

Přehled Chomského hierarchie formálních jazyků a gramatik. Turingovy stroje.

Třídy problémů P, NP, NP-těžký, NP-úplný

BI-AAG

- **Abeceda** - Σ nebo T – konečná množina symbolů
- **Řetězec nad abecedou** – konečná posloupnost symbolů abecedy

- Prázdná posloupnost = **prázdný řetězec** = ϵ

- Σ^* - množina všech řetězců nad Σ :

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

- Σ^+ - množina všech neprázdných řetězců nad Σ

- **Zřetězení**.

- o Připojením řetězce y za řetězec x vznikne řetězec x.y
- o Je asociativní – $(xy)z = x(yz)$
- o Není komutativní
- o ϵ se chová jako neutrální prvek – $\epsilon x = x\epsilon = x$

- **Reverze** x^R

- o $y = abcd, x^R = dcba$

- **Délka řetězce** x : $|x|$

- o $|x| = 0 \Leftrightarrow x = \epsilon$

- **Formální jazyk** L nad Σ : $L \subseteq \Sigma^*$ = množina řetězců

o Operace:

- Množinové operace – sjednocení, průnik, rozdíl
- *Doplňek* (komplement) $L_1 = \overline{L_1} = \Sigma^* \setminus L_1$
- Zřetězení (součin) jazyků
- *n-tá mocnina* jazyka L: $L_n = L \cdot L^{n-1}, L_0 = \{\epsilon\}$
- Iterace L*: $L^* = \bigcup_{n=0}^{\infty} L^n$

$$L^* = L^+ \cup \{\epsilon\}$$

$$L^+ = L \cdot L^* = L^* \cdot L = \bigcup_{n=1}^{\infty} L^n$$

- **Gramatika** – čtverečice $G = (N, \Sigma, P, S)$

- o N – konečná množina **neterminálních symbolů**

- o Σ – konečná množina **terminálních symbolů**

- o P – množina **přepisovacích pravidel**

- o $S \in N$ – **počáteční symbol** gramatiky

o **x přímo derivuje y** ($x \Rightarrow y$), jestliže existuje $(\alpha \rightarrow \beta) \in P$ a $\gamma, \delta \in (N \cup \Sigma)^*$:

- $x = \gamma\alpha\delta, y = \gamma\beta\delta$ ($\gamma\alpha\delta \Rightarrow \gamma\beta\delta$)

o **Derivace řetězce β z řetězce α** s délkou k v gramatice G: $\alpha \Rightarrow^k \beta$:

- $\alpha \Rightarrow^k \beta$, jestliže existuje posloupnost $\alpha_0, \alpha_1, \dots, \alpha_k$ pro $k \geq 0$, $k+1$ řetězců takových, že $\alpha = \alpha_0$, $\alpha_{i-1} \Rightarrow \alpha_i$, pro $1 \leq i \leq k$, a $\alpha_k = \beta$.

o **Tranzitivní uzávěr relace** \Rightarrow^* : $\alpha \Rightarrow^* \beta$, když $\alpha \Rightarrow^i \beta$ pro nějaké $i \geq 1$

- \Rightarrow^* : "derivuje po nenulovém počtu kroků"

o **Tranzitivní a reflexivní uzávěr relace** \Rightarrow : $\alpha \Rightarrow^* \beta$. Když $\alpha \Rightarrow^i \beta$ pro nějaké $i \geq 0$

o **Větná forma** – řetězec α v gramatice G = (N, Σ, P, S) , jestliže když $S \Rightarrow^* \alpha, \alpha \in (N \cup \Sigma)^*$

- Věta generovaná gramatikou G = větná forma, která neobsahuje žádné neterminální symboly

o **Jazyk generovaný gramatikou** G = (N, Σ, P, S) : $L(G) = w : w \in \Sigma^*, \exists S \Rightarrow^* w \{$

= množina všech vět generovaných gramatikou G

o **Ekvivalentní gramatiky** G_1 a G_2 – generují stejný jazyk – $L(G_1) = L(G_2)$

- Chomského klasifikace gramatik

1. **Neomezená** – odpovídá obecné definici gramatiky
2. **Kontextová** – každé pravidlo z P má tvar:
 - a. $\gamma A \delta \rightarrow \gamma \alpha \delta$, kde $\gamma, \delta \in (\Sigma^*)^*$, $\alpha \in (N \cup \Sigma)^+$, $A \in N$
 - b. $S \rightarrow \epsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla
3. **Bezkontextová** – každé pravidlo má tvar
 $A \rightarrow \alpha$, kde $A \in N$, $\alpha \in (N \cup \Sigma)^*$
4. **Regulární** – každé pravidlo má tvar:
 - a. $A \rightarrow aB$ nebo $A \rightarrow a$, kde $A, B \in N$, $a \in \Sigma$
 - b. $S \rightarrow \epsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla

