# I1 Množiny a relace

**(zobrazení, funkce, rozklady a ekvivalence)**

## Množiny

- **množina** je soubor prvků a je jimi plně určena (konečnou mnu můžeme určit výčtem prvků, nekonečnou musíme jinak, např. prvky patří do mny, pokud vyhovují stanovené podmínce)
- nejsou uspořádané
- prvky v nich se nemohou opakovat
- pojmy: podmnožina, vlastní podmnožina, sjednocení, průnik, rozdíl, doplněk, (kartézský) součin, potenční množina (=mna včech podmnožin, $|2^A| = 2^{|A|}$)

## Relace

- **relace** je podmna kartézského součinu
- může být n-ární
- **složení relací**
  - má smysl pouze u binárních
  - $R \subseteq A \times B, S \subseteq B \times C, S \circ R \subseteq A \times C$
- vlastnosti binárních relací
  - **reflexivní** = vždy (a, a)
  - **symetrická** = pokud (a, b), pak taky (b, a)
  - **antisymetrická** = pokud náleží (a, b) a (a != b), pak nenáleží (b, a)
  - **tranzitivní** = pokud (a, b) a (b, c), pak taky (a, c)
  - ekvivalence = ref, sym, tranz
  - (částečné) uspořádání = ref, antisym, tranz
  - kvaziuspořádání/předuspořádání = ref, tranz

## Funkce a zobrazení

- **funkce** = **zobrazení** (termín funkce se více používá v matematice, obzvlášť pro zobrazení do číselných mn)
- **zobrazení** je relace (omezená dalšími podmínkami)
- zobrazení přiřazuje prvkům z jedné množiny prvky z druhé množiny
- **totální fce** z A do B = pro každé *a* existuje právě jedno *b*

- **parciální fce** z A do B = pro každé *a* existuje nejvýše jedno *b*
- vlastnosti
  - **injektivní** (prostá) = pro každé *b* existuje nejvýše jedno *a*
    - takovou funkci lze invertovat
  - **surjektivní** (na) = pro každé *b* existuje alespoň jedno *a* (pokrývá celou cílovou mn)
  - **bijektivní** = injektivní a surjektivní
    - takovou funkci lze použít jako jednoznačný převod mezi dvěma mnami

## Rozklady a ekvivalence

- **rozklad** na M je množina podmnožin M taková, že
  - neobsahuje prázdnou mnu
  - množiny v ní obsažené nemají vzájemný průnik
  - sjednocení všech obsažených mn dá množinu M
- každá ekvivalence definuje rozklad a naopak
- značíme $R_N$ = M/R

# I2 Elementární kombinatorika

**(variace, kombinace a permutace)**

## Variace

- uspořádané k-tice z n prvků
- buď při výběru prvku odebíráme ze zdrojové množiny (bez opak.), nebo ne (s opak.)
- **bez opak** $V_k(n) = \frac{n!}{(n-k)!}$
- **s opak** $V'_k(n) = n^k$

## Permutace

- permutace (bez opak) na množině je skupina všech jejích prvků, uspořádaná v určitém pořadí
  - je to speciální případ variace bez opak., kdy vybereme všechny prvky
- permutace s opak. = máme podskupiny prvků, které jsou od sebe navzájem nerozlišitelné
- **bez opak** $P(n) = n!$
- **s opak** $P'(n_1, n_2, \ldots, n_k) = \frac{(n_1 + n_2 + \ldots + n_k)!}{n_1! \cdot n_2! \cdot \ldots \cdot n_k!}$
- **grupa permutací**
  - množina všech permutací na nějaké množině spolu s operací skládání tvoří nekomutativní grupu
  - zápis permutace tabulkou o dvou řádcích, nebo jako složení cyklů a/nebo transpozic (transpozice = cyklus délky 2)
  - permutace **sudá** = sudý počet transpozic, **lichá** analogicky
  - **skládání permutací** (jako skládání relací, operátorem „po“)
  - k = **řád permutace**, kdy $f^k$ = identická permutace
  - **samodružný** prvek se zobrazí na sebe
  - **inverzní** permutace přehozením řádků tabulky
  - grupoid (uzav), pologrupa (+ asoc), monoid (+ neutr), grupa (+ inv)

## Kombinace

- neuspořádané k-tice z n prvků

- **bez opak** $C_k(n) = \frac{n!}{k! \cdot (n-k)!} = \binom{n}{k}$
- **s opak** $C'_k(n) = \binom{n+k-1}{k}$
- **kombinační číslo** $\binom{n}{k} = \binom{n}{n-k}$
- **pascalův trojúhelník**
- **binomická věta** sloužící k výpočtu $(x+y)^n$

# I3 Uspořádání

**(relace uspořádání, uspořádané množiny a svazy, číselné obory)**

## Relace uspořádání

- **uspořádání** = reflexivní, antisymetrická, tranzitivní
- **předuspořádání** (kvazi-, polo-) = reflexivní, tranzitivní

## Uspořádané množiny

- **uspořádaná množina** je dvojice (mna, relace usp.)
- uspořádaná mna se nazývá **řetězec** (= **lineárně uspořádaná** mna), když jsou každé dva prvky srovnatelné (na hasseovském diagramu skutečně vypadá jako řetězec)
- typy uspořádání
  - **po bodech** = klasicky dva prvky mezi sebou
  - **po složkách** = na součinu množin, prvek je menší nebo roven, pokud jeho první i druhá složka jsou menší nebo rovny
  - **lexikografické** = na součinu množin, větší váhu má první množina, pokud jsou prvky stejné, pak se postupuje na další množinu
- pojmy
  - **nejmenší** prvek = je menší nebo roven než každý prvek množiny = jediný a zároveň minimální (analogicky **největší**)
  - **minimální** prvek = neexistuje žádný menší prvek (analogicky **maximální**)
  - **dolní závora** podmnožiny = je menší nebo rovna než každý prvek podmnožiny (analogicky **horní závora**)
  - **infimum** podmnožiny = je největší dolní závora podmnožiny
- **hasseovský diagram**

## Svazy

- **svaz** = uspořádaná mna, jejíž libovolná *dvouprvková* podmna má supremum a infimum
- svazové operace ^ (vybírá infimum ze dvou prvků), v (vybírá supremum ze dvou prvků), svaz značíme (A, ^, v)
- **podsvaz** = taková podmnožina svazu, která je také uzavřená na svazové operace ^, v
- **úplný svaz** = libovolná podmna má supremum a infimum
- postřehy

- pokud je mna konečná a neprázdná, pojmy svaz a úplný svaz splývají
- každý úplný svaz má nejmenší a největší prvek
- každá lineárně uspořádaná mna je svaz (např. (N, <=) - je svaz, ale není úplný, neexistuje sup N)

## Číselné obory

- **přirozená čísla**
  - definice pomocí množin 0 = {}, 1 = {{}}, 2 = {{}, {{}}}, 3 = {{}, {{}}, {{}, {{}}}}
- **celá čísla**
  - definujeme relaci (a, b) ~ (c, d) <=> a + d = c + b
  - 0 = [(0, 0)] = [(1, 1)] = …
  - 1 = [(1, 0)] = [(2, 1)] = …
  - -1 = [(0, 1)] = [(1, 2)] = …
- **racionální čísla**
  - definujeme relaci (a, b) tak, že značí zlomek a/b
  - (1, 1) = 1, (1, 2) = ½
- **reálná čísla**
  - složité, je nutné definovat uspořádání na Z a Q a pak využít řezů Q
- **komplexní čísla**
  - mna dvojic reálných čísel
  - číslo (0, 1) označíme ***i***, $i^2 = -1$
  - sčítání po složkách, násobení normálně roznásobit jako kdyby *i* byla proměnná
  - tvar z = a + bi (a = reálná část, b = imaginární část)

# I4 Pravděpodobnost a statistika

**(klasická a podmíněná pravděpodobnost, distribuční funkce a rozdělení náhodných veličin, výpočet střední hodnoty, rozptylu a kovariance)**

## Základy

- **pravděpodobnost** = číslo <0; 1>, které je mírou očekávanosti výskytu jevu
- **statistika** = zpracování číselných dat o nějakém souboru objektů
- **Ω** = **základní prostor**, množina všech možných výsledků, její prvky **ω** představují jednotlivé možné výsledky, **elementární jevy**
- **Δ** = **jevové pole**, systém podmnožin základního prostoru uzavřený na průnik, sjednocení a rozdíl
- jednotlivé množiny A ∈ Δ nazýváme **náhodné jevy**
- pojmy: jistý jev, nemožný jev, společné nastoupení jevů, nastoupení alespoň jednoho z jevů, neslučitelné jevy, jev A má za důsledek jev B, opačný jev
- **pravděpodobnostní fce** P(A) … pravděpodobnost jevu A
  - nezáporná, pro neslučitelné jevy se dá sčítat, pst jistého jevu je 1, pst opačného jevu je 1 – P(A)

## Klasická pravděpodobnost

- je **pravděpodobnostní prostor** (Ω, Δ, P) s pravděpodobnostní fcí P: Δ → R
- $P(A)=\frac{|A|}{|\Omega|}$
- pozn. **geometrická pst** $P(A)=\frac{vol\,A}{vol\,\Omega}$

## Podmíněná pravděpodobnost

- pst jevu za určité podmínky
- $P(A|H)=\frac{P(A\cap H)}{P(H)}$
- **Bayesův vzorec** $P(A|B)=\frac{P(B|A)\cdot P(A)}{P(B)}$
  - využíváme, pokud chceme P(A|B) a známe P(B|A)

## Distribuční funkce a rozdělení náhodných veličin

- **náhodná veličina** = proměnná, jejíž hodnota je určena výsledkem náhodného pokusu
  - jednoznačně určena rozdělením pravděpodobnosti, ale to je často nepřehledné, tak se charakterizuje veličinami jako střední hodnota, rozptyl, medián...
  - **spojitá** nebo **diskrétní**, pak má taky spojitou (resp. diskrétní) hustotu a distribuční fci
- **hustota pravděpodobnosti f(x)** = fce, která přiřazuje pravděpodobnost elementárním jevům
- **distribuční fce F(x)** = P(X <= x)
  - diskrétní $F(t)=\sum_{x_i \le t} f(x_i)$
  - spojitá $F(t)=\int_{-\infty}^{t} f(x)dx$
- **rozdělení náhodných veličin**
  - diskrétní
    - degenerované = jistý jeden výsledek
    - alternativní = pouze dva možné výsledky (např. hod korunou)
    - binomické = diskrétní Gaussova křivka
    - Poissonovo = aproximuje binomická rozložení
  - spojitá
    - rovnoměrné = všechny jevy stejná pst
    - exponenciální = exponenciálně klesá
    - normální (Gaussovo) = jako binomické pro velké n

## Výpočet střední hodnoty, rozptylu a kovariance (náhodné veličiny)

- **střední hodnota** $EX=\sum_i x_i \cdot f(x_i)$ $EX=\int_{-\infty}^{\infty} x \cdot f(x)dx$
- **rozptyl** $DX = E((X - EX)^2)$ … **směrodatná odchylka** = odmocnina z rozptylu
- **kovariance** = síla lineární závislosti dvou n.v.: $cov(X, Y) = E(XY) - E(X)E(Y)$
- **medián** = dělí n.v. na dvě stejně pravděpodobné poloviny
- **modus** = elementární jev s největší pravděpodobností

# I5 Výroková logika

**(syntax, sémantika, odvozovací systém výrokové logiky, důkazy ve výrokové logice, pravdivost a dokazatelnost logických formulí, rezoluce)**

## Obecně, syntax, sématika

- **logika** se zabývá správností usuzování; předpoklady → závěr
  - **neformální** (selský rozum)
  - **formální**
    - **dvouhodnotová** vs. **vícehodnotová**
    - **extenzionální** vs. **intenzionální**
    - **klasická logika** = dvouhodnotová extenzionální
      - **výroková** = logické spojky *a*, *nebo*, *pak*, ...
      - **predikátová** = přidává kvantifikátory *každý*, *alespoň jeden*
- **syntax**
  - **jednoduché** (neobsahují log. spojku) a **složené** (obsahují) výroky
  - disjunkce, konjunkce, implikace, ekvivalence, Shefferova (NAND), Peircova (NOR)
  - **abeceda**: výrokové symboly, symboly pro spojky, pomocné symboly (závorky)
  - **gramatika**
    - každý výrokový symbol je s.u.f. (správně utvořená formule)
    - negace s.u.f. je s.u.f.
    - (A) v (B), (A) <=> (b), apod. jsou s.u.f. pokud A i B jsou s.u.f.
- **sémantika**
  - **interpretace** (pravdivostní ohodnocení) přiřazuje hodnoty atomickým formulím
  - **valuace** rozšiřuje interpretaci na všechny formule dle pravidel výrokových spojek
  - **tautologie** je splňována každou interpretací, **kontradikce** žádnou
  - **splnitelná** formule alespoň jednou interpretací, tyto nazýváme **modely formule**
  - **množina formulí**, **model množiny** = interpretace, kterou všechny formule splňují
  - pokud formule A logicky vyplývá z množiny T, píšeme **T ⊢ A**
  - ohodnocení výroků **tabulkovou metodou**

