach (výške h): $n = \sum_{i=0}^{h} 2^i = \sum_{i=0}^{K-1} 2^i$
- Opačná úloha – počet úrovní K binárneho stromu o n uzloch:
    - Ak je vyvážený, tak $K_{min} = \log_2 (n+1)$, $K_{max} = n$

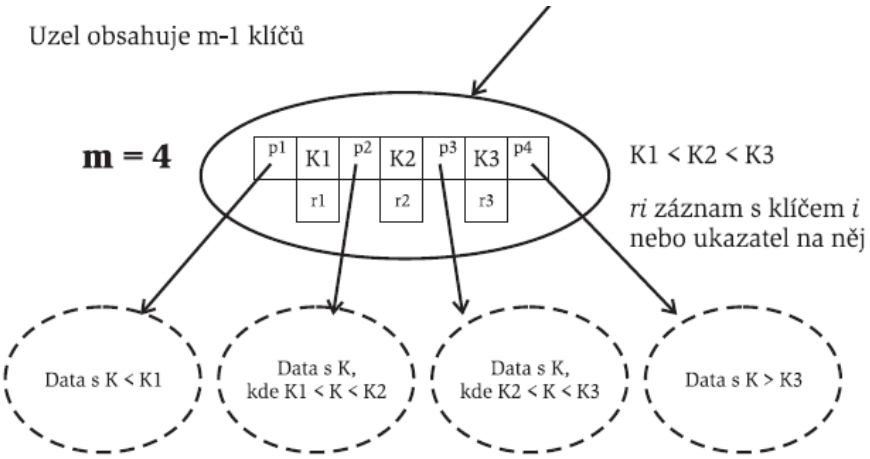


Binárny vyhľadávací strom, BVS:

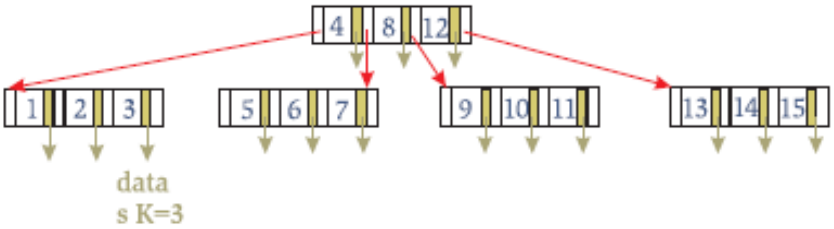
- má uzly usporiadané tak, aby bolo možné rýchlo vyhľadať hodnotu v uzle – kľúč
- niekedy sa rovnaké hodnoty vyskytujú aj v uzloch aj v listoch – strom je *redundantný*
- ak uvažujeme neredundantný BVS s n uzlami, najhoršia doba vyhľadávania odpovedá počtu úrovní stromu, teda $\log_2 (n+1)$
- ale je nutné ho udržovať **vyvážený** (AVL strom, červenočierny strom)

Prakticky ale (kvôli obrovskému počtu záznamov) nestačia binárne stromy – používajú sa **m-árne:**

- vo vnútornom uzle je viacero kľúčov, ktoré sú usporiadané
- zväčšovaním arity pri zachovaní počtu uzlov dosiahneme menšiu výšku stromu => znižovanie počtu diskových operácií pri prechode stromom => *efektívne pre implementáciu indexu*



- **maximálny počet uzlov** v m-árnom vyhľadávacom strome výšky h:
  - v jednotlivých úrovniach i sú počty uzlov m^i (teda $m^0, m^1, m^2, m^3, ...$)
  - súčet prvých h+1 členov geometrickej postupnosti: $(m^{h+1} - 1) / (m-1)$
- **maximálny počet kľúčov** v m-árnom vyhľadávacom strome výšky h:
  - každý uzol obsahuje až m-1 kľúčov, čiže n = (m-1) * počet uzlov = $(m^{h+1} - 1)$
- **minimálna výška** m-árneho vyhľadávacieho stromu s n kľúčmi je h = $\log_m (n+1)$
  - aby sme dosiahli vyhľadávanie v O($\log_m$ n), potrebujeme mať strom **vyvážený**
  - vyváženosť dosahujeme pomocou technológie **B stromov**



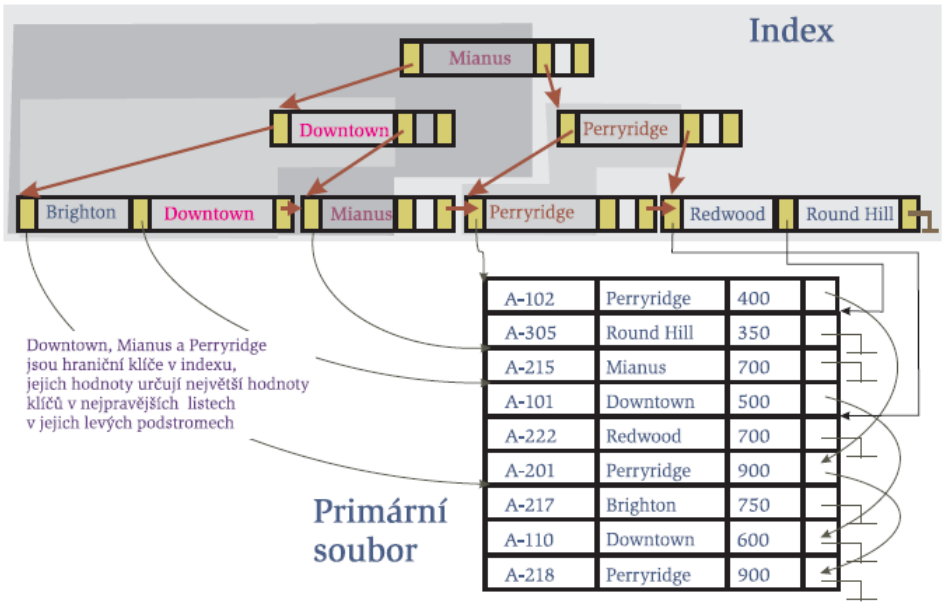
B-strom nie je binárny strom! Je to m-árny vyhľadávací strom s obmedzeniami:

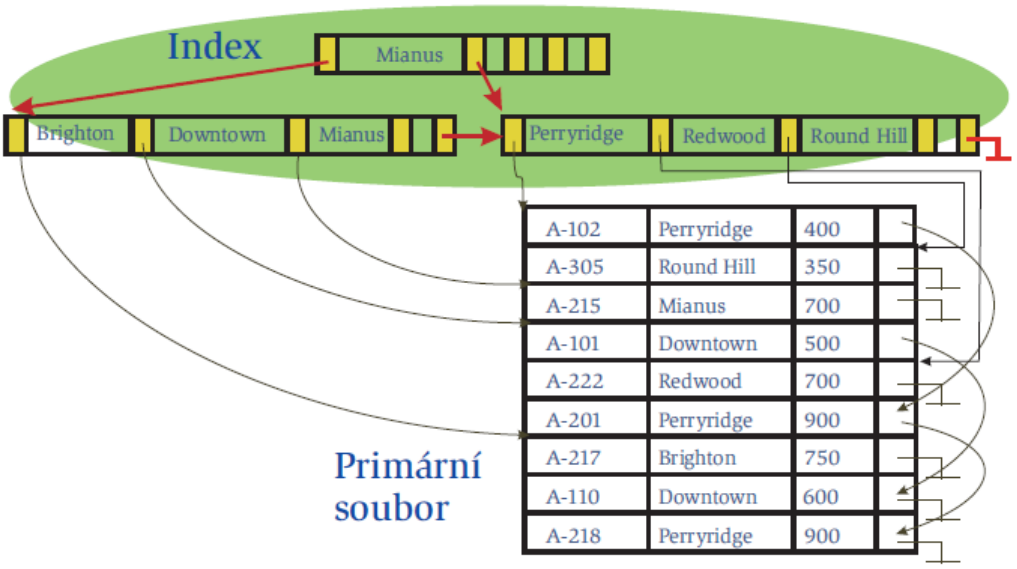
- Každý uzol až na koreň a listy je obsadený aspoň z polovice
- Každý uzol má najviac m potomkov
- Koreň má aspoň dvoch potomkov (ak nie je list)
- Uzol s g <= m potomkami obsahuje g-1 vyhľadávacích kľúčov
- List má aspoň (m-1)/2 kľúčov
- Všetky listy sú na rovnakej úrovni
- Je **neredundantný** – každý kľúč sa v ňom vyskytuje najviac raz
- Viz. príklady: slidy 42-49

B+ strom je redundantná varianta B stromu

- Záznamy s dátami sú adresované len z listov, všetky ostatné uzly sú navigátory
- Listy sú reťazené podľa poradia kľúčov
- Jednoduchšia implementácia ako pri B strome, jednoduchšie vkladanie a rušenie uzlov
- Ak je v súbore n hodnôt kľúča, cesta od koreňa do listu nie je dlhšia ako $\log_{m/2}$ n

B+ stromy: príklad pre m = 3 a m = 5





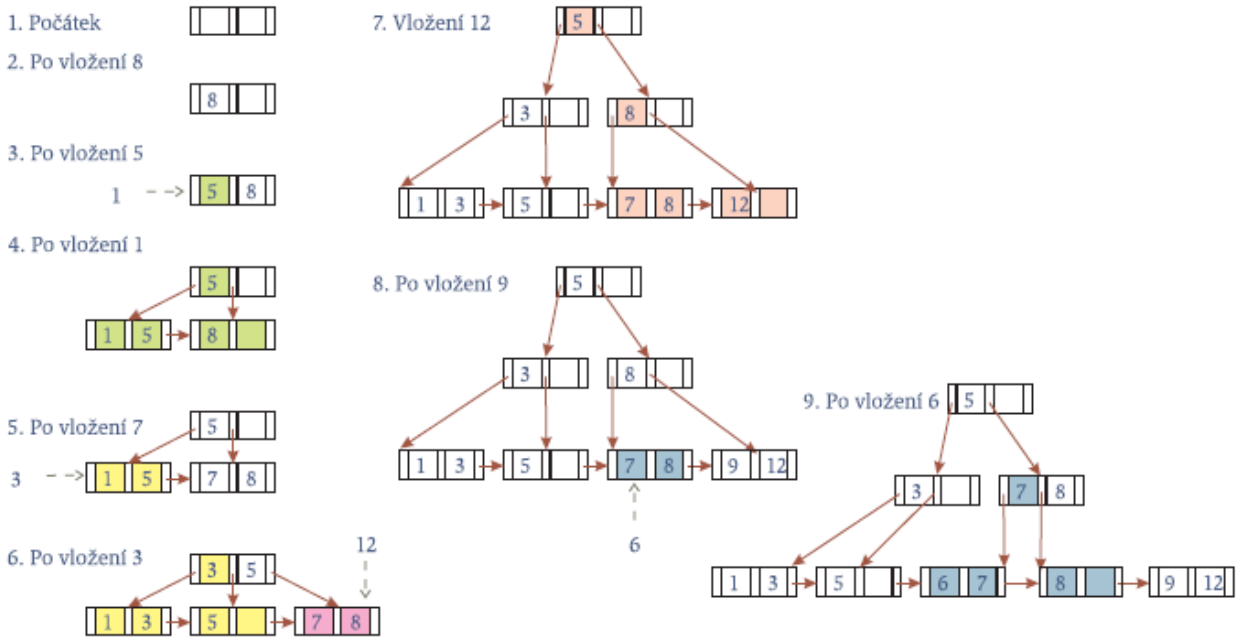
Príklad: vypočítajte optimálny rád B+ stromu za podmienok: jeden kľúč má dĺžku V = 9 B, diskový blok má dĺžku B = 512 B, ukazateľ na záznam s dátami má dĺžku R = 7 B, ukazateľ na indexový záznam má dĺžku P = 6 B.

Vnútorné uzly sú umiestňované po jednom v jednom diskovom bloku a každý obsahuje *m* ukazateľov a *m-1* kľúčov, takže rád vnútorných uzlov bude $m * P + (m-1) * V <= B$
$6m + 9(m-1) <= 512$, teda m = 34.

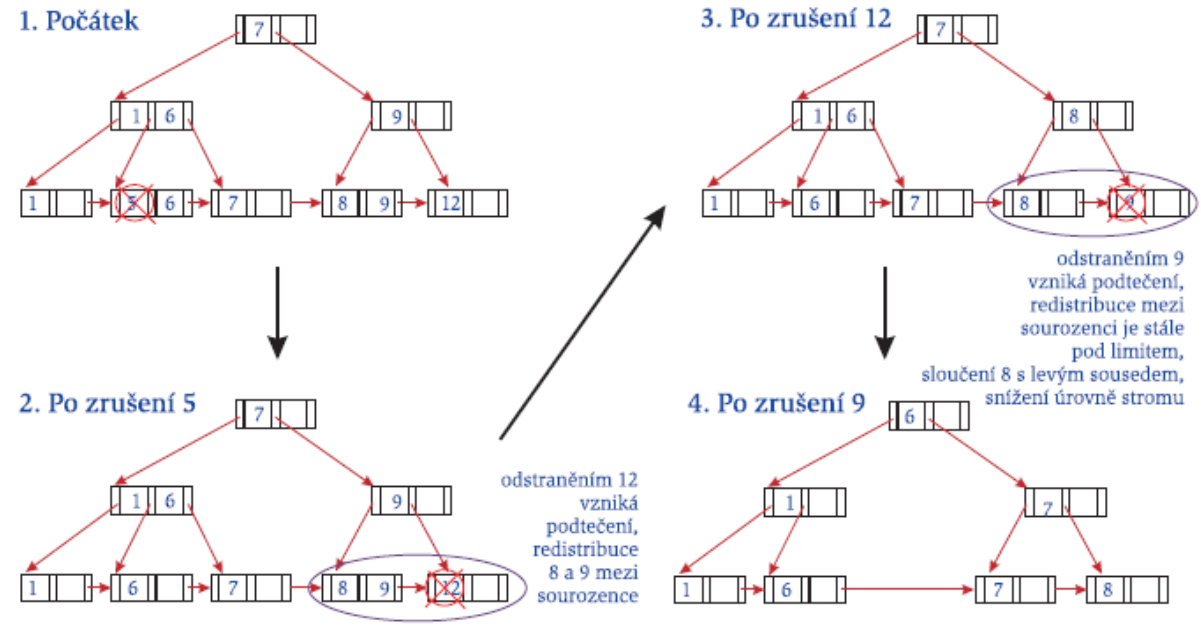
Podobne pre rád listov platí: $m_l * (R + V) + P <= B$
$16m_l + 6 <= 512$, teda $m_l = 31$.