- Klasifikace jazyků

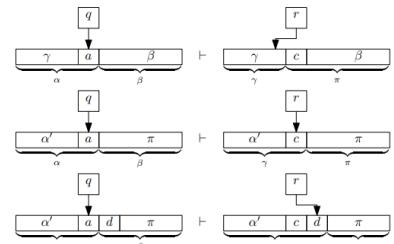
1. **Rekurzivně spočetný** – existuje neomezená gramatika, která ho generuje
 - Rozpoznatelný *Turingovým strojem*
 - např. $L = \text{Halting problem}$ – Turingův stroj R se pro vstup w zastaví
2. **Kontextový** – existuje kontextová gramatika, která ho generuje
 - Rozpoznatelný *lineárně omezeným Turingovým strojem*
 - Lze přijmout *nedeterministickým lineárně omezeným Turingovým strojem*
 - Např. $L = \{a^n b^n c^n, n \in \mathbb{N}\}$
3. **Bezkontextový** – existuje bezkontextová gramatika, která ho generuje
 - Rozpoznatelný *zásobníkovým automatem*
 - Lze přijmout *nedeterministickým zásobníkovým automatem*
 - Např. $L = \{a^n b^n, n \in \mathbb{N}\}$
4. **Regulární** – existuje regulární gramatika, která ho generuje
 - Rozpoznatelný *konečným automatem*
 - Popsatelný *regulárním výrazem*
 - Nejjednodušší množina formálních jazyků
 - Např. $L = \{a^n, n \in \mathbb{N}\}$

Formální jazyky
Rekurzivně spočetné jazyky ± Neomezená gram.
Kontextové jazyky ± Kontextová gram.
Bezkontextové jazyky ± Bezkontextová gram.
Regulární jazyky ± Regulární gram.

- **Uzavřenost bezkontextových jazyků** – uzavřené na operacích sjednocení, součin a iterace
- **Konečné jazyky** – jazyk $L \subset \Sigma^*$ je nazván konečný, pokud $\exists n \in \mathbb{N}: |L| < n$
 - o Nejmenší třída jazyků, která **obsahuje všechny konečné jazyky** a jazyky vzniklé z konečných jazyků pomocí sjednocení, součinu, iterace, doplněk, je množina všech **regulárních jazyků**
- **Deterministický Turingův stroj** – sedmice $R = (Q, \Sigma, G, \delta, q_0, B, F)$
 - o Q – konečná množina vnitřních stavů
 - o Σ – konečná vstupní abeceda
 - o G – konečná pracovní abeceda ($\Sigma \subset G$)
 - o δ – zobrazení z $(Q \setminus F) \times G$ do $Q \times G \times \{-1, 0, 1\}$
 - o $q_0 \in Q$ – počáteční stav
 - o **B – prázdný symbol (Blank, $B \in G \setminus \Sigma$)**
 - o $F \subseteq Q$ – množina koncových stavů
- DTS může zapisovat na vstupní pásku a libovolně se po ní pohybovat
- Turingův stroj se skládá z řídící jednotky, neomezené čtecí pásky rozdělené do buněk a čtecí hlavy.
- Čtecí hlava se umí pohybovat oběma směry, či posečkat na stejném místě.
- **Konfigurace TS R:** $(\alpha, q, \beta) \in G^* \times Q \times G^*$
 - o q je okamžitý vnitřní stav
 - o hlava R na vstupní pásku čte pozici $|\alpha|$
 - o na i-tém políčku vstupní pásky je i-té písmeno řetězce $\alpha\beta$, když $i \leq |\alpha\beta|$, nebo B když $i > |\alpha\beta|$

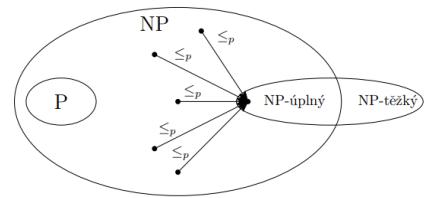
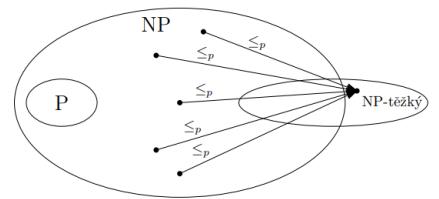
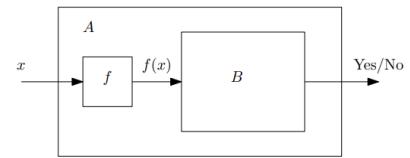
TS přejde $(\alpha, q, \beta) \vdash (\gamma, r, \pi)$, pokud:

- $\alpha = \gamma a, c\beta = \pi, \delta(q, a) = (r, c, -1), a, c \in G$,
- $\alpha = \alpha' a, \gamma = \alpha' c, \beta = \pi, \delta(q, a) = (r, c, 0), a, c \in G$,
- $\alpha = \alpha' a, \gamma = \alpha' cd, \beta = d\pi, \delta(q, a) = (r, c, 1), a, c, d \in G$.