## Odvozovací systém výrokové logiky

- **formální systém výrokové logiky** musí splnit tyto požadavky:
  - **korektnost (bezespornost)** (vždy) = nelze odvodit tvrzení a zároveň jeho negaci
  - **úplný** = přidáme-li neodvoditelnou s.u.f. k systému, stane se sporným
  - **rozhodnutelný** = existuje algoritmus pro ověření dokazatelnosti každé s.u.f.
  - **nezávislost axiomů** = odvoditelný lze vypustit
- pravidlo **modus ponens** (**MP**) $\frac{A.\ A \Rightarrow B}{B}$

## Důkazy ve výrokové logice

- **důkaz** = konečná posloupnost s.u.f (kroků důkazu), každá je buď axiomem nebo byla odvozena pomocí některého pravidla odvození z předcházejících s.u.f.
- formule je **dokazatelná** (= **teorém** v systému S), jestliže v daném S existuje její důkaz

## Pravdivost a dokazatelnost logických formulí

- **pravdivost** je dána sémantikou
- **dokazatelnost** je dána syntaxí (je pravda, pokud k ní existuje důkaz)

## Rezoluce

- **rezoluce** = systém pro strojové dokazování, založený na **vyvracení**
- **klauzule** = mna literálů, v DNF … (a ∨ b ∨ c)
- **formule** = mna klauzulí, v KNF … (a ∨ b ∨ c) ∧ (¬a)
- **rezoluční pravidlo** = $C_1 = \{p\} \cup C'_1 ; C_2 = \{\neg p\} \cup C'_2 ; rezolventa\ C = C'_1 \cup C'_2$
- **rezoluční důkaz** = posloupnost rezolučních kroků; sjednotíme všechny předpoklady, formuli T ∧ ¬A převedeme do KNF; pokud odvodíme {}, tak platí A (protože neplatí ¬A)
  - **začneme s cílem**, rezolvujeme
  - **lineární** = k výsledku vždy zprava připojujeme předtím odvozenou klauzuli (úplná)
  - **lineární vstupní (LI)** = zprava nepřipojujeme mezivýsledky (úplná pro Hornovy klauzule)
  - **LD** = uspořádané klauzule, při rezoluci vkládáme na místo rezolventy
  - **SLD** = LI se selekčním pravidlem – vybírá literál pro rezoluci (v Prologu nejlevější)
- **Hornovy klauzule** = **fakta** (jeden pozitivní literál), **pravidla** (jeden pozitivní, a 1 až * negativních), **cíl** (1 až * negativních)

# I6 Predikátová logika prvního řádu

**(syntax, sémantika, prenexace, skolemizace, unifikace, rezoluce)**

## Obecně, syntax, sémantika

- syntax
  - **predikát** (značíme P, Q, …) = n-ární relace, vyjadřuje vlastnosti a vztahy objektů
  - **konstanta** (značíme a, b, …) = prvek předem specifikované mny = domény
  - **proměnná** (značíme x, y, …) = „zástupce“, může nabývat libovolných hodnot z domény
    - x je **vázaná** pokud existuje podformule obsahující x a začíná ∀x nebo ∃x, jinak je **volná**
  - **funkce** (značíme f(...), g(...), …) reprezentují složená jména objektů, f: $D^n \rightarrow D$
  - **kvantifikátory** – **univerzální** (**obecný**) **∀x** P(x), **existenční ∃x** P(x)
  - **term** = výraz složený z funkcí, konstant a proměnných
  - **atomická formule** = P(term, term, …) **|** $term_1 = term_2$
  - **formule** = at. formule A **|** ¬A **|** (A v B), (A => B), … **|** ∀x A, ∃x A
  - **uzavřená** formule neobsahuje volné výskyty proměnných
  - **otevřená** formule neobsahuje kvantifikátory
  - **substituce** – mna např. {x/f(y), y/6, z/a}; jiný př. A = ∃x P(x, y); A(y/2) = ∃x P(x, 2)
    - **pouze volnou proměnnou** za: proměnnou (ale aby nedošlo k vazbě), konstantu, funkci
    - v rámci jedné substituce paralelně, při **kompozici substitucí** postupně (tranzitivně)
- **sémantika**
  - **interpretace** = struktura I obsahující: doménu, definice všech funkcí, definice relací všech predikátů
  - **valuace** = přiřazení konkrétních hodnot proměnným
  - formule je **pravdivá** v *interpretaci* I, je-li splněna pro každou *valuaci*
    - **tautologie** = pravdivá v každé interpretaci; analogicky def. **splnitelná**, **kontradikce**

## Prenexace

- **prenexová normální forma** převádí libovolnou (uzavřenou) formuli do tvaru, kde jsou všechny kvantifikátory na začátku a následuje otevřené (= bez kvantifikátorů) jádro v KNF nebo DNF
  - eliminovat zbytečné kvantifikátory
  - přejmenovat proměnné, aby u každého kvantifikátoru byla jiná

- eliminovat všechny spojky různé od ∧, ∨
- přesunout negaci dovnitř, kvantifikátory doleva
- převést jádro do KNF nebo DNF

## Skolemizace

- **Skolemova normální forma** je prenexová NF, ve které jsou pouze univerzální kvantifikátory
  - převést do KNF PNF
  - odstranit existenční kvantifikátory – nahradit Skolemovými funkcemi
    $\forall x_1 \forall x_2 ... \exists y\, P(x_1, x_2, y) \quad \rightarrow \quad \forall x_1 \forall x_2 P(x_1, x_2, f(x_1, x_2, ...))$

## Unifikace

- subst. Φ je **unifikátorem** mny S = {$E_1$, $E_2$, …}, pokud $E_1\Phi = E_2\Phi = \ldots$, tedy S Φ má 1 prvek
- z **nejobecnějšího unifikátoru (mgu)** mny S můžeme substitucí vytvořit každý jiný unifikátor

## Rezoluce

- pro strojové zpracování, založená na vyvracení, formule ve Skolemově NF, nepíšeme kvant.
- **literály** představují atomické formule a jejich negace
- **klauzule** = disjunkce literálů; **formule** = konjunkce klauzulí
- rezoluční pravidlo
  - klauzule $C_1$, $C_2$ bez společných proměnných (přejmenovat)
  - $C_1 = C'_1 \cup P(x_1), ..., P(x_n); C_2 = C'_2 \cup P(y_1), ..., P(y_m)$
  - je-li Φ mgu $\{P(x_1), ..., P(x_n), P(y_1), ..., P(y_m)\}$ , rezolventou $C_1$ a $C_2$ je $C'_1\Phi \cup C'_2\Phi$
  - př. rezolvujeme $\{P(x, y), \neg R(x)\}, \{\neg P(a, b)\}$
    - mgu $(\{P(x, y), P(a, b)\}) = \{x/a, y/b\}$
    - aplikací mgu na zbytek první klauzule dostáváme rezolventu $\{\neg R(a)\}$
- stejně jako ve výrokové logice je **korektní** a **úplná**
- lze využít všechny druhy zjemnění: **vstupní**, **lineární vstupní**, **LD**, **SLD**

# I7 Prolog

**(SLD-rezoluce, SLD-stromy, výpočetní mechanizmus Prologu, základy programování v Prologu)**

- **Prolog** je deklarativní – neříkáme jak se dobrat k výsledku, pouze řekneme co ověřit, vložíme pravidla a přesný postup výpočtu necháme na Prologu
- **Hornovy klauzule** – vždy max. 1 pozitivní literál
  - **fakta** (jeden pozitivní literál)
  - **pravidla** (jeden pozitivní, a 1 až * negativních)
  - **cíl** (1 až * negativních)
- rezoluce **lineární**, **lineární vstupní**, **LD**, **SLD**

## SLD-rezoluce

- LI se selekčním pravidlem (vybere literál, na kterém chceme rezolvovat, v Prologu nejlevější)
- úplná pro Hornovy klauzule
- **algoritmus**
  - vybereme z cílové klauzule literál, na kterém chceme rezolvovat
  - najdeme vstupní klauzuli, která obsahuje pozitivní literál takový, aby se dal unifikovat s ⬆
  - unifikujeme a rezolvujeme, na výsledku rezoluce opakujeme první krok
  - pokud rezolucí získáme prázdnou klauzuli, víme, že **platí negace cílové klauzule**

## SLD-stromy

- **SLD-strom** reprezentuje mechanizmus výpočtu pomocí SLD-rezoluce
  - v kořeni je **cílová klauzule**
  - z každého uzlu vedou **hrany**, jedna pro každou klauzuli, se kterou je možné (na prvním literálu) rezolvovat; hrany se obvykle popisují číslem klauzule (a mnou mgu)

## Výpočetní mechanizmus Prologu

- Prolog je založen na predikátové logice prvního řádu, zaměřuje se na SLD-rezoluci
- Prolog se po **nejlevější větvi** zanořuje do hloubky, pokud skončí neúspěchem, vrací se nahoru a zkouší další větve
- zadáním **;** po úspěšném vyhodnocení vynutíme backtracking a hledání alternativní cesty
- Prologu záleží na pořadí klauzulí, není úplný, může se **zacyklit** (raději dát **fakta na začátek**)

## Základy programování v Prologu

- základ prologu je **databáze klauzulí** (fakta a pravidla), nad kterými je možno klást **dotazy formou tvrzení**, Prolog vyhodnotí jejich pravdivost
- **syntax**
  - **termy – datové objekty**
    - **konstanty** = **celá čísla**, **desetinná čísla**, **atomy** (textové řetězce apod.)
    - **proměnné** (N, VYSLEDEK, ...)
    - **složené termy** = **funktor** (jméno, arita), **argumenty** (bod(X, Y, Z), …)
  - **program**
    - mna programových klauzulí
    - proměnné v lokální klauzuli
    - **pravidla**: **q:- r, s, t.**
      - **syn(A,B):- rodic(B, A), muz(A).**
    - **fakta**: **r.**
      - **divka(monika).**
    - **cíle**: **?- q.**
- **poznámky**
  - seznamy definovány induktivně, [] prázdný seznam
  - základními přístupy jsou unifikace a rekurze
  - anonymní proměnná _ když není podstatná: je_dite(X):- dite(X, _).

# I8 Důkazy programů

**(dokazování vlastností progamů, induktivní metody, invarianty cyklů)**

## Dokazování vlastností programů

- **pojmy**
  - **In** = mna vstupních dat
  - **Out** = mna výstupních dat
  - **A: In** → **Out** = algoritmus
  - **φ: In** → **Bool** = vstupní podmínka (z In vybírá hodnoty povolené jako vstup algoritmu)
  - **ψ: In** × **Out** → **Bool** = výstupní podmínka (zda daný y ∈ Out je výstup pro daný x ∈ In)
- dokazujeme, že algoritmus je
  - **konvergentní** vzhledem k φ = zastaví pro každé x, pro které je φ(x) = true
  - **parciálně korektní** vzhledem k φ, ψ = pokud pro každé x, pro které je φ(x) = true, platí ψ(x, A(x)) = true (pokud algoritmus nezastaví, není to na překážku parciální korektnosti)
  - **totálně korektní** vzhledem k φ, ψ = konvergentní vzhledem k φ a parciálně korektní vzhledem k φ, ψ
- dokazování bývá jednodušší pro funkcionální paradigma, protože funkce nemají vedlejší účinky na stav programu
- při větvení programu se často větví i důkaz (dokazuje se zvlášť pro každou větev)
- většinou dokazujeme **totální korektnost** = dokážeme konvergentnost a parciální korektnost

## Induktivní metody

- **indukce**
  - **indukční báze (IB)** – dokážeme platnost dokazovaného tvrzení pro nějaký argument
  - **indukční krok (IK)** – předpokládáme platnost tvrzení pro nějaký vstup n, dokazujeme pro jiný (nejčastěji n+1, n-1, …). IK s IB takto „pokryjí důkazem“ všechny požadované vstupy.
- **fixace parametru**
  - pokud má program více parametrů a jeden z nich se v průběhu počítání nemění, můžeme jej zafixovat = prohlásit za libovolný, ale neměnný. Důkaz vedeme indukcí, musí platit pro jakoukoliv hodnotu fixního parametru.
- **indukce k součtu parametrů**
  - použijeme např. když máme více parametrů, které se postupně zmenšují, ale pokaždé jiný z nich (a chceme třeba dokázat konvergentnost)

- **zesílení dokazovaného tvrzení**
  - dokazujeme silnější tvrzení než požadované (nejlépe když požadované tvrzení triviálně vyplývá z toho silnějšího)
  - při důkazu indukcí někdy chceme mít dostatečně silný předpoklad (tedy musíme zesílit i bázi a krok), aby tvrzení šlo jednodušeji dokázat

## Invarianty cyklů

- **mezilehlá podmínka** = tvrzení, jehož platnost v určitém místě programu dokážeme a pak nám pomůže s dokázáním jiné mezilehlé podmínky, invariantu cyklu nebo výstupní podmínky
- invariant cyklu = mezilehlá podmínka umístěná v cyklu
  - z předpokladů platících před cyklem dokážeme, že platí v prvním běhu
  - dokážeme, že je-li splněn invariant a cyklus se provede, potom invariant opět platí
  - dokážeme, že je-li splněn invariant a cyklus skončí, potom je splněna mezilehlá podmínka za cyklem nebo výstupní podmínka algoritmu