Ak budú nelistové uzly plné zo 69%, budú obsahovať 34 * 0,69 = 23 ukazateľov, teda 22 kľúčov. Ak budú listové uzly plné zo 69%, budú obsahovať 31 * 0,69 = 21 ukazateľov dát. T.j. na 0., 1., 2., a 3. úrovni bude počet uzlov: 1, 23, 529, 12167,
t.j. kapacita 4-úrovňového indexu typu tohoto B+ stromu pokryje 12167 * 21 = 255 507 záznamov.

Príklad: m = 3, vloženie 8, 5, 1, 7, 3, 12, 9, 6



Zrušenie 5, 12, 9:



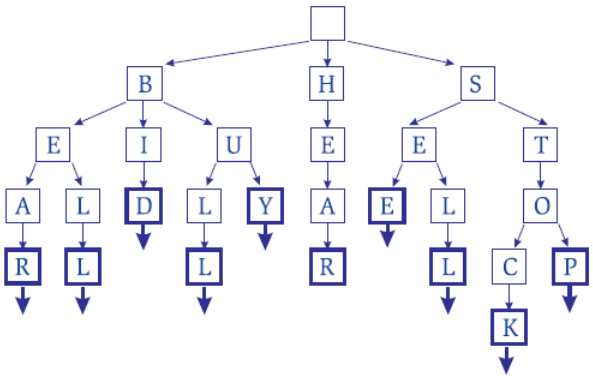
Trie (vyslov „try“) – information re**trie**val – je m-árny strom (**nie** vyhľadávací!), kde koreň je prázdny reťazec a každý uzol reprezentuje jedno slovo alebo prefix slova. Kľúče teda musia byť deliteľné na menšie jednotky (znaky, cifry, ...), pretože sú uchovávané na ceste z koreňa do listu. Uzol vzdialený k hrán od koreňa reprezentuje slovo dĺžky k.

Môže byť použitý aj ako index – v listoch potom budú ukazatele na záznamy s odpovedajúcimi hodnotami kľúča.

- Použitie: prehľadávanie rozsiahlych textov, vyhľadávanie vzorov, konštrukcia adresárov pri extenzibilnom hashovaní
- Výhody:
    - výška trie je daná dĺžkou najdlhšieho kľúča => dĺžka hľadania v trie je priamo úmerná dĺžke kľúča, nie log počtu uzlov v strome
    - neúspešné hľadanie môže skončiť na ktorejkoľvek úrovni
    - priestorovo výhodné pre množstvo krátkych kľúčov
    - strom nie je nutné vyvažovať
    - pri vkladaní sú rýchlejšie ako hashovacie tabuľky

        - Index slov

            bear, bell, bid, bull, buy, hear, see, sell, stock, stop

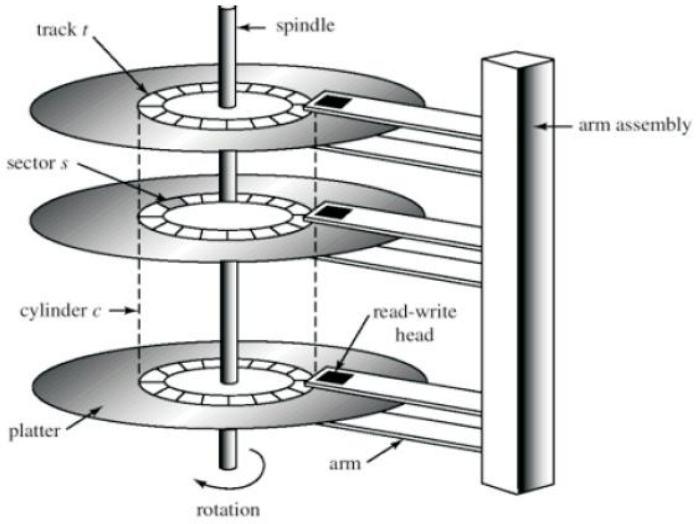


Trie môžu byť aj m-árne (tzv. **radix trie**), kde každý uzol je m-prvkové pole, alebo **komprimované**, kde sa eliminujú uzly s jediným potomkom – vzniká pole.

Lekcia 8 – vonkajšie pamäte

Pevný disk – relatívne lacná a spoľahlivá sekundárna pamäť

- dáta sú uchovávané na povrchu magnetických **diskov**
- niekoľko magnetických diskov nasadených na rotujúcu os tvorí **diskovú zostavu** (disk pack)
    - na jeden disk jedna **čítacia/zapisovacia hlava** veľmi blízko nad povrchom
    - všetky hlavy sú spoločne na jednom **vystavovacom ramienku**
- na povrchu každého disku: sústredné trajektórie – **stopy** (desiatky tisíc)
- stopy s číslom *i* na všetkých povrchoch tvoria **valec**, skupiny valcov = **oddiely**
- stopy majú fixne zakódované do svojho povrchu delenie na **sektory**
    - najmenšia jednotka dát (4 kB), ktorú je možné čítať z / zapísať na disk
    - počet sektorov na stopu: 200 (na vnútorných) až 400 (na vonkajších stopách)
    - sektory sa združujú do **alokačných blokov**
- rýchlosť rotácie 4000 – 15000 rpm, prístupová doba 10 – 4 ms



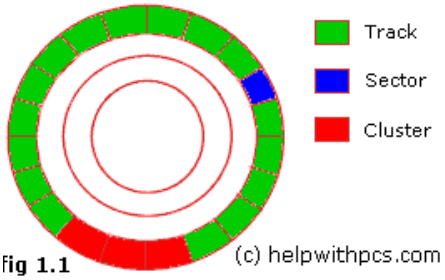


fig 1.1

Čítanie / zápis sektoru:

- hlava sa presunie nad požadovanú stopu
- čakanie na natočenie sektoru pod hlavu
- čítanie/zápis behom prechodu sektoru nad hlavou