- Turingův stroj $R = (Q, \Sigma, G, \delta, q_0, B, F)$ **přijímá slovo** $a\alpha \in \Sigma^*$, pokud $\exists q \in F, (a, q_0, \alpha) \vdash^* (B, q, \epsilon)$
 - o TS R **přijímá ϵ** , pokud $\exists q \in F, (B, q_0, \epsilon) \vdash^* (B, q, \epsilon)$
 - o $L(R)$ je jazyk slov přijímaných TS R
- Jazyk L je **rekurzivně spočetný**, pokud je přijímán nějakým TS R ($L = L(R)$)
- Každý jazyk přijímaný k-páskovým TS, $k \geq 1$, je rekurzivně spočetný
- **Nedeterministický TS** – sedmice $R = (Q, \Sigma, G, \delta, q_0, B, F)$:
 - o Q – konečná množina vnitřních stavů,
 - o Σ – konečná vstupní abeceda
 - o G – konečná pracovní abeceda ($\Sigma \subset G$)
 - o δ – zobrazení z $(Q \setminus F) \times G$ do $P(Q \times G \times \{-1, 0, 1\})$
 - o $q_0 \in Q$ – počáteční stav
 - o B – prázdný symbol (Blank, $B \in G \setminus \Sigma$)
 - o $F \subseteq Q$ – množina koncových stavů
- Na rozdíl od DTS může NTS mít v daném stavu několik přechodů pro daný symbol. NTS si tedy může vybrat, jakým způsobem bude ve výpočtu pokračovat.
- přijímají právě rekurzivně spočetné jazyky
- NTS **přijímá slovo $a\alpha$** , jestliže existuje $q \in F$ tak, že $(a, q_0, \alpha) \vdash^* (B, q, \epsilon)$
- NTS **přijímá slovo ϵ** , jestliže existuje $q \in F$ tak, že $(B, q_0, \epsilon) \vdash^* (B, q, \epsilon)$
- **Přijetí vstupu**
 - o DTS přijme vstup, pokud přejde do konečného stavu a páška je prázdná (pouze B symboly).
 - o NTS přijme vstupní řetězec, pokud existuje alespoň jedna posloupnost přechodů, kdy TS přejde do koncového stavu a páška je na konci výpočtu prázdná.
 - o vstup není přijat, pokud:
 - TS je v koncovém stavu, ale páška není prázdná
 - TS se zacyklil
 - TS nemůže pokračovat (nedefinovaný přechod)
- Jestliže MN je nedeterministický TS, pak existuje deterministický TS MD takový, že $L(MN) = L(MD)$
 - ⇒ Nedeterministické Turingovy stroje přijímají právě rekurzivně spočetné jazyky
- **Lineárně omezený Turingův stroj** (lineárně omezený automat) – nemůže překročit délku k -násobku vstupního slova pro nějaké $k \geq 1$
- Pro každou gramatiku G existuje TS R takový, že $L(G) = L(R)$. Pro každou nezkracující gramatiku G existuje lineárně omezený TS R takový, že $L(G) = L(R)$
 - ⇒ Gramatiky generují právě rekurzivně spočetné jazyky. Kontextové jazyky jsou přijímány právě lineárně omezenými TS.
- **Rekurzivně spočetné jazyky** jsou **uzavřené** na operacích **sjednocení, součin a iterace**
- **Kontextové jazyky** jsou **uzavřené** na operacích **sjednocení, součin, iterace a doplněk**
- TS R **rozhoduje jazyk** L nad Σ – výpočet se pro každé slovo zastaví a $L(R) = L$
- Jazyk L je **rekurzivní**, pokud **existuje TS, který ho rozhoduje**
- L je **rekurzivní**, právě když L a \bar{L} jsou rekurzivně spočetné
- Každý kontextový jazyk je rekurzivní
- **Church-Turingova teze** – Každý jazyk, který lze nějakým způsobem popsat konečným výrazem, je rekurzivně spočetný. Ke každému algoritmu existuje ekvivalentní Turingův stroj
- **Univerzální TS** – přijímá všechny dvojice (kód (R); α) takové, že TS R přijímá slovo α .
 - o Přijímá jako vstup zakódovaný TS a vstup. Simuluje TS
- **Nerozhodnutelné problémy**:
 - o **Halting problem** – problém zastavení TS – není rekurzivní
 - Zastaví se daný Turingův stroj T pro daný text w? Je to jazyk dvojic (R, w), kde R je TS a $w \in \Sigma^*$ takových, že R se zastaví, má-li na vstupu w.