# I9 Rekurze

**(rekurzivní definice funkcí, funkce vyššího řádu, částečná aplikace, curryfikace, definice funkcí rekurzivně a pomocí kombinátorů, definice vyšších funkcí bez použití formálních parametrů)**

## Rekurzivní definice funkce

- rekurze nastává, když funkce volá sama sebe (s jinými parametry, jinak by cyklila)
  - **přímá** = fce volá přímo sama sebe (př. faktoriál)
  - **nepřímá** = dvě nebo více fcí se střídavě volají navzájem (př. even – odd)

## Funkce vyšších řádů, částečná aplikace, curryfikace

- **funkce vyšších řádů** jsou funkce, jejichž argument nebo výsledek může být zase funkce
- některé jazyky pohlížejí na funkce vyšších řádů jako na unární fce, které vracejí opět funkci
- vyšší funkce provádějí **částečnou aplikaci** (aplikují se pouze na první argument) a **vracejí funkci** s aritou o 1 nižší
- **curryfikace** je převod nižších funkcí (které mají za argument dvojice) na vyšší
  - fce **curry** převádí na funkci vyššího řádu (= na částečnou aplikaci)
    curry :: ((a, b) → c) → a → b → c
  - fce **uncurry** převádí na funkci nižšího řádu (= fce bere dvojici)
    uncurry :: (a → b → c) → (a, b) → c
  - mějme fce: add :: Int → Int → Int add' :: (Int, Int) → Int
    pak platí: curry add' = add uncurry add = add'

## Definice funkce rekurzivně a pomocí kombinátorů

- krom přímé rekurze lze využít **kombinátorů**, které mohou samy v sobě rekurzi obsahovat
- příkladem kombinátorů s rekurzí mohou být funkce **foldl**, **foldr**
  - **foldr :: (b → a → a) → a → [b] → a** bere následující parametry:
    - funkci *(b → a → a)*, která se vždy aplikuje na mezivýsledek a další prvek seznamu
    - počáteční hodnotu *a*, která bude tvořit mezivýsledek před aplikací fce na první prvek
    - seznam *[b]*, na který se má aplikovat
    - vrací hodnotu mezivýsledku po aplikaci fce na všechny prvky seznamu
    - rekurzi obsahuje v tom smyslu, že po spočtení mezivýsledku volá sama sebe na seznam zkrácení o jeden prvek

## Definice vyšších funkcí bez použití formálních parametrů

- vyšší funkce lze často **definovat pomocí jiných funkcí** tak, že lze vynechat formální parametry – jednodušší a přehlednější zápis
- např. definice funkce zip pomocí zipWith
  - zip s t = zipWith (,) s t  je ekvivalentní s definicí  zip = zipWith (,)
- tomuto principu se říká **η-redukce** (eta redukce)
  - vychází z **principu extenzionality**: jestliže platí f x = g x pro všechna x ze sjednocení def. oborů fcí f a g, pak f = g

# I10 Vyhodnocování výrazů

**(pořadí vyhodnocování, striktní a normální redukce, líná redukce, efektivita, nekonečné datové struktury, definice funkcí nad nekonečnými datovými strukturami)**

## Pořadí vyhodnocování

- programovací jazyky mají stanovené **pořadí vyhodnocování jazykových konstrukcí**
- typicky může vypadat takto: **skládání funkcí** > indexování v seznamu > **matematika** (umocnění > multiplikativní op. > aditivní op.) > přidání prvku a spojování seznamů > **relační operátory** > logická **konjunkce** > logická **disjunkce**

## Striktní, normální a líná redukce

- **redukční krok** = úprava výrazu, kdy se některý z podvýrazů nahradí pravou stranou definice funkce, nebo vyčíslíme jednoduchou operaci aplikovanou na jednoduché argumenty
- **redukční strategie** = pravidlo, které pro každý výraz jednoznačně určuje následující redukční krok (tj. určuje, který podvýraz se bude redukovat jako první)
- **striktní** redukční strategie = **nejprve vyhodnocujeme argumenty**
  - postupujeme zevnitř
  - argumenty **vyhodnocujeme právě jednou**
  - v imperativních jazycích odpovídá **volání hodnotou**
- **normální** redukční strategie = **nejprve redukujeme (nahrazujeme pravou stranou def.)**
  - postupujeme z vnějšku
  - argumenty **vyhodnocujeme i víckrát** – tolikrát, kolikrát na to při výpočtu dojde
  - je **efektivně normalizující** – nezacyklí se, pokud nemusí
  - v imperativních jazycích odpovídá **volání jménem**
- **líná** redukční strategie = **nevyhodnocujeme to, co nemusíme**
  - **modifikace normální** redukční strategie, je taky **efektivně normalizující**
  - využívá toho, že funkcionální jazyky jsou **referenčně transparentní** (výraz se vyhodnotí vždy stejně, lze se vyhnout jeho opakovanému vyhodnocování)
  - argumenty **vyhodnocujeme nejvýše jednou**
  - je možné pracovat s **nekonečnými datovými strukturami**
  - v jazycích s vedlejšími efekty by líné vyhodn. způsobilo nepřehledné chování programu
    - ale operátory && a || se ve většině imperativních jazyků chovají líně

## Efektivita

- **normální** redukce je efektivně normalizující = necyklí, když nemusí
- **striktní** redukce je efektivnější než normální (vyhodnocuje argumenty právě jednou), ale může se zacyklit
- **líná** redukce je efektivnější než normální (vyhodnocuje argumenty nejvýše jednou), ale vyžaduje referenční transparentnost jazyka

## Nekonečné datové struktury a definice funkcí nad nimi

- líná redukce umožňuje pracovat s funkcemi, které jsou definovány na **nedefinovaných argumentech** (např. fce const x y = x, volání const 1 (1/0))
- nedefinovaným podvýrazem jsou také **nekonečné datové struktury** (fce ones = 1 : ones)
- často se jedná o nekonečné seznamy – seznamový konstruktor : nevyhodnocuje argumenty, dokud nemusí, při práci s nekonečným seznamem vyhodnotí jen tolik, kolik je potřeba
- imperativní jazyky jako Python a Ruby umožňují definici nekonečných d. s. pomocí klíčového slova **yield**, které slouží jako „vícenásobný return“
- intuitivní omezení při práci s nekonečnými datovými strukturami
  - prvky čteme z jedné strany a postupně
  - nesmíme provést operaci, která si vyžádá všechny prvky seznamu

# I11 Regulární jazyky

**(regulární jazyky, způsoby jejich reprezentace, vlastnosti regulárních jazyků, vztah mezi konečnými automaty a regulárními jazyky)**

## Regulární jazyky a způsoby jejich reprezentace

- **abeceda** = libovolná konečná mna znaků (písmen, symbolů)
- **slovo** nad abecedou Σ = libovolná konečná posloupnost znaků této abecedy
- **jazyk** nad abecedou Σ = libovolná množina slov nad Σ
- **regulární jazyk** má tyto **způsoby reprezentace**:
  - je generován **regulární gramatikou**
  - je akceptován **konečným automatem** (deterministickým nebo nedeterministickým)
  - je popsatelný **regulárním výrazem**
- **obecně** – **gramatika** G je **čtveřice (N, Σ, P, S)**, kde:
  - N je neprázdná konečná mna neterminálů
  - Σ je mna terminálů (N ∪ Σ = V, mna všech symbolů gramatiky)
  - P ⊆ V*NV* x V* je množina pravidel
  - S ∈ N je počáteční neterminál
- **regulární gramatika** má každé pravidlo ve tvaru **A** → **aB** nebo **A** → **a** s výjimkou **S** → **ε**, pokud se **S nevyskytuje na pravé straně** žádného pravidla
- **konečný automat** M je **pětice (Q, Σ, δ, $q_0$, F)**, kde
  - Q je neprázdná konečná mna **stavů**
  - Σ konečná mna vstupních symbolů = **vstupní abeceda**
  - δ: Q x Σ → Q je parciální **přechodová funkce** (δ: Q x Σ → $2^Q$ totální v případě nedeterm.)
  - $q_0$ ∈ Q je **počáteční stav**
  - F ⊆ F je mna **koncových stavů**
  - pozn.: rozšířená přechodová funkce – pro celá slova
- **regulární výraz** nad abecedou Σ
  - **ε**, Ø a **a** pro každé a ∈ Σ jsou reg. výrazy nad Σ
  - jsou-li E, F reg. výrazy nad Σ, pak taky **(E.F)**, **(E+F)**, a **(E*)** jsou reg. výrazy nad Σ
  - každý reg. výraz vznikne po konečném počtu aplikací prvních dvou kroků
  - pozn.: Kleeneho věta: jazyk je rozpoznatelný kon. aut. <=> je popsatelný reg. výrazem

- **Chomského hierarchie**: 0 frázové, 1 kontextové, 2 bezkontextové, 3 regulární
- **pumping lemma** slouží k důkazu **neregularity** jazyka
  - pro libovolné $n \in \mathbb{N}$ (vybírá nepřítel)
  - existuje takové $w \in L$, délky alespoň n, pro které platí (vybírám já)
  - při libovolném rozdělení slova w na tři části x, y, z takovém, že $|xy| < n$ a $y \neq \varepsilon$ (nepřítel)
  - existuje alespoň jedno $i \in N0$ takové, že $xy^iz \notin L$ (já)
- **Myhill-Nerodova věta** je nutnou a **postačující podmínkou** pro regularitu jazyka
  - M-N věta říká, že tato tvrzení jsou ekvivalentní:
    - L je rozpoznatelný konečným automatem
    - L je sjednocením některých tříd rozkladu určeného pravou kong. na $\Sigma^*$ s koneč. indexem
    - relace $\sim_L$ má konečný index
  - **pravá kongruence** ~ je ekvivalence, pro níž platí $u \sim v \Rightarrow uw \sim vw$
    - index ~ je počet tříd rozkladu $\Sigma^*/\sim$ (může být nekonečně mnoho)
  - **prefixová ekvivalence** $\sim_L$ je ekvivalence $u \sim_L v \Leftrightarrow (uw \in L \Leftrightarrow vw \in L)$

## Vlastnosti regulárních jazyků

- třída reg. jazyků je **uzavřena na**: sjednocení, průnik, rozdíl, **doplňek**, zřetězení, iteraci, pozitivní iteraci, **reverzi**
- **rozhodnutelné problémy** pro třídu reg. jazyků: ekvivalence, inkluze, příslušnost, prázdnost, univerzalita, konečnost

## Vztah mezi konečnými automaty a regulárními jazyky

- třída jazyků generována **reg. gramatikami =** třída jazyků rozpoznávána **konečnými aut.**
- regulární gramatika → konečný automat
  - neterminály → stavy
  - pro každé pravidlo A → aB přidáme B do (A, a)
  - pro pravidla C → a zavedeme spec. stav $q_f$, který přidáme do (C, a)
  - poč. stav S, konc. stav $q_f$ (+ S, pokud G obsahuje S → ε)
- konečný automat → regulární gramatika
  - stavy → neterminály
  - přechodová fce → pravidla
  - taková G může mít S → ε a S na pravé straně, ale lze zkonstruovat ekvivalentní reg. gr.