K procesoru sa disky pripojujú pomocou I/O **zbernice**. Rôzne rozhrania (štandardy) pre prenos dát medzi počítačom a periférami: ATA, SATA, SCSI, SAS, Fibre Channel:

- **PATA** (Parallel Advanced Technology Attachment) – nový názov pre ATA, nahradené SATA
- **SATA** (Serial ATA) – sériová verzia ATA, 7 kanálov namiesto 40, rýchlejší prenos dát
- **SCSI** (Small Computer System Interconnect) – paralelný zbernicový port (8 zariadení), rýchle, spoľahlivé – servery
- **SAS** (Serial Attached SCSI) – sériové rozšírenie SCSI spätne kompatibilné so SATA diskami
- **FC** (Fibre Channel) – primárne určené pre **sieťové prepojenie** mnohých diskov so servermi

Prenos dát po zbernici ovládajú **radiče** (controllers) – radič disku a riadiaca jednotka disku:

Radič disku (disk controller):

- Na strane I/O zbernice
- Poskytuje **rozhranie** pre ovládanie softwarom
- Akceptuje príkazy (read, write) od driveru, poskytuje mu stavové informácie
- Počíta a kontroluje **kontrolné súčty** čítaných/zapisovaných dát
- Overuje kvalitu zápisu sektoru jeho čítaním

Riadiaca jednotka disku (disc control unit):

- Ovláda diskový mechanizmus, aby sa vykonala požadovaná operácia
- Má vstavanú cache pamäť

Dáta sa prenášajú:

- Medzi povrchom disku a cache pamäťou v rytme rotácie disku
- Medzi cache pamäťou a radičom disku rýchlosťou I/O zbernice
- Medzi radičom disku a hlavnou pamäťou pomocou DMA
- **Šírka pásma** = počet prenesených B / doba od zadania požiadavky do jej ukončenia

Možnosti pripojenia diskov **do siete**:

- **SAN** (Storage Access Networks) – uzly siete sú disky, ku ktorým pristupujú servery
    - Disky sa pripájajú a odpájajú dynamicky
    - File system sprístupneného disku rieši klientská strana
- **NAS** (Network-Attached Storage) – okrem úložného priestoru zabezpečuje aj file system
- **iSCSI** (Internet SCSI) – príkazy SCSI sú prenášané Internet Protocolom cez LAN

Plánovanie činnosti disku:

- cieľ: minimalizácia *doby vystavenia* – prechod na valec so stopou s adresovaným sektorom
- *doba rotačného omeškania* – prechod adresovaného sektoru pod hlavu – je konštantná
- požiadavky vo fronte riešia rôzne politiky:
    - FCFS, **SSTF** (Shortest Seek Time First, obdoba SJF) - prirodzené
    - **SCAN** (výťah – ramienko sa opakovane pohybuje z okraja až úplne do stredu a zase späť a postupne vybavuje požiadavky) a **C-SCAN** (jednosmerný výťah – pri návrate späť nevybavuje požiadavky) – vhodné pri veľkej záťaži disku
    - **LOOK** (ako SCAN, ale nejde vždy úplne do stredu/na kraj, ale zmení smer, keď vybaví poslednú požiadavku v danom smere) – častá implicitná voľba

Formátovanie disku: (holé zariadenie, raw device – len pole blokov dát, musí sa sformátovať)

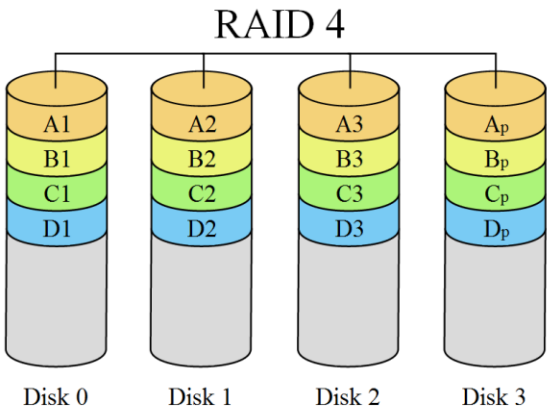
- **Fyzické** (low-level) – delenie disku na sektory pomocou špeciálnych dát čítaných / zapisovaných radičom
- **Logické** (partitioning) – delenie disku na jednu alebo viac **skupín valcov** – oddielov (partitions) + vytváranie súborového systému

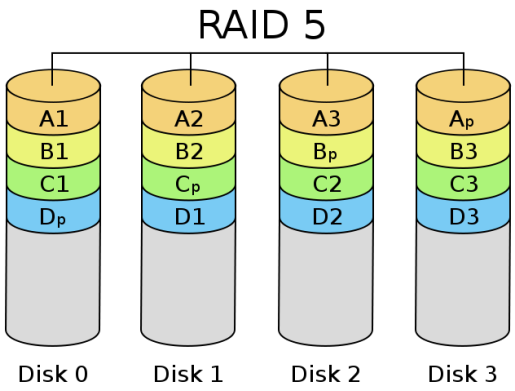
RAID (Redundant Arrays of Independent Disks) = pole fyzických diskov riadené tak, že poskytuje dojem jedného logického disku:

- S veľkou kapacitou a rýchlosťou: viac diskov uchováva/poskytuje dáta paralelne
    - **disk striping** – diskový priestor sa **rozdelí** do samostatne adresovateľných jednotiek
        - bit-level striping = delenie bitov každého disku medzi samostatné disky
        - block-level striping – systém s n diskami, i-ty blok súboru sa zapisuje na disk (i mod n) + 1
- S veľkou spoľahlivosťou:
    - **mirroring** (zrkadlenie diskov) – dáta sú zapisované redundantne na viacero diskov – t.j. disky sa duplikujú – 1 logický na 2 fyzické (aspoň 1 musí ostať funkčný)
    - **block interleaved parity** – samoopravné kódy pri čítaní dát

Úrovne RAID:

- **0** – len data striping, žiadne zrkadlenie ani odolnosť voči poruchám – porucha jedného disku = strata dát, žiadna kontrola parity, vysoká priepustnosť pri čítaní a zapisovaní
- **1** – len mirroring – paralelný zápis, číta sa z rýchlejšieho z diskov; pre kritické aplikácie
- **2, 3** – bit-level striping + kontrola parity, napr. Hammingov kód
- **4** – konfigurácia RAID 0 + jeden disk obsahuje paritné bloky (viz. ďalej), nahradené RAID 5
- **5** – aspoň 3 disky, paritné dáta sú už rozložené po všetkých diskoch => čítanie zo všetkých diskov naraz a vyrovnaná priepustnosť, úroveň zabezpečenia je zhodná s RAID 4