- Postův korespondenční problém – Mějme dvě posloupnosti $U = (u_1, u_2, \dots, u_m)$ a $V = (v_1, v_2, \dots, v_m)$ řetězců $u_i, v_i \in \Sigma^*$, $|\Sigma| \geq 2$. Zjistěte, zda existuje konečná posloupnost (i_1, i_2, \dots, i_p) , $i_j \in \{1, \dots, m\}$ taková, že $u_{i_1}u_{i_2} \dots u_{i_p} = v_{i_1}v_{i_2} \dots v_{i_p}$
- Třídy P a NP – problémy rozhodovací (ano/ne) a optimalizační (nejlepší řešení)
 - **Polynomiální redukce** – jazyk $A \subseteq \{0, 1\}^*$ je *redukovatelný v polynomiálním čase* na jazyk $B \subseteq \{0, 1\}^*$ ($A \leq_p B$), jestliže existuje funkce $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$, kterou lze spočítat v polynomiálním čase, taková, že pro každé $x \in \{0, 1\}^*$, $x \in A$ právě tehdy, když $f(x) \in B$
 - **P(*polynomial-time*)** – třída problémů, které lze řešit v **polynomiálně omezeném čase** na **deterministickém** Turingově stroji
 - Např. GCD
 - **NP (*non-deterministic polynomial-time*)** – třída problémů, které lze řešit v **polynomiálně omezeném čase** na **nedeterministickém** Turingově stroji
 - Všechny P patří do NP
 - Např. faktorizace přirozených čísel
 - **NP-těžký problém** $B - A \leq_p B$ pro každé $A \in NP$
 - = takové problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP. (Alespoň tak těžké, jako nejtěžší problémy v NP.)
 - Nemusí spadat do třídy NP
 - **NP-úplný problém** $B - B$ je NP-těžký a $B \in NP$
 - = nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP. (NP-úplné = „nejtěžší problémy v NP“)
 - Barvení grafu, kachličkování, Cook's theorem
 - využití v kryptografii
 - Pokud by byl nalezen polynomiálně deterministický algoritmus pro libovolnou NP-úplnou úlohu, všechny NP by byly vyřešeny.



Regulární jazyky: Deterministické a nedeterministické konečné automaty.

Determinizace konečného automatu. Minimalizace deterministického konečného automatu. Operace s konečnými automaty. Regulární gramatiky, regulární výrazy, regulárni rovnice