# I12 Konečné automaty

**definice, konstrukce konečného automatu, minimalizace konečného automatu, převod nedeterministického konečného automatu na deterministický automat)**

## Definice, konstrukce

- **konečný automat** M je **pětice (Q, Σ, δ, $q_0$, F)**, kde
  - Q je neprázdná konečná mna **stavů**
  - Σ konečná mna vstupních symbolů = **vstupní abeceda**
  - δ: Q x Σ → Q je parciální **přechodová funkce** (δ: Q x Σ → $2^Q$ totální v případě nedeterm.)
  - $q_0$ ∈ Q je **počáteční stav**
  - F ⊆ F je mna **koncových stavů**
- **deterministický** (max. jeden přechod pro dvojici ($q_?$, a))**, nedeterministický** (více přechodů)
- **rozšířená přechodová funkce** δ: Q x Σ* → Q definována induktivně pro celá slova
- **jazyk akceptovaný automatem** L(M) = slova, pod kterými přejde z počátečního do jednoho z koncových stavů
- **konstrukce a vyjádření**:
  - **uspořádanou pěticí**
  - **tabulkou** (řádky – stavy; sloupce – znaky; v buňkách stavy, kam přejde)
  - **přechodovým grafem** (uzly – stavy, hrany – znaky)
  - výpočetním stromem (nejednoznačný a škaredý, raději nezmiňovat)
- **synchronní paralelní kompozice** = průnik, sjednocení, rozdíl
  - nutné automaty s totální přechodovou fcí
  - sloupce – stavy jednoho automatu, řádky – stavy druhého automatu, přechody simultánně
  - koncové stavy jsou pak průnik, sjednocení, rozdíl koncových stavů obou automatů
- **automat pro komplement**
  - nutný automat s totální přechodovou fcí
  - prohodíme koncové stavy s nekoncovými

## Minimalizace

- minimální konečný automat = s nejmenším počtem stavů, který rozpoznává daný jazyk L
- z M-N věty: počet stavů minimálního automatu pro L je roven indexu $\sim_L$

    - minimální FA pro jazyk L je určen jednoznačně až na isomorfismus (přejmenování stavů)
- odstranit **nedosažitelné stavy** = stavy, do kterých se výpočet nikdy nedostane
- ztotožnit navzájem **ekvivalentní stavy** = pokud se z nich dostaneme do akceptujících stavů pod shodnými množinami slov
- **algoritmus**
    - převedeme na **totální** (přidáním stavu N, „žumpy“)
    - řádky – stavy rozdělené do skupin, sloupce – znaky abecedy, buňky – čísla skupin
    - stavy rozdělíme do **skupin** označených římskými číslicemi (v prvním kroku 2 skupiny – **koncové a ostatní**); do tabulky zapisujeme, **do jaké skupiny se přechází** z daného stavu
    - v dalších krocích vytváříme nové skupiny tak dlouho, dokud všechny **stavy ve skupině nemají shodné přechody** (shodně vyplněný řádek)
    - skupiny prohlásíme za stavy minimálního automatu (má totální přechodovou fci)

## Převod nedeterministického na deterministický

- **algoritmus**
    - řádky – mny stavů, sloupce – znaky abecedy, buňky – mny stavů
    - začínáme počátečním stavem – do tabulky napíšeme, do jakých stavů se z něj dostaneme pod daným znakem
    - pokud ještě není v hlavičce tabulky některý ze stavů nebo některá z množin stavů, které se vyskytují v buňkách tabulky, **zapíšeme je do hlavičky nového řádku**
    - **rekurzivně vyplňujeme** tabulku
    - pro množiny stavů se do tabulky vyplňuje **sjednocení** všech stavů, do kterých se lze dostat ze stavů v dané množině
    - po vyplnění celé tabulky přejmenujeme stavy (**z množin uděláme jeden nový stav**), tj. co řádek, to stav
    - koncové stavy jsou všechny mny, které obsahují **alespoň jeden koncový stav** původního automatu
    - výsledný automat má totální přechodovou fci

# I13 Bezkontextové jazyky

**(definice, vlastnosti, způsoby jejich reprezentace, konstrukce bezkontextové gramatiky a zásobníkových automatů, normální formy bezkontextových jazyků, použití lematu o vkládání pro bezkontextové jazyky, uzávěrové vlastnosti bezkontextových jazyků)**

## Definice, vlastnosti, způsoby reprezentace, konstrukce CFG a PDA

- **jazyk** je bezkontextový, pokud existuje bezkontextová gramatika, která jej generuje
- CFL reprezentujeme **bezkontextovou gramatikou** nebo **zásobníkovým automatem**
- **obecně** – **gramatika** G je **čtveřice (N, Σ, P, S)**, kde:
  - N je neprázdná konečná mna neterminálů
  - Σ je mna terminálů (N ∪ Σ = V, mna všech symbolů gramatiky)
  - P ⊆ V*NV* x V* je množina pravidel
  - S ∈ N je počáteční neterminál
- **bezkontextová gramatika** má P ⊆ N x V* (1 neterminál → posloupnost neterm. a term.)
- **redukovaná** = **bez nepoužitelných symbolů**
  - nepoužitelnost typu 1 – neterminál nelze přepsat na sekvenci terminálů
  - nepoužitelnost typu 2 – neterminál je nedosažitelný
- **bez epsilon pravidel** = neobsahuje A → ε, nebo obsahuje pouze S → ε (a S není napravo)
- **necyklická** = A nelze přepsat zpět na A (neplatí $A \Rightarrow^{+} A$)
- **vlastní** = bez nepoužitelných symbolů, bez ε-pravidel a necyklická
- **rekurzivní** = platí $A \Rightarrow^{+} \alpha A \beta$ (**levorekurzivní** pokud α = ε, analogicky **pravorekurzivní**)

## Normální formy CFG

- **Chomského** NF
  - **bez ε-pravidel**
  - každé pravidlo je v jednom z tvarů
    - **A → BC**
    - **A → a**
  - algoritmus transformace
    - pro každý terminál vytvoříme neterminál, který se na něj přepíše
    - pravidla X → ABC apod. předěláme na X → A<BC> apod., kde <BC> je nový neterminál

- **Greibachové** NF
  - **bez ε-pravidel**
  - každé pravidlo je tvaru **A** → **aα**, kde a je terminál a α je libovolná sekvence term. a neterm.
  - algoritmus je složitější, vyžaduje odstranění levé rekurze

## Lemma o vkládání pro CFL

- k důkazu toho, že daný jazyk **není bezkontextový**, když
  - pro libovolnoé n (vybírá nepřítel)
  - existuje slovo z, |z| > n takové, že (vybírám já)
  - pro všechna slova u, v, w, x, y splňující: z = uvwxy, vx ≠ ε, |vwx| <= n (vybírá nepřítel)
  - existuje takové i, že $uv^iwx^iy \notin L$ (vybírám já)

## Uzávěrové vlastnosti CFL

- třída CFL je **uzavřená** na:
  - sjednocení
  - zřetězení
  - iterace
  - pozitivní iterace
  - **průnik s regulárním jazykem**
- třída CFL **není uzavřená** na:
  - **průnik**
  - **doplněk**
- **rozhodnutelné problémy**
  - příslušnosti
  - prázdnosti
  - konečnosti

# I14 Zásobníkové automaty, syntaktická analýza

**(definice, převod bezkontextové gramatiky na zásobníkový automat; syntaktická analýza shora dolů a zdola nahoru, průběh analýzy daného slova)**

## Definice, převod CFG na PDA

- zásobníkový automat má zásobník, může ukládat na vrchol a číst z vrcholu
- **zásobníkový automat** je sedmice $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
  - $Q$ je neprázdná konečná mna **stavů**
  - $\Sigma$ konečná mna vstupních symbolů = **vstupní abeceda**
  - $\Gamma$ konečná mna symbolů zásobníku = **zásobníková abeceda**
  - $\delta$: $(Q \times (\Sigma \cup \varepsilon) \times \Gamma) \rightarrow P_{fin}(Q \times \Gamma^*)$ je (parc.) **přechodová funkce** $P_{fin}$ = mna koneč. podmn.
  - $q_0 \in Q$ je **počáteční stav**
  - $Z_0 \in \Gamma$ je **počáteční symbol v zásobníku**
  - $F \subseteq F$ je mna **koncových stavů**
- vrchol zásobníku běžného PDA se píše vlevo
- konfigurace $(p, w, a) \in Q \times \Sigma^* \times \Gamma^*$ (tj. stav, slovo k přečtení, stav zásobníku)
- PDA akceptuje dvěma způsoby (mají ekvivalentní vyjadřovací sílu)
  - L(M) koncovým stavem = mna slov, pod kterými přejde do některého z konc. stavů
    - přidám nové dno zásobníku, jakmile jsem schopen ho číst, přejdu do konc. stavu
  - $L_\varepsilon(M)$ prázdným zásobníkem = mna slov, pod kterými automat vyprázdní zásobník
    - nový symbol na dno, ze všech konc. stavů přechod do $q_e$, tam vyprázdním zásobník
- **rozšířený PDA** může číst více symbolů ze zásobníku naráz
  - ekvivalentní vyjadřovací síla jako obyčejný PDA, ale pohodlněji vyjádřený
- problém sestrojení **PDA pro danou CFG** řeší **syntaktická analýza**

## Syntaktická analýza

- **shora dolů nedeterministická**
  - vytvoříme PDA simulující **levé derivace** v G (**rozvíjíme nejlevější neterminál**)
    - má jediný stav q, akceptuje prázdným zásobníkem
    - $\delta(q, \varepsilon, A)$ obsahuje $(q, \alpha)$ právě když $A \rightarrow \alpha \in P$ … přepis nejlevějšího neterminálu na jeho pravou stranu

    - $\delta(q, a, a) = \{(q, \varepsilon)\}$ pro všechna $a \in \Sigma$ … odmazávání terminálů
  - **výpočet**
    - pokud automat najde **na vstupu i na zásobníku** stejný **terminál**, přečte jej na vstupu a **umaže jej** ze zásobníku
    - pokud automat najde na zásobníku **neterminál**, nic nečte ze vstupu, ale neterminál **rozvine v zásobníku** (nedeterministicky vybere jedno pravidlo)
    - výpočet končí úspěchem, pokud se podaří **přečíst celý vstup a zásobník je prázdný**
- **zdola nahoru nedeterministická**
  - **vrchol zásobníku** píšeme v tomto případě **vpravo**
  - vytvoříme PDA simulující pravou **derivaci v obráceném pořadí**
    - má 2 stavy: q – tam se provádí výpočet; r – tento stav akceptuje
    - $\delta(q, a, \varepsilon) = \{(q, a)\}$ pro všechna $a \in \Sigma$ … načítání vstupu na zásobník
    - $\delta(q, \varepsilon, \alpha)$ obsahuje $(q, A)$ právě když $A \to \alpha \in P$ … redukce pravé strany na neterminál
    - $\delta(q, \varepsilon, \perp S) = \{(r, \varepsilon)\}$ … akceptuje (r je koncový stav)
  - **výpočet**
    - může kdykoli **číst do zásobníku vstupní symbol**
    - je-li na zásobníku **řetězec tvořící pravou stranu** nějakého pravidla v G, může ho **nahradit odpovídajícím** levostranným **neterminálem** (a ze vstupu nic nečte)
    - **akceptuje**, pokud má na zásobníku pouze **počáteční neterminál** G (a pod ním dno)
- **Cocke-Younger-Kasami zdola nahoru deterministická** (složitost $O(\mathbf{n^3})$)
  - vstupem je CFG v **Chomského NF**
  - řádky – „slovo délky“, sloupce – jednotlivá písmena slova, buňky – mna neterminálů, ze kterých odvodíme slovo o délce dané řádkem a začínající na pozici dané sloupcem
  - do 0. řádku napíšeme **slovo** – do každého sloupce jeden terminál
  - do 1. řádku napíšeme **neterminály, ze kterých se vygeneruje terminál** daného sloupce
  - další řádky vyplňujeme postupem, který se textem popisuje špatně :)
  - slovo náleží do jazyka dané CFG, pokud levé horní pole obsahuje počáteční neterminál

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 7 | {S, A, B} | | | | | | |
| 6 | {B} | {S, A, B} | | | | | |
| 5 | Ø | {B} | {A, B} | | | | |
| 4 | {C} | Ø | {B} | {A, B} | | | |
| 3 | {C} | Ø | Ø | {B} | {S, A} | | |
| 2 | {S} | Ø | {C} | Ø | {A} | {A} | |
| 1 | {S, A} | {A} | {C} | {C} | {S, B} | {S, B} | {S, A} |
| | a | c | d | d | b | b | a |

# I15 Datové struktury a jejich implementace

**(seznam, zásobník, fronta, binární strom, obecný strom, vyhledávací stromy a jejich modifikace, implementace binárních a vyhledávacích stromů a operací nad nimi)**

## Seznam

- = posloupnost prvků s **pevně stanoveným pořadím**
- implementace pomocí **dynamického pole**: rychlý přístup O(1), pomalé mazání a vkládání (až O(n)), vkládání může způsobit realokaci pole
- implementace pomocí **spojového seznamu**: pomalý přístup (až O(n)), rychlé vkládání a mazání O(1)
  - spojový seznam může být **oboustranný** nebo **jednostranný**

## Zásobník

- = seznam, jehož **jeden konec slouží k přidávání i odebírání záznamů**, **LIFO** struktura
- použití při prohledávání do hloubky, **volání funkcí** apod.
- implementace pomocí **pole**: opět může být nutná realokace při vkládání
- implementace **spojovým seznamem**

## Fronta

- = seznam, do nějž **vkládáme na jednom konci a odebíráme z něj na druhém**, **FIFO**
- použití při prohledávání do šířky, **load balancingu** apod.
- implementace pomocí **pole**: opět může být nutná realokace při vkládání
  - posouvání prvků v poli není příliš vhodné, je lepší si představit pole „**zatočené dokola**“
- implementace pomocí **spojovaného seznamu**
- speciální typy front
  - **obousměrná** – vkládání i odebírání na obou koncích
  - **prioritní** – prvky se ve frontě předbíhají podle priority

## Strom

- DS simulující strom z teorie grafů – **acyklický souvislý graf**
- v informatice se používá **zakořeněný strom** – jeden uzel je označen jako kořen
- implementován bývá pomocí **třídy reprezentující uzel** stromu, každý uzel nese data a má ukazatele na své **potomky**

    - uzly mohou mít i ukazatel na svého **rodiče**, ale zde je třeba hlídat konzistenci
- **výška stromu** je počet jeho hladin (počet uzlů na nejdelší větvi)
- **binární strom** je strom, jehož každý uzel má maximálně dva potomky