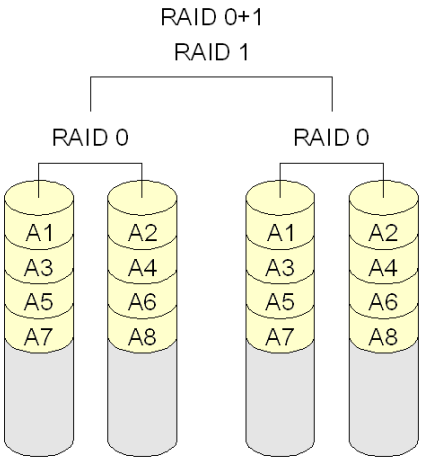


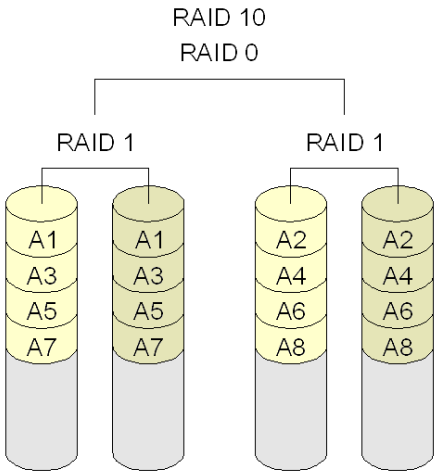
Kontrola parity: pomocou **XOR**, napr. dva disky obsahujú dáta: Disk 1: 01101101, Disk 2: 11010100. Ich parita: 10111001 je uložená na disku 3.

- Čítanie: len blok s dátami
- Čítanie pri chybe disku: čítajú sa bloky so všetkých diskov + paritný blok, a stratená hodnota sa dopočíta – napr. ak zlyhá disk 2, zoberie sa 01101101 XOR 10111001 = 11010100
- Zapisovanie: prečíta sa prepisovaný blok, prečíta sa odpovedajúci paritný blok, zapíše sa blok, zapíše sa nový paritný blok = pôvHodnBloku XOR nováHodnBloku XOR pôvHodnParitBloku

Hybridné úrovne RAID:

- 0+1, Mirrored Stripes – dve delenia na bázi RAID 0 a nad nimi zrkadlenie na bázi RAID 1 – ak vypadne jedna sada RAID 0, výpadok je akceptovateľný, (nepočíta sa parita)
- 1+0, Striped Mirrors – zrkadlené disky na bázi RAID 1 a nad nimi delenie na bázi RAID 0 – môže vypadnúť jedna sada RAID 1 (nepočíta sa parita), ale vyšší výkon aj rýchlosť ako RAID 5





SSD – Solid State Disk – pamäťové médium, ktoré ukladá dáta na flash pamäť – neobsahuje pohyblivé mechanické časti ako pevný disk

- Čip na rozhraní kvôli kompatibilite emuluje rozhranie používané pre HDD (typicky SATA)
- Výhody: sú menej náchylné na nárazy a otrasy, nehlučné, ľahšie, nižšia spotreba, nižší čas na alokáciu dát, vyššia prenosová rýchlosť
- Nevýhody: obmedzená životnosť maximálnym počtom zápisov do rovnakého miesta, niektoré OS (Win) k nim pristupujú kvôli kompatibilite ako ku klasickým HDD – obmedzenia

Ternárne (terciálne) **pamäte**

- **Pásky**: v porovnaní s diskami: lacnejšie, s väčšou kapacitou, ale pomalšie, sekvenčný prístup. Typy: LTO (Linear Tape-Open), DLT (Digital Linear Tape). Pásky sú **append only**. Nemajú FS.
- **Disketa**: tenký pružný magnetický disk v plastovom puzdre.
- **Magnetooptický disk**: laserový lúč zapisuje/číta z kotúča pokrytého elektromagnetom.
- **Optický disk**: dáta sú laserom zapísané na stopu zatočenú do špirály, ktorá vedie od stredu až po okraj disku. Čítanie: mechanika roztočí disk, plocha je (od stredu) osvetľovaná laserovou diódou: 0 pri nedostatku odrazu laserového svetla (priehlbinka v špirále), 1 pri odraze.
- **WORM**: tenký hliníkový film medzi 2 sklenenými alebo plastovými povrchmi. Jednorázový zápis bitu: laser vypáli vo filme dierku. Spoľahlivá a trvalá technológia.
- Budúci rozvoj: **holografické pamäte**, **MEMS** (micro-electromechanical systems)

Rýchlosť ternárnych pamätí ovplyvňuje:

- **Šírka pásma**, bandwidth – 2 aspekty:
    - Prenosová šírka pásma – priemerná rýchlosť *počas* prenosu (počet prenesených B)
    - Efektívna šírka pásma – priemerná rýchlosť prenosu vrátane I/O operácií
- **Latencia** – doba potrebná pre vyhľadanie dát po prístupe
    - Napr. disk: vystavenie, natočenie vs. páska: pretočenie na požadovaný blok

Lekcia 9 – súborové systémy

Prostriedok pre **organizáciu dát na vonkajšej pamäti** + pre **správu miesta** (blokov) **v pamäti**

- Premosťujú *nízkoúrovňový pohľad* na súbor (pole blokov dát na disku) a *užívateľský pohľad* (kolekcia dátových záznamov)
- Poskytujú systém pomenovania súborov a ich správy:
    - Vytváranie a rušenie (operácie create a delete) súborov a adresárov
    - Skladovanie na vonkajšej pamäti, ich vyhľadanie a zobrazenie do operačnej pamäte
    - Manipulácia so súbormi (read, seek), ich zmena (write), archivácia...
- Poskytujú jednotnú podporu I/O pre rôzne vonkajšie pamäte
- Minimalizuje riziko straty či poškodenia dát

Štruktúra súboru z pohľadu OS:

- Voľná = postupnosť B ukladaná sekvenčne, jednoduchá správa
- Rigidne formátovaná = postupnosť záznamov – indexy na bázi B stromov, hašovanie

Pomenovanie súborov:

- Vonkajšie: textové, trvalé po dobu existencie súborov (napr. file.txt)
    - môže mať zakázané znaky (napr. /)
    - + aplikujú sa iné obmedzenia (max. dĺžka mena, case senzitivita...)
- Vnútorné: číselné, **fd** (file descriptor - UNIX), **fh** (file handle – Windows): informácia pre OS pri sprístupnení (otvorení) súboru a počas práce s ním – prideľuje sa dynamicky

Atribúty súboru = meno (vonkajšie pomenovanie), typ (prípona), umiestnenie, veľkosť, prístupové práva, id vlastníka, čas a dátum vytvorenia... Spolu s dátami sú uchovávané v zvláštnej adresárovej štruktúre – **file control block (FCB).**

OS si udržuje informácie o otvorených súboroch – v tabuľkách otvorených súborov

- procesu (1/proces) – čo s daným otvoreným súborom robí proces
- a systému (1/systém) – čo platí o každom otvorenom súbore nezávisle na procesoch, ktoré s ním pracujú

Záznam o súbore v tabuľke otvorených súborov **procesu** obsahuje: ukazateľ na práve sprístupňovaný záznam, prístupové práva.