BI-AAG

- **Regulární jazyky** – existuje regulární gramatika, která ho generuje
 - o Rozpoznatelný konečným automatem
 - o **Konečné jazyky** - jazyk $L \subset \Sigma^*$ je nazván konečný, pokud $\exists n \in N: |L| < n$
 - Nejmenší třída jazyků, která **obsahuje všechny konečné jazyky** a jazyky vzniklé z konečných jazyků pomocí sjednocení, součinu, iterace, doplněk, je množina všech **regulárních jazyků**
 - o **Kleenova věta** – libovolný jazyk je regulární, právě když je přijímaný konečným automatem
- **Deterministický konečný automat M** – pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde:
 - o Q – konečná množina vnitřních stavů
 - o Σ – konečná vstupní abeceda
 - o δ – zobrazení $Q \times \Sigma$ do Q
 - o $q_0 \in Q$ – počáteční stav
 - o $F \subseteq Q$ – množina koncových stavů
- **Konfigurace konečného automatu M** = $(Q, \Sigma, \delta, q_0, F)$
 - o Dvojice $(q, w) \in Q \times \Sigma^*$ - konfigurace konečného automatu M
 - o (q_0, w) – počáteční konfigurace KA M
 - o $(q, \epsilon), q \in F$ – koncová konfigurace KA M
- **Přechod v DKA**
 - o $M = (Q, \Sigma, \delta, q_0, F)$ – deterministický konečný automat
 - o \vdash_M – relace nad $Q \times \Sigma^*$ (podmnožina $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$) taková, že $(q, w) \vdash_M (p, w')$ právě tehdy, když $w = aw'$ a $p = \delta(q, a)$ pro nějaké $a \in \Sigma, w \in \Sigma^*$
 - \vdash_M – přechod v automatu M
 - \vdash_M^k – k-tá mocnina relace \vdash_M
 - $(\alpha_0, \beta_0) \vdash_M (\alpha_k, \beta_k)$ pokud $\exists (\alpha_i, \beta_i), 0 < i < k : (\alpha_i, \beta_i) \vdash_M (\alpha_{i+1}, \beta_{i+1}), 0 \leq i < k$
 - \vdash_M^+ – tranzitivní uzávěr relace \vdash_M
 - \vdash_M^* – tranzitivní a reflexivní uzávěr relace \vdash_M
 - $(q, aw') \vdash_M (p, w')$ znamená $((q, aw'), (p, w')) \in \vdash_M$
- **Jazyk přijímaný DKA:**
 - o **řetězec** $w \in \Sigma^*$ **je přijat** DKA, $M = (Q, \Sigma, \delta, q_0, F)$, jestliže $\exists (q_0, w) \vdash^* M (q, \epsilon)$ pro nějaké $q \in F$
 - o Jazyk přijímaný automatem M - $L(M) = \{w : w \in \Sigma^*, \exists q \in F : (q_0, w) \vdash^* (q, \epsilon)\}$
 - o $w \in L(M)$, jestliže existuje posloupnost přechodů taková, která z počáteční konfigurace (q_0, w) vede do koncové konfigurace (q, ϵ)
- **Úplně určený DKA M** = $(Q, \Sigma, \delta, q_0, F)$ – má zobrazení $\delta(q, a)$ definováno pro všechny dvojice stavů $q \in Q$ a vstupních symbolů $a \in \Sigma$
- **Nedeterministický konečný automat** – pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde:
 - o Q – konečná množina vnitřních stavů
 - o Σ – konečná vstupní abeceda
 - o δ – zobrazení $Q \times \Sigma$ do množiny všech podmnožin Q (2^Q)
 - o $q_0 \in Q$ – počáteční stav
 - o $F \subseteq Q$ – množina koncových stavů
- **Přechod v NKA**
 - o $M = (Q, \Sigma, \delta, q_0, F)$ – nedeterministický konečný automat