## Vyhledávací stromy

- **binární vyhledávací strom** = levý potomek uzlu má menší hodnotu, pravý potomek větší
    - pokud je **vyvážený**, umožňuje vyhledávání se složitostí **O(log n)**
- **dokonale vyvážený** strom = vzdálenost od kořene k listům je rozdílná max. o 1
    - AVL ani červeno-černý nejsou dokonale vyvážené, pouze vyvážené
- **AVL-strom**
    - výška levého a pravého podstromu se liší max. o 1 (informace uchovávaná v každém uzlu)
    - v případě porušení pravidla se prohazuje dle předem stanovených vzorců
    - **přidání** – přídá se jako list na správné místo, v případě nutnosti se vyvažuje
    - **odebrání** – prohodí se s listem (ne nutně potomkem), který je těsně menší, vyvažuje se
- **červeno-černý strom**
    - každý uzel má černou nebo červenou barvu (info uchovávané v každém uzlu)
    - kořen je černý
    - každý následník červeného je černý
    - každá větev obsahuje stejný počet černých
    - zaručuje, že z kořene do nejvzdálenějšího listu je cesta **max. 2x delší**, než do nejbližšího
    - **přidání** - přídá se jako list na správné místo, v případě nutnosti se vyvažuje
    - **odebrání** - prohodí se s listem (ne nutně potomkem), který je těsně menší, vyvažuje se
    - vyvažování je spojeno s přebarvováním uzlů
- **halda** = tvar binárního stromu, využití např. pro řazení
    - potomci každého uzlu jsou menší než on sám = **maximová**, analogicky **minimová**
    - v každé hladině je maximální počet uzlů (vyjma poslední)
    - poslední hladina jsou umístěny prvky co nejvíce vlevo
    - **přidání** prvku – přidat jako list a opravit, **odebrání** – vyměnit s nejpravějším listem, odebrat list, opravit

# I16 Třídění

**(základní algoritmy, algoritmy řazení haldou, slučováním, rozdělováním)**

## Select Sort (výběrem)

- najde **nejmenší prvek v nesetříděné části,** prohodí ho s **prvním prvkem nesetříděné části**; setříděnou část tímto zvětší o 1; opakuje
- $O(n^2)$, stabilní může být (záleží na implementaci)

## Insert Sort (vkládáním)

- **první prvek nesetříděné části** vloží do setříděné části na jeho správné místo, všechny **prvky setříděné části, které se nacházejí za ním, posune o 1** doprava; tímto zvětší setříděnou část o 1; opakuje
- $O(n^2)$, stabilní

## Bubble Sort (bublinkové)

- **prohazuje dva sousední** prvky, pokud jejich vzájemné pořadí není správné; po prvním průchodu je největší prvek poslední; **nesetříděnou část každým průchodem sníží o 1**
- může si pamatovat, zda v daném průchodu prohazoval – pokud na konci zjistí, že ne, znamená to, že pole již je setříděné a algoritmus může skončit
- $O(n^2)$, stabilní

## Heap Sort (haldou)

- prvky se naplní do haldy
  - rozdíl délky mezi všemi větvemi nejvýše 1
  - nejnižší úroveň se plní zleva
  - hodnoty uzlů na každé větvi jsou vzestupně / sestupně uspořádány
- odebere se kořen haldy (prohozením s nejpravějším listem a odebráním), halda se spraví
- $O(n \log n)$, nestabilní

## Merge Sort (slučováním)

- rekurzivně rozděluje pole na dvě přibližně stejně velké části až do velikosti 1, pak pole zpetně spojuje a předpokládá, že spojovaná pole jsou seřazená – trik je v tom, že **stačí porovnávat pouze první prvek z každého**
- $O(n \log n)$, stabilní, navíc n paměti, ale existuje in-place varianta

## Quick Sort (rozdělováním)

- **rekurzivně rozděluje čísla na menší a větší „polovinu“**
  - vybere **pivota** (často první prvek dané části)
  - prohazuje prvky mezi částmi pole „**větší než pivot**“ a „**menší než pivot**“
  - **pivota dá doprostřed** mezi ně (prohodí pivota s posledním prvkem skupiny)
  - zavolá znova **quicksort na každou polovinu**
- O(n log n), nejhorší $O(n^2)$, nestabilní, navíc log n paměti

# I17 Grafové algoritmy

**(procházení grafu do hloubky a do šířky, složitost procházení grafu)**

## Procházení grafu do hloubky (depth-first search)

- **expanduje vždy prvního následníka** daného uzlu (pokud jej ještě nenavštívil)
- uzly k navštívení si ukládá do **zásobníku** (**LIFO**) – při navštívení uzlu jej odebere ze zásobníku (+ přidá do mny navštívených, aby jej nenavštívil podruhé) a přidá na zásobník jeho potomky
- pokud z daného uzlu již nemůže pokračovat dál, vrací se výše a zkouší jinou cestu (stačí provést operaci *pop* ze zásobníku), tomuto se říká **backtracking**
- typy
  - **preorder** – navštívený, levý, pravý
  - **inorder** – levý, navštívený, pravý
  - **postorder** – levý, pravý, navštívený
- složitost
  - časová O(|V| + |E|)
  - prostorová O(|V|)

## Procházení grafu do šířky (breadth-first search)

- expanduje **postupně všechny následníky, až potom expanduje jejich potomky** (postupuje ve vrstvách)
- uzly k navštívení si ukládá do fronty (FIFO)
- složitost
  - časová O(|V| + |E|)
  - prostorová O(|V| + |E|), jinak taky (stupeň větvení stromu) ^ (max. hloubka) – exponenciální závislost na hloubce grafu
    - kvůli větší prostorové složitosti se nehodí na rozsáhlé grafy (stromy s větší hloubkou)

## Doplnění všeobecných znalostí

- **minimální kostra** grafu G
  - souvislý acyklický podgraf G s minimální váhou (spojíme všechny uzly nejkratšími cestami)

    - **Kruskalův** (Borůvkův) algoritmus
        - přidává **nejkratší hranu** v rámci **celého grafu**
    - **Primův** (Jarníkův) algoritmus
        - přidává **nejkratší hranu sousedící s dosavadně zpracovanou kostrou**
- **nejkratší cesty** v grafu
    - **Dijkstrův** algoritmus (nejpoužívanější)
        - z kandidátů na zpracování vybere uzel s nejmenší vzdáleností od výchozího bodu a zpracuje jej
        - všechny nezpracované sousedy zpracovávaného uzlu zařadí mezi kandidáty na zpracování a dopočítá jejich vzdálenost

# P1 Výpočetní systémy 1

**(číselné soustavy, vztahy mezi číselnými soustavami, zobrazení čísel v počítači, principy provádění aritmetických operací, Booleova, Shefferova a Peircova algebra, kobinační a sekvenční logické obvody)**

## Číselné soustavy

- **číselná soustava** = způsob reprezentace čísel
  - **nepoziční** (např. římské), sčítáním hodnot
  - **poziční** (hodnota číslice dána pozicí)
    - **polyadické** (základ = počet číslic, řád = pozice číslice) – dnes používané; $543 = 5 * 10^2 + 4 * 10^1 + 3 * 10^0$
- v počítačové praxi – binární (dvojková), osmičková (oktalová), šestnáctková (hexadecimální)
- **převody a vztahy mezi soustavami**
  - **do desítkové** soustavy převádíme roznásobováním (číslice) * $(\text{základ})^{(\text{řád})}$
    - platí i pro desetinná čísla
  - **celou část** čísla převedeme **z desítkové do jiné** takto
    - dělíme číslo základem sestavy, výsledek opět dělíme tak dlouho, **dokud nedostaneme podíl 0**, při tom si zapisujeme u každého dělení zbytek; přečteme **zbytky od posledního k prvnímu**, ty tvoří převedené číslo
  - **desetinnou část** převedeme **z desítkové do jiné** takto
    - desetinnou část v tabulce rozdělíme do **dvou sloupců** (případná celá část výsledku násobení, desetinná část výsledku násobení) a **násobíme základem** (i v dalších iteracích násobíme **vždy pouze desetinnou část**); čísla ve sloupci celé části čtená od prvního k poslednímu tvoří převedenou desetinnou část
  - snadno se převádí mezi soustavami o základu z a $z^k$ (např. 2 a 8, 2 a 16, ale ne 8 a 16)
    - **nahrazujeme po k-ticích číslic**

## Zobrazení čísel v počítači

- **celá čísla**
  - **kladná**
    - intuitivně, rozsah $<0; 2^n - 1>$
  - **záporná**
    - **přímý** kód se znaménkem = jako kladné, nejvyšší bit znaménkový; problém **dvou nul**

    - **inverzní** kód = pokud je číslo záporné (znam. bit 1), celé číslo **invertujeme**; **dvě nuly**
    - **doplňkový** kód = zobrazit kladné binárně, pak **invertovat, pak přičíst 1**; **jedna nula**
    - kód **s posunutou nulou** = přičteme posunutí (na 1B bývá 127); **jedna nula**
- **desetinná (jednoduchá přesnost, 4B)**
    - znam. bit (1b), exponent (8b, kód s posunutou nulou), mantisa (23b, přímý kód)
    - číslo má tvar $M * Z^E$
    - dvě nuly, problém nekonečna (řešen spec. hodnotami)
- **principy provádění aritmetických operací**
    - **doplňkový** kód = všechny bity se sčítají stejně
    - **inverzní** kód = problém dvou nul → **kruhový přenos** = přičtení přenosu z nejvyššího řádu
    - stejný algoritmus jako v dekadické soustavě

## Booleova, Shefferova a Peircova algebra

- **Booleova** algebra
    - **šestice** (A, ∧, ∨, ¬, 1, 0) = (mna literálů, konj., disj., neg., pravda, nepravda)
    - další fce – implikace, ekvivalence apod. odvoditelné ze základních
    - **De Morganovy** zákony (¬x ∧ ¬y) = ¬(x ∨ y), analogicky pro disjunkci
- **Shefferova** = jediná funkce NAND, ze které se poskládá zbytek
- **Peircova** = jediná funkce NOR, ze které se poskládá zbytek
- aplikace S. a P. v technice – je výhodné stavět mikroelektroniku z jednoho druhu součástek

## Kombinační a sekvenční logické obvody

- **kombinační** – základní: **invertor**, **and**, **or**, **nand**, **nor**, **xor**
- **sekvenční**
    - má vnitřní **stav**, **synchronní** (potřebuje signál timeru ke změně hodnoty) vs. **asynchronní**
    - **RS** = jednobitová paměť, vstupy S set a R reset, výstupy Q a ¬Q
    - **D** = časovačová jednobitová paměť, vstupy D a C clock, výstupy Q a ¬Q
    - **JK** = jako RS, ale s časovačem, vstupy J, K, C clock, výstupy Q a ¬Q (J∧K∧C invertuje)
    - složitější
        - čítače, multiplexory, dekodéry

# P2 Výpočetní systémy 2

**(procesory, jejich parametry a architektury; architektura Intel; vnitřní a vnější paměti a principy jejich funkce; vstupní a výstupní zařízení počítače a jejich připojování)**

- **koncepce Johna von Neumanna**
  - ALJ, operační paměť, v/v zařízení, řadič
  - binární instrukce i data uložena ve stejné paměti, označená adresami

## Procesory

- procesor je synchronní stroj řízený řadičem
- parametry a architektury
  - frekvence
  - délka operandu (16b, 32b, 64b, …)
  - počet jader
  - cache (L1 - nejrychlejší, L2 – pomalejší, sdílení mezi jádry)
  - pipelining = souběžné zpracování instrukcí
- architektury
  - CISC (complex instruction set computer) – x86
  - RISC (reduced …) – ARM, x86-64, ...