Záznam o súbore v tabuľke otvorených súborov **systému** obsahuje: umiestnenie na disku, veľkosť, údaj o čase sprístupnenia, čítač otvorení (++ pri open, -- pri close; odstrániť je možné len súbor, ktorý nemá otvorený žiadny proces), či je súbor *zamknutý* (zámok byť poradný alebo povinný).

Typy súborov:

- textové, binárne, súbory konkrétnych aplikácií (Excel, Word – nepodporuje priamo OS)
- deklarujú sa príponou oddelenou bodkou od mena
- špeciality v UNIXe: s tlačiarňou (IO zariadením) môžem pracovať ako so súborom

Adresár: špeciálny typ súboru, ktorý umožňuje efektívnu organizáciu súborov + obsahuje informácie o uchovávaných súboroch.

- **1-úrovňový:** jediný adresár v celom systéme – nemožné zoskupovať súbory, nemožné ich unikátne pomenovať – typicky v telefóne, kamere...
- **2-úrovňový:** pri vzniku multiužívateľských OS – separácia adresárov jednotlivých užívateľov; súbor pomenujeme jeho *prístupovou cestou* (user1/file.txt, user3/file.txt je OK)
- **Hierarchický:** stromová štruktúra, efektívne vyhľadávanie, nezávislé pomenovanie
    - Pracovný adresár = dynamicky určený východzí bod; sme v ňom v aktuálnom okamihu a na súbory pod ním sa odkazujeme **relatívne**
    - **Absolútna** cesta = začína v koreni stromu
- **Acyklický:** nie je strom; umožňuje zdieľať súbory pomocou viacerých *ukazateľov* na jeden súbor => **aliasing**: jeden objekt môže byť sprístupnený pod rôznymi pomenovaniami
    - Pri odstránení ukazateľa na objekt môžu ostať ukazatele na už prázdne miesto
    - Riešenie: čítač ukazateľov pri objekte – zmazanie je možné, len ak je čítač rovný 0
    - Nutné zabrániť zacykleniu: napr. povolí sa ukazateľ len na súbor, nie na adresár

File system **mounting** – súborový systém (na prenosnom médiu, resp. na disku, ktorý bol dosiaľ nedostupný) sa musí v danom mieste (mount point) pripojiť k aktuálnemu súborovému systému.

Zdieľanie súborov – prirodzená požiadavka, v multiužívateľskom OS – ale nutné nástroje ochrany:

- **DAC** – Discretionary Access Control, voliteľné riadenie prístupu
    - **vlastník** súboru má možnosť určovať, čo je možné so súborom robiť (rwx) a kto to môže robiť (ugo) – UNIXovský model, alebo **ACL** vo Win
    - jediný nedostatok: zradca z group si skopíruje súbor a nastaví mu rwx aj pre others
- **MAC** – Mandatory Access Control, povinné riadenie prístupu
    - **admin** obmedzuje prístup k súborom – jednotná bezpečnostná politika

Prístup k vzdialeným súborom pomocou sieťových služieb: manuálne: ftp, poloautomaticky: www, automaticky: Remote File Systems, napr. NFS (UNIX), CIFS (Win) pre model klient/server.

Implementácia adresára:

- FCB – lineárny zoznam mien súborov s ukazateľmi na bloky dát
    - Spracovanie dotazu v O(n) pri neusporiadaní adresára, napr. FAT v MS Windows
    - Spracovanie dotazu v O(log n) pri usporiadaní adresára – ale veľká dodatočná réžia
- B+ strom – napr. NTFS v MS Windows, logaritmická zložitosť vyhľadávania
- Hašovaná tabuľka – vyhľadávanie v O(1), ale môžu nastať kolízie

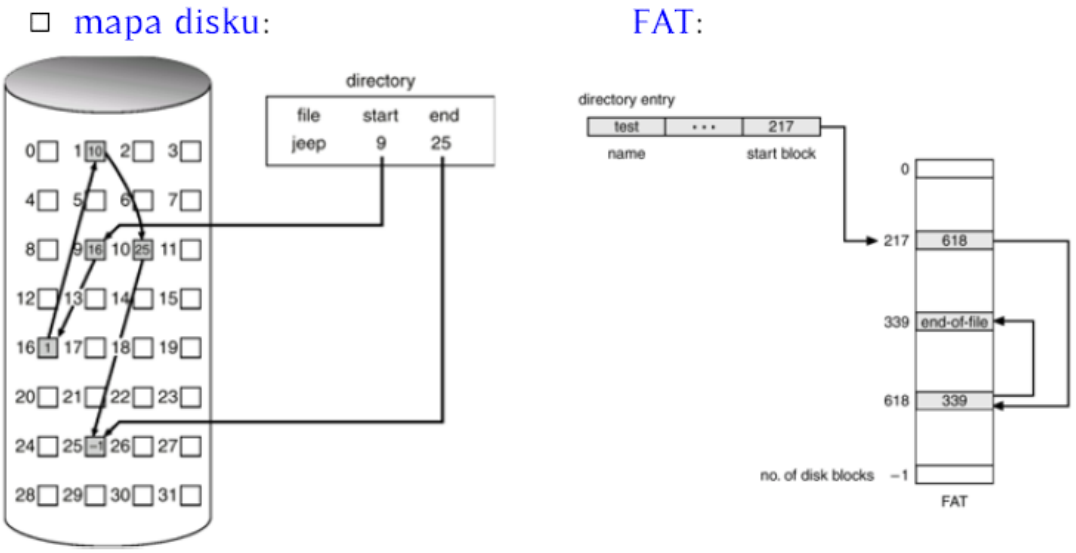
3 metódy prideľovania pamäťového priestoru (alokačných blokov) súborom:

1.) Prideľovanie **súvislých** diskových priestorov (extent-based file system)

- Každý súbor zaberá množinu susedných blokov – stačí zadať bázu a dĺžku (počet blokov)
- Sekvenčný aj priamy prístup
- Malé pohyby vystavovacieho mechanizmu disku
- Hľadanie voľného priestoru – rôzne algoritmy:
    - First fit – použije sa prvý voľný blok, do ktorého sa súbor vojde (prakticky najlepšie)
    - Best/Worst fit – čo najmenší blok, ktorý postačuje / čo najväčší blok
    - Dochádza k **externej fragmentácii** – nutná reorganizácia
- Súbory nemôžu rásť, a ak sa vopred alokuje viac pamäte, ako sa použije, časť sa nepoužíva = **interná fragmentácia**

2.) **Viazané** prideľovanie diskového priestoru, tzv. **mapa disku**

- Súbor = **previazaný zoznam** alokačných blokov, ktoré môžu byť rozptýlené ľubovoľne
- Bloky sú adresované z adresára – stačí zadať bázu (ukazateľ na prvý blok súboru)
- Súbor môže ľahko rásť; ale možný je len sekvenčný prístup
- Istá časť disku je index súborov (napr. **FAT**) – obsahuje ukazatele na počiatočné bloky