- \vdash_M – relace nad $Q \times \Sigma^*$ (podmnožina $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$) taková, že $(q, w) \vdash_M (p, w')$ právě tehdy, když $w = aw'$ a $p \in \delta(q, a)$ pro nějaké $a \in \Sigma$, $w \in \Sigma^*$
 - \vdash_M – přechod v automatu M
 - \vdash_M^k – k -tá mocnina relace \vdash_M
 - \vdash_M^+ – tranzitivní uzávěr relace \vdash_M
 - \vdash_M^* – tranzitivní a reflexivní uzávěr relace \vdash_M
- **Jazyk přijímaný NKA:**
 - řetězec $w \in \Sigma^*$ je přijat NKA, $M = (Q, \Sigma, \delta, q_0, F)$, jestliže $\exists (q_0, w) \vdash^* M (q, \epsilon)$ pro nějaké $q \in F$
 - Jazyk přijímaný automatem M – $L(M) = \{w : w \in \Sigma^*, \exists q \in F : (q_0, w) \vdash^* (q, \epsilon)\}$
- **Dosažitelný stav automatu** $M = (Q, \Sigma, \delta, q_0, F)$
 - stav $q \in Q$ je dosažitelný, pokud existuje řetězec $w \in \Sigma^*$ takový, že existuje posloupnost přechodů, která vede z počátečního stavu q_0 do stavu q : $(q_0, w) \vdash^* (q, \epsilon)$
 - stav, který není dosažitelný, je nedosažitelný
- **Užitečný a zbytečný stav automatu** $M = (Q, \Sigma, \delta, q_0, F)$:
 - stav $q \in Q$ je užitečný, pokud existuje řetězec $w \in \Sigma^*$ takový, že existuje posloupnost přechodů, která vede ze stavu q do nějakého koncového stavu: $\exists p \in F : (q, w) \vdash^* (p, \epsilon)$
- **Konečné automaty s ϵ -přechody**
 - NKA s ϵ -přechody je pětice $M = (Q, \Sigma, \delta, q_0, F)$:
 - Q – konečná množina vnitřních stavů
 - Σ – konečná vstupní abeceda
 - δ – zobrazení $Q \times (\Sigma \cup \{\epsilon\})$ do množiny všech podmnožin Q (2^Q)
 - $q_0 \in Q$ – počáteční stav
 - $F \subseteq Q$ – množina koncových stavů
- **Přechod v NKA s ϵ -přechody** – prvek relace $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$
 - Jestliže $p \in \delta(q, a)$, $a \in \Sigma \cup \{\epsilon\}$, pak $(q, aw) \vdash_M (p, w)$ pro libovolné $w \in \Sigma^*$
- **ϵ -closure** – funkce $Q \rightarrow 2^Q$
 - $\epsilon\text{-Closure}(q) = \{p : (q, \epsilon) \vdash^* (p, \epsilon), p \in Q\}$
- **Konečné automaty s více počátečními stavy**
 - NKA s množinou počátečních stavů I je pětice $M = (Q, \Sigma, \delta, I, F)$:
 - Q – konečná množina vnitřních stavů
 - Σ – konečná vstupní abeceda
 - δ – zobrazení $Q \times \Sigma$ do množiny všech podmnožin Q (2^Q)
 - I – neprázdná podmnožina množiny stavů, $I \subseteq Q$
 - Posloupnost přechodů tohoto KA může začít v libovolném stavu $q \in I$
 - $F \subseteq Q$ – množina koncových stavů
- **Ekvivalentní automaty** M_1 a M_2 – přijímají stejný jazyk, tj. $L(M_1) = L(M_2)$
 - Ke každému NKA M existuje ekvivalentní DKA M' .
- **Množina cílových stavů** $Q(a) \subseteq Q$ automatu $M = (Q, \Sigma, \delta, q_0, F)$ pro libovolné $a \in \Sigma$:
 - $Q(a) = \{q : q \in \delta(p, a), p, q \in Q\}$
- **Homogenní konečný automat** $M = (Q, \Sigma, \delta, q_0, F)$ s množinami cílových stavů $Q(a)$, kde $\forall a \in \Sigma$:
 - Pro všechny dvojice symbolů $a, b \in \Sigma$, $a \neq b$, platí $Q(a) \cap Q(b) = \emptyset$
 - Pro množinu stavů homogenního automatu $M = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů existuje následující rozklad:
$$Q = \biguplus_{a \in \Sigma \cup \{\epsilon\}} Q(a), \quad \text{kde} \quad Q(\epsilon) = \{q_0\} \setminus \bigcup_{a \in \Sigma} Q(a)$$
 - Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je homogenní NKA. Pak počet stavů ekvivalentního DKA $M' = (Q', \Sigma', \delta', q_0', F')$ získaného standardní determinizací (podmnožinovou konstrukcí) je omezen vztahem:
$$|Q'| \leq \sum_{a \in \Sigma} (2^{|Q(a)|}) - |\Sigma| + 1.$$
- **Úplně určený NKA** $M = (Q, \Sigma, \delta, q_0, F)$ – když zobrazení $\delta(q, a) \neq \emptyset$, $\forall q \in Q, a \in \Sigma$.

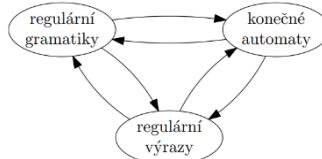
- **Minimální DKA:** $M = (Q, \Sigma, \delta, q_0, F)$ je (stavově) minimální, pokud neexistuje $M' = (Q', \Sigma, \delta', q'_0, F')$ takový, že $L(M) = L(M')$ a $|Q| > |Q'|$.
- **Determinizace KA –** pro každý NKA existuje ekvivalentní DKA
 1. Spojení více počátečních stavů do jednoho (např vstup A a D \Rightarrow nový poč. stav AD)
 2. Vytváření nových stavů, jejichž přechody vzniknou sloučením vstupních symbolů (např. přechod 0 v A \rightarrow B, D \rightarrow C, takže nový stav BC a přechod AD \rightarrow BC)
- **Minimalizace DKA**
 1. (převod na DKA)
 2. Odstranění **nedosažitelných** stavů
 3. Odstranění **zbytečných** stavů
 4. Redukce **ekvivalentních** stavů (prvně rozdelení na koncové a nekoncové stavy, sloučení ekvivalentních stavů a opakovat, dokud se to neustálí)
- **Operace s KA**
 - sjednocení: $L(M) = L(M1) \cup L(M2)$
 - průnik: $L(M) = L(M1) \cap L(M2)$
 - doplněk: vstup musí být úplně určený DKA; prohození koncových stavů: $(L(M')) = \Sigma^* \setminus L(M)$
 - součin (zřetězení): ke koncovému stavu M1 přidáme ϵ -přechod do počátečního stavu M2, $L(M) = L(M1) \cdot L(M2)$
 - iterace: přidá se nový počáteční stav s ϵ -přechodem do původního počátečního stavu (ten už není počáteční), $L = L^*$
- **Regulární gramatika –** každé pravidlo má tvar:
 - $A \rightarrow aB$ nebo $A \rightarrow a$, kde $A, B \in N$, $a \in \Sigma$
 - $S \rightarrow \epsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla
- **Regulární výraz V nad abecedou Σ :**
 1. \emptyset, ϵ, a jsou regulární výrazy pro všechna $a \in \Sigma$.
 2. Jsou-li x, y regulární výrazy nad Σ , pak:
 - a. $x+y$ (sjednocení, alternativa)
 - b. $x.y$ (zřetězení)
 - c. x^* (iterace)
 jsou regulární výrazy nad Σ .
- Hodnota $h(x)$ regulárního výrazu x:
 1. $h(\emptyset) = \emptyset, h(\epsilon) = \{\epsilon\}, h(a) = \{a\}, a \in \Sigma$
 2. $h(x+y) = h(x) \cup h(y),$
 $h(x.y) = h(x).h(y),$
 $h(x^*) = (h(x))^*,$ kde x, y jsou regulární výrazy
- **Identické RV –** $x \equiv y$ – x a y jsou úplně stejné řetězce symbolů
- **Ekvivalentní RV –** $x = y$ – mají stejnou hodnotu ($h(x)=h(y)$) – regulární množiny popisující tyto výrazy jsou stejné
- **Podobné RV –** $x \cong y$ – dají se na sebe převést
- **Soustava regulárních rovnic:**