## Architektura Intel

- 8086
  - 16b, 10 MHz, 20b adresování (segment:offset), vnější a vnitřní přerušení
- 80286
  - 16b, 16 MHz, reálný (8086) a **chráněný** režim (4 úrovně), virtuální adresa až 1 GB (30b)
- 80386
  - 32b (fyz i log adresa), 40 MHz, podpora **stránkování**
- 80486
  - 32b, 160 MHz (DX4), + FPU, interní cache
- Pentium
  - 32b, 300 Mhz, předvídání podmíněných skoků, režim správy systému (podobný reálnému)
- x86_64

- 64b režim / kompatibilní režim

## Vnitřní a vnější paměti a principy jejich funkce

- vnitřní paměti – RAM, cache, registry, ...
  - **okamžitý přístup** kamkoliv, potřebují přísun proudu
  - větší kapacitu RAM simuluje **stránkování**
- vnější paměti – HDD, CD/DVD, flash disky, …
  - větší kapacita, trvalé uchování, přístupová doba / rychlost čtení na různých místech média se může lišit
  - pevné disky
    - nejmenší = sektor, soustředné kružnice = stopy, sektory přístupné bez pohybu čtecí hlavy = válec
    - starý přístup CHS (cylinder, head, sector), teď lineární LBA (logical block addressing)
    - souvislá alokace / zřetězený seznam / FAT / i-nodes (atributy, adresy)

## Vstupní a výstupní zařízení a jejich připojování

- zprostředkovávají interakci počítače s okolím
- příklady V/V zařízení: klávesnice, monitor, tiskárna, …
- připojování
  - USB
    - sériové, dnes nejtypičtější rozhraní
    - čtyřdrátová sběrnice (+5V, data -, data +, zem) + uzemňovací okraj konektoru
    - 1.0 (1996), 1.1 (1998), 2.0 (1999)
  - RS-232
    - starší, umožňovalo propojit 2 počítače přes tzv. **nulmodem**
    - až 15 V

# P3 Programování

**(strukturované programování v imperativním jazyce, datové a řídící struktury programovacích jazyků, datové typy, procedury a funkce, bloková a modulární struktura programu)**

## Strukturované programování v imperativním jazyce

- **imperativní jazyk** = program je posloupnost příkazů, které jsou postupně prováděny
- **strukturované programování** = řešený problém je rozdělen na dílčí úlohy, skládá se z menších bloků, které jsou na nejnižší úrovni tvořeny jednotlivými příkazy nebo voláním fcí

## Datové a řídící struktury

- **řídící struktury**
  - **větvení**
    - if (else if, else) = základní větvení dle podmínky
    - switch = vhodný pro více podmínek nad stejným výrazem
  - **cykly**
    - while = provádí cyklus, dokud je splněna podmínka
    - do-while = jako while, ale tělo cyklu je provedeno alespoň jednou
    - for = užívaný pro předem stanovený počet iterací (varianta foreach pro kolekce)
  - **skok** (goto)
    - dnes používán velice výjimečně, může znepřehledňovat program
- datové struktury
  - mají zjednodušit a zpřehlednit program, obzvlášť přínosné jsou při práci s mnoha proměnnými stejného typu
  - statické = nemůžou měnit rozsah
    - pole
  - dynamické = mění rozsah
    - dynamické pole, seznam
    - zásobník, fronta
    - strom, ...

## Datové typy

- definuje druh proměnných (= **doménu**, obor hodnot)
- **jednoduché**
  - **celočíselné**, **reálné**, **znak**, **logická hodnota**
- **strukturované**
  - **homogenní** – **pole**, **textový řetězec**, **výčtový typ**
  - **heterogenní** – **struktura**, **seznam** (= heterogenní pole)
- **abstraktní datové typy** = všeobecně použitelné datové struktury s operacemi
  - zásobník, seznam, fronta, množina, ...
- **generické datové typy** = parametrizovatelný jiným typem (třeba parametrizované ADT)

## Procedury a funkce

- funkce / procedura = ucelená **posloupnost instrukcí** (často označená jménem), která může dostat **na vstupu parametry**
  - funkce vrací hodnotu
  - procedura nevrací hodnotu
  - modernější jazyky procedury a funkce nerozlišují, místo procedur návratový typ *void*

## Bloková a modulární struktura programu

- **bloková**
  - rekurzivně zanořovatelná struktura bloků
  - blok má část **deklarační** a **příkazovou**
  - deklarace platí uvnitř bloku a vnořených bloků, **ne ve vnějších blocích**
- **modulární**
  - modul je ucelená programová jednotka, v ní deklarované proměnné a funkce jsou použitelné v jiných modulech
  - mívá specifikační a implementační část (např. knihovna v C)

# P4 Objektově orientované programování

**(základní pojmy OOP, zapouzdření, dědičnost, polymorfizmus, OOP v imperativním jazyce, spolupráce objektů; událostmi řízené programování; výjimky)**

## Základní pojmy

- **objekt** reprezentuje nějakou entitu reálného světa (ale i abstraktní); funguje jako černá skříňka navenek zpřístupněná rozhraním; má atributy (data) a metody (operace); objekt je **instancí třídy**
- **třída** = typ objektu (předloha pro vytváření objektů)
- **atributy** tvoří stav objektu, slouží k uchování dat
- **metody** definují operace, které jsou na objektu proveditelné
- **rozhraní** je množina hlaviček metod, které může být implementováno třídami; toto se využívá k tvorbě vyměnitelných komponent programu

## Koncepty OOP

- **zapouzdření** = objekt je černá skříňka s informacemi, komunikujeme s ním pomocí rozhraní, nemůžeme sahat dovnitř
- **dědičnost** umožňuje tvořit hierarchii tříd; potomci přebírají vlastnosti rodičovské třídy, které rozšiřují nebo pozměňují
- **polymorfizmus**
  - na místě rodiče může vystupovat potomek
  - potomek může přepsat (override) metody rodiče

## Spolupráce objektů, událostmi řízené programování

- OO program je tvořen objekty, které **vznikají, zanikají** a komunikují pomocí **zasílání zpráv** (= volají navzájem své metody)
- OO návrh spolupráce objektů
  - **volná vazba** (loose coupling) = minimalizovat počet závislostí mezi třídami
  - **silná koheze** (strong cohesion) = třída poskytuje ucelenou, silně související funkcionalitu
- často se komponenty dělají **vyměnitelně** – implementace rozhraní
- **událostmi řízené programování**
  - často používané u programů s GUI, většinou vícevláknové
  - objekty mají události (kliknutí, načtení, zavření, ...) a programátor k nim vytváří listenery

## Výjimky

- **výjimka** = objekt s informacemi o chybě programu, slouží k ošetření výjimečných situací
- **hlídané** (pokud může vzniknout, musíme odchytnout) / **nehlídané** (nemusíme odchytnout)
- **ošetření** = při výjimce program přeskočí do korespondujícího catch bloku
  - **propagace** = pokud výjimka nemá catch blok na stávající úrovni, hledá se na vyšších úrovních (výjimka se propaguje po call stacku), pokud se catch blok vůbec nenajde, program se násilně ukončí

# P5 Operační systémy

**(architektury operačních systémů, rozhraní operačních systémů; procesy, synchronizace procesů a metody ochrany proti uváznutí; práce s pamětí, logický a fyzický adresový prostor, správa paměti a způsoby jejího provádění)**

## Architektury a rozhraní operačních systémů

- **architektury**
  - **mikrokernel** = v jádře jen velice jednoduché základní funkce, zbytek v nadstavbách
  - **makrokernel** (monolitické) = obsahuje velké množství fcí pro různé aspekty systému
  - **modulární** = kompromis, fakticky makrojádro (v privilegovaném režimu), ale jeho části (moduly) je možné přidávat a odebírat za běhu
- **rozhraní**
  - **uživatelské** = pro člověka – shell, GUI
  - **rozhraní služeb** = pro software – funkce OS pro práci s různými podsystémy (souborový, RAM, vlákna, síť, ...)

## Procesy, synchronizace a metody proti uváznutí

- **proces** = program zavedený do operační paměti a data související s jeho instancí, má PID
  - **vlákno** = „odlehčený proces“, více v 1 procesu, sdílí s ostatními vlákny paměť (prostředky)
- **multitasking** = procesor přepíná mezi procesy a uděluje jim procesorový čas
- **stavy procesu** = nový (vytvářen), připravený (čeká na procesor), běžící (právě vykonáván procesorem), čekající (čeká na nějakou událost, tj. v/v apod.), ukončený
- **synchronizace**
  - řeší **souběžné sdílení prostředků** (paměť, zařízení, soubory, …), protože při souběžném přístupu mohou vznikat **časově závislé chyby a nekonzistence**
  - **kritická sekce** = prostředek **vzájemného vyloučení** – brání více procesům/vláknům nacházet se ve stejném místě programu zároveň
    - **výhradní přístup** (pouze 1 proces), **omezené čekání** (ne nekonečně)
- **uváznutí**
  - **nutné podmínky**: vzájemné vyloučení (prostředek si zamyká 1 proces), inkrementálnost požadavků (žádají postupně), nepředbíhatelnost (není možné procesu sebrat prostředek)
  - **postačující podmínka**: cyklické čekání (A na B, B na C, C na A)
  - **metody proti uváznutí**

    - ignorace – nezabývá se deadlockem – např. UNIX
    - prevence – ruší se platnost některé nutné podmínky nebo současná platnost všech
    - detekce – detekuje se existence a řeší se následky

## Práce s pamětí, logický a fyzický adresový prostor

- při kompilaci se neví, na jaké fyzické adrese v RAM bude program umístěn při spuštění → je nutné převádět odkazy v programu (LAP) na skutečné odkazy ve fyzické paměti (FAP)
- logický adresový prostor
    - virtuální adresový prostor, jádro i každý proces má svůj vlastní
    - kapacita je dána šířkou adresy v instrukci
- fyzický adresový prostor
    - skutečný adresový prostor RAM paměti
    - kapacita je dána velikostí RAM na daném počítači

## Správa paměti a způsoby jejího provádění

- paměť bývá rozdělena na rezidentní část OS (na začátku) a prostor pro programy
- **ochrana**
    - program nesmí bez povolení zasáhnout do paměti jiného programu nebo jádra
- **sdílení**
    - více procesů může sdílet stejnou paměť (obzvlášť konstantní, např. instrukce)
- **způsoby přidělování**
    - first-fit (nejčastěji používaný)
    - best-fit
    - worst-fit
- **fragmentace**
    - vnitřní – aplikace nevyužije přidělenou paměť
    - vnější – volné paměti je dost, ale ne v souvislém bloku
- **stránkování**
    - FAP se dělí na rámce, LAP na stránky (stejné velikosti, např. 4KB)
    - překlad se řeší pomocí page table

# P6 Plánování v operačních systémech

**(správa a plánování činnosti procesorů, systémy souborů, správa a plánování v/v zařízení)**

## Správa a plánování činnosti procesorů

- = rozhodování, kterému procesu má být přiřazen procesorový čas
- **krátkodobé** = vybírá z ready procesů ten, který poběží
- **střednědobé** = swapuje na disk → ready/blocked a ready suspended / blocked suspended
- **dlouhodobé** = při spouštění nového procesu vybírá, která úloha bude spuštěna (význam spíš při dávkovém zpracování)
- **preemptivní** = s předbíháním, lze procesu odebrat procesor
- **nepreemptivní** = bez předbíhání, musí se vzdát dobrovolně
- **plánovací algoritmy**
  - **first come first serve** = nepreemptivní s FIFO frontou
  - **prioritní** = procesům s vyšší prioritou dá více času, může být nepreemptivní i preemptivní
    - preemptivní varianta prioritního plánování je běžná ve většině OS
  - **shortest job first** = napřed ty co potřebují nejkratší dávku (většinou ale není známa, používá se exponenciální průměrování)
  - **round robin** = preemptivní s FIFO frontou

## Systémy souborů

- zajišťuje pojmenování souborů a jejich uspořádání v adresářích, poskytuje standardní rozhraní pro IO operace, optimalizuje výkon, brání poškození, …
- **operace**: vytváření, mazání; otevírání, zavírání; zapisování, čtení, mazání záznamů, seek
- **zamykání** souborů
  - **sdílený** zámek – všichni smí číst, nikdo zapisovat
  - **exkluzivní** zámek – přistupuje výhradně jeden proces
- **přístup**
  - **sekvenční** (mag. páska)
  - **přímý** (HDD)
- **plánovací algoritmy**
  - **first come first served** = fronta, neefektivní, může hodně skákat

- **shortest seek time first** = obsluhuje nejbližší; hrozí stárnutí požadavků
- **scan** = jezdí tam a zpátky, přitom obsluhuje; nevýhodné pro data na krajích disku
- **c-scan** = vyřizuje požadavky pouze jedním směrem, na druhou stranu se vrací bez čtení
- **c-look** = jako c-scan, ale nejede až na konec disku, pouze po poslední/první požadavek
- často bývá implicitní volba **sstf** nebo **c-look**

## Správa a plánování V/V zařízení

- **druhy**
  - **blokující** = program stojí, dokud není IO hotovo
  - **neblokující**
    - **polling** = forma činného čekání, program se neustále ptá, jestli je IO hotovo
    - **přerušení** = program dostane informaci o dokončení IO přerušením
- **DMA** (direct memory access)
  - náhrada programového IO, data nepřesouvá procesor, ale DMA řadič, který přímo přistupuje do paměti; procesor je během IO nezatížen; dnes používané pro všechny HDD
- **techniky plánování**
  - optimalizuje výkon, ale není nezbytné
  - **buffering** = vyrovnávací paměť pokud zařízení přijímá data pomaleji než je program generuje, nebo program přijímá pomaleji než zařízení generuje
  - **caching** – kopie dat v paměti, zvětšuje výkon
  - **spooling** – požadavky se hromadí do fronty (např. tiskárna)
  - **rezervace** – exkluzivní přístup k zařízení

# P7 Počítačové sítě

**(topologie, přístupové metody a architektury počítačových sítí (Ethernet, Fast Ethernet, Token-ring, ATM, …); bezdrátové komunikační technologie; model OSI; protokol TCP/IP; propojování počítačových sítí a směrování informací)**