- **Alokačný blok, cluster** = postupnosť susedných sektorov
    - najmenšia logická jednotka diskového priestoru, ktorá môže byť alokovaná pre súbor
- Počet alokačných blokov (1 alebo viac) – akurát pre veľkosť súboru, žiadna rezerva
- Dĺžka - 1 alokačný blok môže byť:
    - krátky, alokuje sa veľa nesusedných blokov – externá fragmentácia súboru
    - dlhý, nevyužíva sa celá jeho kapacita – interná fragmentácia

Príklad: súborový systém **FAT**:

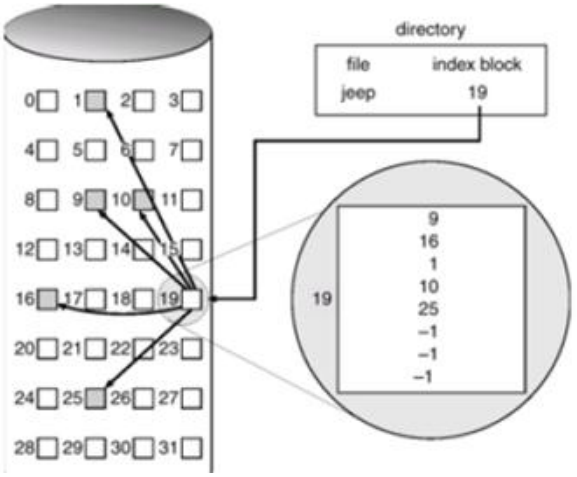
- Indexová tabuľka (File Allocation Table) je umiestnená na počiatku zväzku na pevnom mieste
- Má pevný počet položiek: napr. u FAT16 je to 2^{16} => 65536 clusterov (alokačných blokov):

Sektorov na cluster	**=> Veľkosť clusteru**	**=> Kapacita zväzku**
8 sektorov po 512 B	4 KB	*Menej* ako 65536 * 4 KB = 256 MB
16 sektorov po 512 B	8 KB	256 – 511 MB
32 sektorov po 512 B	16 KB	512 – 1023 MB
64 sektorov po 512 B	32 KB	1024 – 2047 MB
128 sektorov po 512 B	64 KB	2048 – 4095 MB
256 sektorov po 512 B	128 KB	4096 – 8191 MB
512 sektorov po 512 B	256 KB	8192 – 16383 MB

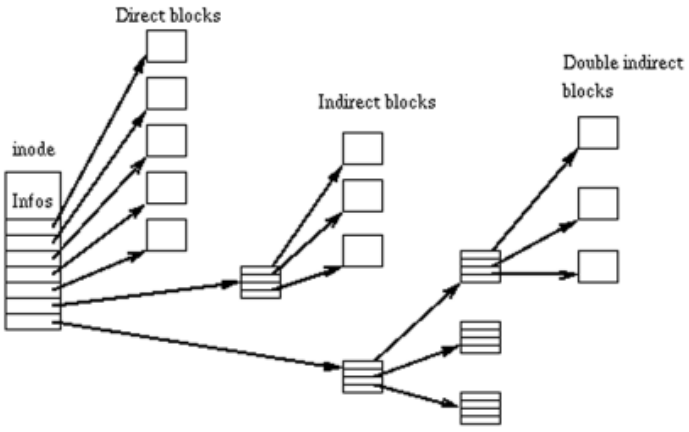
- FAT sú vhodné pre malé disky (do 0,5 GB) – napr. súbor o veľkosti 10 B zaberie na 2 GB disku 32 KB (kvôli veľkosti clustera – menší priestor sa vyhradiť nedá)
- Dáta sa čítajú z celej plochy disku, ale informácie o pozícii clusterov sú na začiatku disku => pri vyhľadávaní vo FAT tabuľke musíme stále pristupovať na disk (pomalé)
    - Efektívnejšie: kópia FAT v operačnej pamäti, alebo hašovacia tabuľka
- **FAT32** – zvýšený počet položiek tabuľky (počet clusterov) na 2^{32} (max. veľkosť súboru je 2^{32} – 1 B = 4 GB) + efektívnejšie využíva priestor na disku – pre disky s kapacitou do 8 GB vyhradzuje cluster o veľkosti 4 KB; max. podporovaná veľkosť disku je 32 GB.

3.) **Indexované** prideľovanie diskového priestoru, tzv. **mapa súboru**

- Súboru sú pridelené bloky a ukazatele na tieto bloky sú zoskupené v indexe – tzv. **indexovom bloku**
- Nemapujeme disk, ale index
- Vhodné aj pre priamy prístup, bez externej fragmentácie



- UNIX: nech réžia je úmerná veľkosti súboru – **inodes**
- V každom inode je 15 pointerov
- 12 priamo ukazuje na bloky dát súboru (prvých 48 KB súboru)
- 1 nepriamy pointer na blok pointerov, ktoré ukazujú na bloky dát súboru (ďalších 4 MB)
- 1 dvojito nepriamy pointer (ďalších 4 GB)
- 1 trojito nepriamy pointer



Správa voľnej pamäti:

- **Bitová mapa**: pre každý alokačný blok na disku 1 bit – 1 = plný, 0 = voľný
    - Ak dĺžka 1 bloku = 4 KB = 2^{12} B a kapacita disku = 1 GB = 2^{30} B,
      dĺžka bitovej mapy n = 2^{30} / 2^{12} = 2^{18} b = 32 KB, zanedbateľné
    - Mapa musí byť trvale pamätaná na disku, pri práci s diskom je zároveň v RAM
    - Nesmieme pripustiť, aby pre blok i platilo bit[i] = 1 v RAM a bit[i] = 0 na disku
    - Nutné najprv nastaviť bit[i] na 1 na disku, prideliť blok a následne bit[i] = 1 v RAM
- **Reťazenie:** ukazatele na voľné bloky, náročné hľadanie súvislého priestoru
- **Adresovanie:** analógia indexu voľných blokov

Ako zlepšiť výkon:

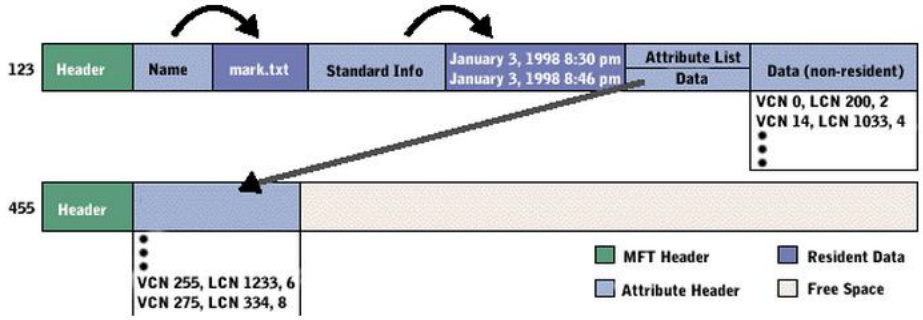
- Udržiavať diskovú cache pamäť – obsahuje často používané bloky
- Načítanie potrebných diskových blokov skôr, ako o ne aplikácia požiada
- Do časti RAM nahrať často používanú časť disku => rýchly prístup k daným dátam