## Model OSI

- **fyzická** = elektrické a fyzikální vlastnosti zařízení, „vrstva drátů“
- **datových spojů** = řeší spojení mezi sousedními zařízeními, „vrstva switchů“
- **síťová** = směrování v síti a síťové adresování, „vrstva routerů“, protokol IP
- **transportní** = přenos dat mezi konc. uzly, protokoly TCP, UDP
- **relační** = ustavení, udržení a ukončení relace mezi uzly, protokol SSL
- **prezentační** = transformace dat do tvaru, které používají aplikace
- **aplikační** = poskytnout aplikacím přístup ke komunikačnímu systému a umožnit spolupráci

## Topologie, přístupové metody a architektury

- **topologie** – bus, star, ring, mesh
- architektury a přístupové metody
  - **MAC** (medium access control) protokoly bývají založeny na: soupeření, rezervaci, předávání, nebo jejich kombinaci
  - **se soupeřením**
    - Aloha
    - **CSMA** (carrier sense multiple access) – nesynchronizovaná kolizní metoda
      - nenaléhající, 1-naléhající, p-naléhající
    - **CSMA/CD** (s vysíláním poslouchá, nejde u rádiových přenosů)
    - **CSMA/CA** (použití u **bezdrátových sítí**) – synchronizované začátky čekání
    - **Ethernet** = 1-naléhající CSMA/CD s exponenciálním růstem čekacího intervalu
      - při rostoucím počtu stanic přestává být efektivní, nutnost switche
  - **s rezervací**
    - TDMA (time division multiple access) – pevně stanovená časová okna (např. GSM)
    - SDH / SONET – striktně point-to-point,
    - ATM – spojovaná služba, zajišťuje QoS

    - **s předáváním**
        - token ring / token bus
        - nutné řešit problémy při ztrátě tokenu

## Bezdrátové komunikační technologie

- **ad-hoc** (krátkodobé, ustanovené dynamicky) vs. **infrastrukturní** (stabilní, dlouhodobé)
- typy
    - **DSSS** (direct sequence spread spectrum) – velká redundance, ztracená data lze spočítat
    - **FHSS** (freq. hopping spread spectrum) – krátce nosná, pak skok na náhodnou
- **Wi-Fi** – DSSS, ad-hoc i infrastrukturní, 2.4 / 5 GHz, až 54 Mb/s
- **Bluetooth** – FHSS, 2.4 GHz, až 720 Kb/s, master-slave, max. 8 zařízení
- **mobilní** – GSM, GPRS, EDGE, UMTS

## Protokol TCP/IP

- stejně jako UDP používá 65 tisíc portů
- poskytuje **zaručený proud slabik**, **zachovává pořadí** paketů (čísluje je), používá **piggybacking** (potvrzení přilepuje k dalším datům)
- principy
    - **pomalý start** = začíná se s velice malým objemem dat, exponenciálně se zvětšuje, až dosáhne určité velikosti, pokračuje se metodou *zábrana zahlcení*
    - **zábrana zahlcení** = objem dat se zvětšuje pouze lineárně
    - **rychlá retransmise** = reakce na ztrátu segmentu, následuje opětovné zaslání
    - **rychlé vzpamatování** = zabránění návratu do fáze *pomalý start* při ztrátě paketů

## Propojování a směrování

- **statické** (známe topologii, ručně zadané), **dynamické** (adaptabilní, dočasné nekonzistence)
- **směrování v Internetu** – dynamické, krok za krokem, distribuované, deterministické, jednocestné
- směrovací algoritmy
    - **distance vector** (DV)
        - šíří se vzdálenost k cílům (cesty)
    - **link state** (LS)
        - šíří se topologie, cesty si směrovače dopočítávají samy Dijkstrou

# P8 Organizace souborů

**(schémata o.s.; statické o.s.: sekvenční soubory, indexové a přímé o.s., statické hašování; dynamické o.s.: dynamické hašování, B-stromy a jejich varianty; základy teorie informace, komprese dat)**

- **schéma organizace souborů** popisuje, jak jsou data uložena do paměti, optimalizuje
  - typicky obsahují primární soubor s daty a pomocné soubory (indexy, rejstříky)

## Statické organizace souborů

- **homogenní** (každý záznam stejného typu) vs. **nehomogenní** (záznamy různých typů, těmi se nebudeme dále zabývat)
- **sekvenční soubor** = homogenní soubor se sekvenčním přístupem
  - **uspořádaný** podle klíče → náročná reorganizace při vkládání/mazání
  - vylepšením může být **řetězená struktura** = každý záznam má ukazatel na následníka
- **index-sekvenční soubor**
  - soubor **uspořádaný** podle primárního klíče, je k němu vytvořena **struktura indexů**, k datům se přistupuje libovolným z těchto dvou způsobů
- **indexovaný soubor**
  - **uspořádává se index**, primární soubor je **neuspořádaný**
- **soubor s přímým přístupem**
  - **algoritmická transformace** vyhledávacího klíče na adresu záznamu, nejčastěji **hašování**
  - **statické hašování**
    - **hašování** = převedení libovolně dlouhého vstupu na výstup pevné délky
    - vždy se používá stejná část haše – mohou vznikat místa s přeplněnými buckety a prázdná místa

## Dynamické organizace souborů

- **dynamické hašování**
  - k výpočtu se používá **prvních *i* bitů** z výstupu hašovací fce – toto ***i* se dynamicky mění**, může se zvětšovat i zmenšovat a **v rámci jednoho souboru mít různou velikost** pro různé hodnoty hašovací funkce
- **B strom**
  - uzel stromu obsahuje jak **ukazatele na potomky**, tak **hodnoty klíčů** (pro strom řádu m obsahuje každý uzel m-1 klíčů)

    - každý uzel až na kořen a listy má aspoň m/2 potomků
    - kořen má aspoň 2 potomky, pokud není jediným uzlem stromu
    - **klíče se v celém stromu vyskytují právě jednou**
    - vlevo od klíče ukazatel na potomka s menšími hodnotami, vpravo s většími
- **B+ strom**
    - **klíče se mohou ve stromě opakovat**
    - všechny klíče stromu jsou také v listech, listy jsou **zřetězené → rychlé sekvenční procházení**
- B* strom (B strom, ale minimální obsazení uzlu je 2/3, ne 1/2)

## Základy teorie informace, komprese dat

- entropie = míra neurčitosti; velikost odstraněné neurčitosti = množství získané informace
- kódování = nahrazování jedné posloupnosti symbolů jinou
- komprese
    - **neztrátová** (získáme původní data), **ztrátová** (menší, ale menší přesnost rekonstrukce)
        - je-li H entropie zdroje, pak lze **neztrátově komprimovat** až do vyjádření každého symbolu **průměrně H bity**
    - **statická** (neměnnný algoritmus), **adaptivní** (přizpůsobuje se datům, zefektivňuje se tak)
    - **fyzická** (ignoruje význam dat), **logická** (zohledňuje význam, typicky je ztrátová)
- typy
    - **základní (intuitivní)** – např. run-length (symbol a číslo, kolikrát se vyskytuje)
    - **statistické** – např. Huffmanovo (na základě pravděpodobnosti výskytu)
        - **aritmetické** – funkce mapující posloupnost do intervalu <0; 1), rekurzivně dělí intervaly
    - **slovníkové** – např. LZW (kóduje se po často se vyskytujících vzorcích, ne po symbolech)
    - **další** – MP3, JPEG, ...

# P9 Databáze 1

**(relační model, relační schéma, klíče relačních schémat, integritní omezení, relační algebra, spojování relací)**

## Relační model, relační schéma

- **relační model** dat = logický model dat; data jsou tvořena **relacemi**
- **relace** = podmnožina kartézského součinu množin, kterým říkáme **atributy** (relace odpovídá **všem** datům v jedné tabulce)
- **relační schéma** = je (uspořádaná) mna atributů (odpovídá definici tabulky)
- **atribut** = pojmenovaná množina možných hodnot (má **doménu**) (odpovídá definici sloupce)

## Klíče relačních schémat, integritní omezení

- **klíč** je část relačního schématu, podmnožina jeho atributů
- **superklíč** je klíč dostatečný pro jednoznačnou identifikaci záznamu
- **kandidátní klíč** je minimální superklíč (po odebrání jakéhokoliv atributu už by neidentifikoval jednoznačně)
- **primární klíč** je jeden zvolený kandidátní klíč
- **integritní omezení**
  - **entitní** – každé relační schema má primární klíč, jehož hodnota je jedinečná napříč záznamy
  - **doménové** – sloupce mají domény (datové typy), navíc klauzule check
  - **referenční integrita** – pokud entita ukazuje na jinou prostřednictvím prim. klíče, kontroluje existenci cílové entity

## Relační algebra

- umožňuje provádět operace s konečnými relacemi
- základní operace
  - **selekce** = omezuje požadované záznamy v relaci podmínkou na atributech (vybírá řádky)
  - **projekce** = omezuje výsledek na vybrané atributy (vybírá sloupce)
  - **přejmenování** = mění jméno relace a atributů
  - **sjednocení** = klasické, relace musejí mít stejnou aritu a kompatibilní domény
  - **rozdíl** = klasické, relace musejí mít stejnou aritu a kompatibilní domény

- **kartézský součin** = klasický, nesmí dojít ke kolizi jmen atributů

## Spojování relací

- **vnitřní spojení** (**inner**) = kartézský součin omezený požadavkem na shodnost v daném atributu
  - **přirozené spojení** (**natural**) = požadavek na shodnost ve stejně pojmenovaném atributu
- **vnější** (**outer**) zahrnuje i prvky, které nelze navázat shodnou hodnotou vybraného atributu
  - **left** = všechny prvky z levé relace
  - **right** = všechny prvky z pravé relace
  - **full** = všechny prvky z levé i pravé relace

# P10 Databáze 2

**(funkční závislosti; klíče relačních schémat; Armstrongovy axiomy; dekompozice relačních schémat; normální formy obecně, 1NF, 2NF, 3NF, Boyce-Coddova NF, vztahy mezi NF; převody relačních schémat do NF)**

## Funkční závislosti, klíče relačních schémat

- **Y je funkčně závislé na X**, píšeme **X → Y**, pokud platí, že **mají-li dva prvky stejnou hodnotu X, pak mají také stejnou hodnotu Y**
- **klíč** je část relačního schématu, podmnožina jeho atributů
- **superklíč** je klíč dostatečný pro jednoznačnou identifikaci záznamu (pokud platí K → R)
- **kandidátní klíč** je minimální superklíč (po odebrání jakéhokoliv atributu už by neidentifikoval jednoznačně)
- **primární klíč** je jeden zvolený kandidátní klíč

## Armstrongovy axiomy

- pro danou mnu funkčních závislostí existují další funkční závislosti, které F implikuje, tzv. **uzávěr množiny F**, značíme **$F^+$**; můžeme je nají pomocí **Armstrongových axiomů**
  - **reflexivita**: je-li b ⊆ a, pak a → b
  - **rozšíření**: je-li a → b, pak c ∪ a → c ∪ b
  - **tranzitivita**: je-li a → b a b → c, pak a → c
- lze z nich odvodit další:
  - **sjednocení**: je-li a → b, a → c, pak a → b ∪ c
  - **rozklad**: je-li a → bc, pak a → b, a → c
  - **pseudotranzitivita**: je-li a → b, bd → c, pak a → c

## Normální formy (1NF, 2NF, 3NF, BCNF) a vztahy mezi nimi

- **neprimární atribut** = atribut, který není součástí žádného kandidátního klíče
- **1NF**
  - Relační schéma R je v první normální formě, když **každý jeho atribut je atomický** (= dále nedělitelný).
- **2NF**
  - Relační schéma R je v druhé normální formě, když je v **první normální formě** a každý neprimární atribut platí že **je závislý na každém celém kandidátním klíči**.

- **3NF**
  - Relační schéma R je v třetí normální formě, když je v **druhé normální formě** a každý neprimární atribut A j**e netranzitivně závislý na každém kandidátním klíči**.
- **BCNF**
  - Relační schéma R je v Boyce-Coddově normální formě, když je ve **třetí normální formě** a **všechny závislosti** v relačním schématu jsou **na kandidátních klíčích nebo jejich nadmnožinách**.
  - Ne vždy je možné BCNF dosáhnout.
- **vztahy mezi NF**: Každá NF obsahuje jako **nutnou podmínku všechny nižší NF**.

## Dekompozice, převody do NF

- **dekompozice** = rozklad relačního schématu na více relačních schémat a jejich propojení pomocí primárních a cizích klíčů
  - vždy platí, že zpětné spojení musí být **bezztrátové**
- **převod z nižších NF do vyšších NF** se řeší právě dekompozicí

# P11 SQL

**(syntax a sémantika příkazů; vestavěné funkce, triggery, uložené procedury; příkazy pro definici dat; transakční zpracování; atomické operace; optimalizace dotazů)**

## Sytnax a sémantika

- **syntax** byla navržena tak, aby připomínala **obyčejný jazyk**
  - na začátku klíčová slova příkazů, pak se k nim specifikují parametry
  - je zvykem psát klíčová slova velkými písmeny
- **sémanticky** se příkazy dělí do 3 skupin:
  - **DDL** (data definition lang.) = manipulace se schématem databáze
  - **DML** (data manipulation lang.) = manipulace s daty (s obsahem tabulek)
  - **DCL** (data control lang.) = řízení práce s daty – transakce, přístupová práva apod.