NTFS (New Technology File System) – štandardný na Windows NT. Základná štruktúra: **NTFS zväzok** (volume) – *logická* úložná oblasť, ktorá sa typicky nachádza na jednom oddieli (partition) disku

- Rozlišovať oddiel – **partition** = disková oblasť, ktorá vzniká pri formátovaní disku
- a zväzok – **volume** = abstrakcia pre NTFS, označuje sa C:, D:, E:, ...
- Jeden zväzok môže byť na časti disku, na celom disku, alebo na viacerých diskoch
- Na zväzok sú ukladané súbory, adresáre, bitmapy a všetky metadáta (= informácie o zväzku)
- Každý NTFS zväzok = **postupnosť alokačných blokov** – clusterov
    - Pre disky s kapacitou > 2 GB: 1 cluster = 8 sektorov po 4 KB
    - Menej ako u FAT16: minimalizácia internej fragmentácie, ale nižšia efektivita prenosu
    - Každý cluster má 64-bitové poradové číslo: LCN (Logical Cluster Number)

Súbor v NTFS nie je len prúd B ako v MS-DOS alebo UNIXe, ale je to štruktúrovaný objekt, tvoria ho:

- **Pomenované atribúty:** meno, prístupové práva, doba vytvorenia, ...
- **Nepomenované atribúty:** dáta
- Každý súbor (alebo adresár) je definovaný záznamom v poli **Master File Table** (MFT)
    - Hlavná dátová štruktúra na jednom NTFS zväzku, definuje obsah zväzku
    - De facto relačná databáza – riadok (záznam) = súbor, stĺpce = jeho atribúty
    - Keď na zväzku vznikne nový súbor, vytvorí sa preň **bázový** záznam v MFT
    - Záznam obsahuje **atribúty** súboru
    - + ak pre súbor nestačí jeden záznam, aj odkazy na ďalšie záznamy v MFT, kde sú ukazatele na alokačné bloky, ktoré sa nevošli do bázového záznamu



- MFT záznam je tvorený postupnosťou {záhlavie_atribútu, hodnota}
    - Záhlavie atribútu: meno a jeho dĺžka
    - Hodnota: ak sa nezmestí do záznamu, umiestni sa tam len odkaz na iné miesto
- Voľné a obsadené MFT záznamy sú definované bitovou mapou
- MFT môže mať až $\mathbf{2^{48}}$ záznamov
- MFT je súbor a môže byť uložený na disku kdekoľvek
- Keďže obsahuje zoznamy všetkých komponent na zväzku, je dôležitý: má **zrkadlovú kópiu**

Každý súbor na NTFS zväzku má:

- **Vonkajšie pomenovanie** (až 255 znakov)
- **Vnútorné pomenovanie**, (fr, file reference): jedinečné ID, má 64 bitov:
    - 48-bitové *číslo súboru* (poradové číslo záznamu v MFT)
    - 16-bitové číslo, zvyšuje sa každým použitím záznamu z MFT (kontrola konzistencie)

Obsah adresárov v NTFS je organizovaný do **B+ stromu** – v jeho listoch sú ukazatele na dáta a zopakované hodnoty atribútov (meno, rozmer, doba vytvorenia...) pre rýchly výpis atribútov.

Všetky korekcie dátových štruktúr NTFS sú **transakcie** – ukladajú sa do logu (protokolu):

- Skôr, než sa dátová štruktúra zmení, zapíše sa záznam do logu, ktorý obsahuje *redo* (ako operáciu zopakovať) a *undo* (ako vrátiť pôvodný stav) informácie
- Po zmene dátovej štruktúry sa do logu poznačí potvrdzovací záznam (*commit record*), ktorý potvrdzuje úspešné vykonanie transakcie
- V prípade pádu systému je pomocou záznamov v protokole možné vrátiť dátové štruktúry do konzistentného stavu z okamihu pred výpadkom – **recovery**
- Nezaručuje sa ale obnova všetkých užívateľských súborov
- Protokol transakcií je uchovávaný ako metadátový súbor na začiatku zväzku

Bezpečnosť: každý **súbor** má uložený vo svojom zázname v MFT atribút **security descriptor**, obsahuje

- Príznak prístupových práv vlastníka súboru (**access token**)
- Zoznam prístupových práv (**access control list, ACL**), ktorý definuje práva každého užívateľa, ktorý bude pristupovať k súboru
- + bezpečnosť **zväzku**: NTFS podporuje zrkadlenie disku (RAID 1), riešenie problému vadných sektorov (HW technika: sector sparing, SW technika: cluster remapping)

Porovnanie NTFS a FAT:

- FAT: menej pamäťovo náročný, jednoduchšia štruktúra a rýchlejšie základné operácie (napr. otvorenie súboru), ale nemá bezpečnostný deskriptor ani podporu obnovy + pri hľadaní súboru vždy sekvenčný prechod alokačnými blokmi adresárov: O(n/2)
- NTFS: pre disky väčšie ako 500 MB, body regenerácie, B+ stromy – pri hľadaní súboru log n, minimalizácia prístupov na disk, až 2 TB súbor na zväzok (oproti 4 GB) + **stripe set** – paralelný prenos dát z veľkých súborov

Príklad systému súborov z pohľadu užívateľa: **UNIX** – stromová štruktúra adresárov, koreň stromu: /, pracovný adresár: práve používaný, relatívna a absolútna cesta, súbory . a .. , súbor je len postupnosť bajtov, I/O zariadenia sú súbory, presmerovanie stdin, stdout, príkazy shellu...

Otázky z termínu 4.6.2012:

1. Správa pozostávajúca zo znakov A, B, C, D s pravdepodobnosťami výskytu 1/8, 1/2, 1/4, 1/8 bola zakódovaná do binárnej abecedy unárnym kódom. Určte entropiu zdroja. Je takéto kódovanie optimálne?
2. Na zakódovanie správy ABRAKADABRA bolo použité Shannon-Fanovo kódovanie. Napíšte, ako vyzerala zakódovaná správa a aký je kompresný zisk oproti 7-bitovému blokovému kódu.
3. Lineárne Litwinovo hašovanie: hašujeme do kapies A, B, C, D, E. Ukážte postup riešenia pretečenia kapsy C, ak sa pri poslednom pretečení štiepila kapsa D.
4. Určte optimálnu aritu vyhľadávacieho stromu s výškou 4, ktorý obsahuje 64K kľúčov.
5. Definujte B strom.
6. Definujte pojmy: dotaz nad súborom, jednorozmerný dotaz, ortogonálny dotaz, dotaz na čiastočnú/úplnú intervalovú zhodu.