## Vestavěné funkce, triggery, uložené procedury

- **vestavěné funkce**
  - běžné **agregační** fce jako COUNT, AVG, SUM, MIN, MAX, …
  - fce pro práci s **řetězci** (SUBSTRING, LOWER, UPPER, …)
  - velice se liší dle použitého DBMS
- **uložené procedury** = v databázi uložené programy, které pracují s daty
  - mají své parametry, lokální proměnné, funkce mohou vracet hodnoty (třeba celé relace)
- **triggery** = uložené procedury spouštěné automaticky při manipulaci s daty
  - after (insert | update | delete)
  - before (insert | update | delete)

## Příkazy pro definici dat

- CREATE TABLE tab (tab_id Int PRIMARY KEY, val VARCHAR, val2 BIGINT)
- ALTER TABLE ADD COLUMN val3 BOOLEAN
- DROP TABLE

## Transakční zpracování, atomické operace

- **atomická operace** je nedělitelná – dojde-li k chybě uprostřed zpracování, nebude mít žádný efekt
- **transakce** je posloupnost DML příkazů, které převedou db schéma z jednoho konzistentního stavu do druhého
  - A – atomic – provede se celá, nebo vůbec
  - C – consistent – po provedení je databáze v konzistentním stavu
  - I – isolated – je oddělená od ostatních transakcí
  - D – durable – po ukončení jsou data trvale uložena
- **izolovanost**
  - **read uncommitted** – brání souběžné aktualizaci, ale nebrání čtení nepotvrzených změn
  - **read committed** – zabrání i čtení nepotvrzené změny, ale nebrání fant. a neopak. čtení
  - **repeatable read** – zabrání neopakovatelnému čtení, nezabrání fantomovému čtení
  - **serializable** – transakce se provedou tak, jako by probíhaly vždy jedna po druhé
- **problémy**
  - **neopakovatelné čtení** – v transakci projde první select, stejný druhý neprojde (mezitím už někdo zamkl tabulku)
  - **fantomové čtení** – v transakci dva stejné selecty za sebou vracejí jiné hodnoty

## Optimalizace dotazů

- **index** = nejdůležitější a nejúčinnější urychlení dotazů je tvorba indexů nad správnými sloupci
- **denormalizace tabulek** = větší rychlost za cenu hrozby nekonzistence
- **podmínky dotazu** zapisovat od více selektivních k méně selektivním
- **pořadí spojení** také od více selektivních k méně selektivním
- většina DBMS podporuje zobrazení **plánu dotazu**, kde lze zkontrolovat chybějící indexy apod.

# P12 Základy datového modelování

**(návrh datových struktur; ER diagramy; entity, atributy, vztahy; grafické vyjádření)**

## Návrh datových struktur

- cílem **datového modelování** je navrhnout kvalitní **datovou strukturu** pro konkrétní aplikaci
- k popisu datové struktury slouží **konceptuální datový model –** definuje entity, jejich atributy a vzájemné vztahy; je nezávislý na implementačním prostředí
- zkonkrétněním konceptuálního datového modelu pro určitý DBMS získáme **datový model** závislý na implementačním prostředí, podle kterého pak systém můžeme vytvořit

## Entity, atributy

- **entita** = **objekt**, který existuje (ale může být abstraktní), je odlišitelný od ostatních a je potřebné o něm uchovávat informace
- **entitní třída** = **skupina entit** stejného typu, které sdílejí stejné vlastnosti (**mnu atributů**)
- **atribut** = **popisná vlastnost** entitní třídy nebo vztahu, jejíž hodnotu chceme uchovat v systému a používat
  - má určenou **doménu** (tj. datový typ, mnu povolených hodnot)
    - jednoduché (např. rok narození)
    - složené (např. datum)
    - odvozené (např. věk)
- **klíče**
  - **klíč** je podmnožina atributů entity
  - **superklíč** je klíč dostatečný pro jednoznačnou identifikaci entity
  - **kandidátní klíč** je minimální superklíč (po odebrání jakéhokoliv atributu už by neidentifikoval jednoznačně)
  - **primární klíč** je jeden zvolený kandidátní klíč

## Vztahy

- **vztah** je spojení mezi 2 a více entitami, které evidujeme a případně o něm uchováváme další informace
  - v databázi je realizován primárními a cizími klíči
- **vztahová množina** je množina vztahů stejného druhu
- **stupeň vztahu** označuje počet entitních tříd, které vystupují ve vztahové množině

- **četnost vztahu** označuje počet entit, se kterými mohou být ostatní entity propojeny (1:1, 1:N, M:N)
- **existenční závislost** = existence entity x (**slabá**) závisí na existenci entity y (**silná**)
  - primární klíč slabé entitní třídy je tvořen **prim. klíčem silné entitní třídy a parciálním klíčem slabé entitní třídy**
- **specializace**
  - **úplná** = nemůže existovat entita vyšší třídy, která by nebyla zároveň entitou některé z nižších tříd; **odpovídá abstraktním třídám** v OOP
  - **částečná** = může existovat entita vyšší třídy, která není specializovaná

## ER diagramy, grafické vyjádření

- **entitně relační model je konceptuální model**, ze kterého se odvozuje **schéma databáze**
- existují různé ER notace
  - **entity** jsou obdélníky
  - **atribut** primárního klíče je podtržený
  - **Chenova** (původí)
    - **atribut** je elipsa
    - **slabé entity** dvojitě orámované
    - **vztah** v kosočtverci, **kardinalita** číslem
    - **dědičnost** trojúhelníkem postaveným na špičku, nahoře je rodič
  - **Crow's Foot**
    - prostorově úspornější
    - **slabé entity** mají kulaté rohy
    - **atributy** jsou vypsány uvnitř entity
    - **vztah** pouze čára, **kardinalita** graficky na koncích vztahu
    - **dědičnost** podtrženým kolečkem
  - UML má také ERD notaci, ale tam už se spíš používají **diagramy analytických tříd**

# P13 Analýza a návrh systémů

**(problémy spojené s řešením rozsáhlých systémů; empirické zákony softwarového inženýrství; modelovací nástroje funkční a datové dekompozice; konzistence modelu; metody strukturované analýzy, yourdonova metoda; strukturovaný návrh, nástroje metody, metriky a heuristiky návrhu)**

## Problémy spojené s řešením rozsáhlých systémů, empirické zákony

- **problémy**
  - **složitost, velikost** – mnoho zdrojového kódu, mnoho míst pro potenciální chybu
  - **komunikace** – s větším týmem narůstá počet komunikačních kanálů
  - **čas, plán** – problém správného odhadu a včasného řešení problémů
  - **neviditelnost sw** – u zákazníka se dlouho nic neděje, je nervózní
- **empirické zákony** softwarového inženýrství
  - **Lehmanovy zákony**
    - **trvalé proměny** = systém se neustále mění, dokud není lepší restruktur. / udělat nový
    - **rostoucí složitosti** = evolucí se program znepřehledňuje a zvyšuje se vnitřní složitost
    - **invariantní spotřeby práce** = rychlost vývoje je +- konst., ne dle vynaložených prostř.
    - **omezené velikosti přírůstku** = změny po malých krůčcích, nebo se systém sesype
  - **Brooksův zákon** = přidání dalších řešitelů projekt ještě více zpozdí

## Modelovací nástroje funkční a datové dekompozice, konzistence modelu

- **DFD** = zobrazení systému jako síť procesů, které si mezi sebou předávají data
  - terminátory, procesy, datové toky, paměti
  - CDFD = rozšířená varianta DFD, jsou na něm zakresleny i řídící procesy (jejich úkolem je řízení a synchronizace systému, funkce popsána pomocí STD)
- **Seznam událostí** = textový výčet podnětů vnějšího světa, na které systém reaguje
  - flow, temporal, control
- **Minispecifikace** = podrobná logika nejnižší úrovně DFD procesů, pseudokód
- **STD** = stavově přechodový diagram (stavy, přechody, podmínky a akce)
- **ERD** = (konceptuální) datový model (entity, vztahy)
- **DD** = datový slovník, výčet datových prvků (entit, atributů, …) systému
- **konzistence** se udržuje kontrolou výskytu elementů navzájem mezi diagramy (DD – ERD,

DD – DFD, DFD – Minisp., ...)

## Metody strukturované analýzy, Yourdonova metoda

- **Yourdonova metoda**
  - **esenciální model** = okolí + chování
  - **postup**
    - **model okolí** (účel, kontextový diagram, seznam událostí)
    - **prvotní model chování** (DFD, postup zdola nahoru, ERD, stavový model, minispecifik.)
    - **dokončení esenciálního modelu** (vyvažování DFD, dokončení minispec. a ERD)
    - **implementační model** (které procesy automatizovat (systém), které ručně (terminátor))
- **Strukturovaná analýza a specifikace** (**SASS**)
  - **postup**: studie stávajícího fyz. systému, odvození ekvivalentního log. systému, odvození nového log. systému, vytvoření nového fyz. systému
- **SSADM**
  - detailně specifikovaná a strukturovaná v každé etapě
  - **postup**: analýza stávajícího systému, specifikace požadavků, výběr tech. možností, logická data, logické procesy, fyzický návrh

## Nástroje, metody, metriky a heuristiky návrhu

- **transformační analýza**
  - restrukturalizuje DFD, snaží se vyfaktorovat transformační centrum, zleva dává vstupní toky, zprava výstupní
  - **transformační centrum** modeluje část systému, kde dochází ke klíčovému zpracování dat – **vstupní data se zde mění na výstupní**
- **transakční analýza**
  - restrukturalizuje DFD, snaží se vyfaktorovat transakční centrum
  - **transakční centrum** modeluje část systému, kde dochází k rozhodování – **větvení toků**
- **jacksonovy strukturogramy** – dělí operace na podoperace s užitím sekvencí, iterací a výběrů
- **heuristiky**
  - návrh se snaží **redukovat propojení** (závislosti) **a zvýšit kohezi** (soudržnost) **modulů**
    - ideální moduly jsou „jeden vstup, jeden výstup“, řeší jednu věc a propojení minimální
  - minimalizace rozšiřujících se hierarchických struktur

# P14 Objektově orientovaná analýza a návrh

**(nástroje UML, modely různých aspektů systémů v UML, vysvětlení a aplikace empirických zákonů)**

## Nástroje UML, modely různých aspektů systémů v UML

- UML (Unified Modelling Language) je standardní jazyk pro specifikaci, konstrukci a dokumentaci SW systémů
- kombinuje datové modelování, modelování procesů, objektů a komponent
- poskytuje nástroje pro mnoho fází tvorby software, od specifikace požadavků až po vizualizaci průběhu metody v programu
- **diagramy** bývají budovány nad **modelem** – změna názvu v modelu změní název ve všech diagramech → snadnější udržování konzistence
- **Diagram případů užití** (Use Case)
  - poskytuje vnější pohled na systém, zachycuje vztahy mezi účastníky a případy užití
  - vztah «uses» = chování společné více případům užití
  - vztah «extends» = volitelné, rozšiřující chování
- **Diagram posloupností** (Sequence)
  - „plavecké dráhy“, čas jde vertikálně shora dolů, vedle sebe jsou uspořádané objekty, šipky mezi nimi určují, jak si navzájem předávají kontrolu
- **Diagram komunikace** (Communication, dříve spolupráce/collaboration)
  - ukazuje interakci objektů a jejich propojení; je příbuzný s diagramem posloupností ale nedává důraz na časové hledisko
- **Diagram tříd** (Class)
  - ukazuje existenci tříd a jejich vzájemných vztahů
  - **třída** – jméno, atributy, metody
  - **vztahy** – asociace (obecný, často obousměrná), agregace (celek – část), kompozice (definující, silná agregace), dědičnost (rodič – potomek), závislost (poskytovatel – klient, často jednosměrná)
    - násobnosti, směry, jména rolí
- **Stavový diagram** (State)
  - znázorňuje životní cyklus objektu, který má stavy
  - **stavy** objektu; události způsobující přechody mezi stavy; **akce** k provedení při přechodu
- **Diagram komponent** (Component)

    - znázorňuje rozklad systému na ucelené samostatné komponenty
- **Diagram nasazení** (Deployment)
    - znázorňuje nasazení systému na hardware

## Vysvětlení a aplikace empirických zákonů

- **Lehmanovy zákony**
    - **trvalé proměny** = systém se neustále mění, dokud není lepší restrukturalizovat nebo udělat nový
    - **rostoucí složitosti** = evolucí se program znepřehledňuje a zvyšuje se vnitřní složitost
    - **invariantní spotřeby práce** = rychlost vývoje je +- konst., ne dle vynaložených prostř.
    - **omezené velikosti přírůstku** = změny po malých krůčcích, nebo se systém sesype
- **Brooksův zákon** = přidání dalších řešitelů projekt ještě více zpozdí

## Poznámky navíc

- objektově orientovaný přístup je **přirozenější** než strukturovaný
    - snadněji pochopitelný pro laiky, lepší **komunikace se zákazníkem**
    - odolnější vůči změnám, zvýšení **konzistence** modelu, lépe **udržovatelný**