$X_i = \alpha_{i0} + \alpha_{i1}X_1 + \alpha_{i2}X_2 + \dots + \alpha_{in}X_n, 1 \leq i \leq n$

 - X_1, X_2, \dots, X_n - neznámé
 - α_{ij} - regulární výrazy nad abecedou Σ , která neobsahuje X_1, X_2, \dots, X_n
 - Nechť x, α, β jsou RV. Pak platí:

$$V_1: x = x\alpha + \beta \Rightarrow x = \beta\alpha^* \quad (\text{řešení levé regulární rovnice})$$

$$V_2: x = \alpha x + \beta \Rightarrow x = \alpha^*\beta \quad (\text{řešení pravé regulární rovnice})$$
- Vztahy mezi formálními systémy pro popis RJ:



- Vztah mezi RV a KA - převod **RV -> KA**
 - Metoda **sousedů**
 - Metoda **derivací**
 - Metoda **postupnou konstrukcí**
 - *Každý regulární výraz má pouze konečný počet nepodobných derivací*
 - Důsledek – pro konstrukci DKA pro daný RV pomocí metody derivací stačí uvažovat pouze podobnosti (\cong) výrazů
- Vztah mezi KA a RV - převod **KA -> RV**:
 - Metoda **eliminací stavů**
 - Metoda **regulárních rovnic – příchozí přechody**
 - Metoda **regulárních rovnic – odchozí přechody**
 - *Pro každý NKA M lze sestavit RV V takový, že $L(M) = h(V)$*
 - **Rozšířený konečný automat (RKA) $M = (Q, \Sigma, \gamma, q_0, F)$:**
 - Q – konečná množina vnitřních stavů
 - Σ – konečná vstupní abeceda
 - γ – **zobrazení z $Q \times Q$ do R_T** (R_T je množina všech RV nad Σ)
 - $\gamma(p, q) = \emptyset$, když přechod z p do q není definován
 - $q_0 \in Q$ – počáteční stav
 - $F \subseteq Q$ – množina koncových stavů
 - Jazyk přijímaný RKA M :

$$L(M) = \{x: x \in \Sigma^*, x = x_1x_2\dots x_n, x_i \in \Sigma^* \text{ a existuje posloupnost stavů } q_0, q_1, \dots, q_n, q_n \in F \text{ taková, že } x_1 \in h(\gamma(q_0, q_1)), x_2 \in h(\gamma(q_1, q_2)), \dots, x_n \in h(\gamma(q_{n-1}, q_n))\}$$
 - Nechť $M = (Q, \Sigma, \gamma, q_0, F)$ je RKA. Předpokládejme, že $q \in Q$ není ani počáteční, ani koncový stav. Potom ekvivalentní RKA M' je $(Q \setminus \{q\}, \Sigma, \gamma', q_0, F)$, kde zobrazení γ' je definováno pro každou dvojici $p, r \in (Q \setminus \{q\})$ takto:

$$\gamma'(p, r) = \gamma(p, r) + \gamma(p, q)\gamma(q, q)^*\gamma(q, r)$$
- Vztah mezi RG a RV – převod **RG -> RV**
 - Metoda **eliminací neterminálních symbolů**
 - Metoda **regulárních rovnic**
 - *Pro každou regulární gramatiku $G = (N, \Sigma, P, S)$ lze sestrojit regulární výraz V takový, že $L(G) = h(V)$*
 - **Rozšířená regulární gramatika (RRG) $G = (N, \Sigma, P, S)$:**
 - N – konečná množina neterminálních symbolů
 - Σ – konečná množina terminálních symbolů
 - P – množina pravidel tvaru $A \rightarrow \alpha B$ nebo $A \rightarrow \alpha$, $A, B \in N$, $\alpha \in R_T$
 - $S \in N$ – počáteční symbol
 - Jazyk generovaný RRG:

$$L(G) = \{x: x \in \Sigma^*, x = x_1x_2\dots x_n \text{ a existuje derivace } S \Rightarrow \alpha_1A_1 \Rightarrow \alpha_1\alpha_2A_2 \Rightarrow \dots \Rightarrow \alpha_1\alpha_2\dots\alpha_{n-1}A_{n-1} \Rightarrow \alpha_1\alpha_2\dots\alpha_{n-1}\alpha_n \text{ taková, že } x_i \in h(\alpha_i), 1 \leq i \leq n\}$$
 - Nechť $G = (N, \Sigma, P, S)$ je RRG a A je neterminál, který není počáteční symbol gramatiky G ($A \in N, A \neq S$). RRG $G' = (N \setminus \{A\}, \Sigma, P', S)$, kde pravidla v P' vytvoříme následujícím způsobem, je ekvivalentní s G ($L(G) = L(G')$). Jsou-li v gramatici G pravidla:

$$B \rightarrow \alpha_1C \quad B \rightarrow \alpha_2A \quad A \rightarrow \alpha_3A \quad A \rightarrow \alpha_4C,$$

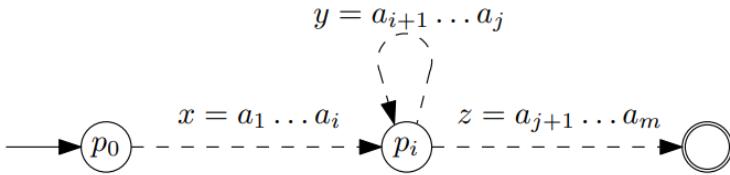
$$\forall B \in N, \forall C \in N \cup \{\epsilon\}, \text{ pak v gramatici } G' \text{ budou v } P' \text{ pravidla:}$$

$$B \rightarrow (\alpha_1 + \alpha_2\alpha_3^*\alpha_4)C$$
 - Pokud se některé z uvedených pravidel v gramatici G nevyskytuje, pak je nahrazeno pravidlem $X \rightarrow \emptyset Y$, kde $X \in \{A, B\}$, $Y \in \{A, C\}$. Odpovídající pravidla nebudou ani v G'
- Vztah mezi RV a RG – převod **RV -> RG**
 - Metoda **postupnou konstrukcí**
 - Metoda **derivací**
 - *Pro každý regulární výraz V lze sestrojit regulární gramatiku G takovou, že $L(G) = h(V)$*

- **Pumping lemma** – nechť L je RJ. Pak pro jazyk L existuje konstanta $p \geq 1$ taková, že pro každou větu $w \in L$ platí:

Jestliže $|w| \geq p$, pak w lze zapsat ve tvaru $w = xyz$ tak, že:

- $y \neq \epsilon$ (t.j. $|y| \geq 1$)
- $|xy| \leq p$,
- $\forall k \geq 0$ platí, že $xy^kz \in L$



- Důkaz, že jazyk $L = \{0^n1^n : n \geq 1\}$ není regulární:

- o Předpokládejme, že L je regulární \rightarrow platí Pumping lemma
- o Věta $w = 0^p1^p \in L$ je zřejmě delší než $p \rightarrow$ musí splňovat požadavky PL.
- o K důkazu, že L není regulární, zkusíme **všechna možná rozdělení věty w na xyz** . Musíme pak dokázat, že **každé z nich porušuje alespoň jednu z podmínek**.
 - A. Rozdělení nesplňuje některou z prvních dvou podmínek
 - B. xy je neprázdná posloupnost nul
 - C. y je neprázdná posloupnost nul
 - D. z obsahuje všechny jedničky.
- o Ale pak xy^0z (odstraníme y z $w = xyz$) nepatří do L ($|0|_{xy^0z} < |1|$)!

$\Rightarrow L$ není regulární jazyk.

- Ekvivalence \sim – reflexivní, symetrická a tranzitivní binární relace
- Třída ekvivalence prvku a na množině X – podmnožina X pbsahující všechny prvky ekvivalentní s a
- Rozklad množiny X podle \sim : X/\sim – množina všech tříd ekvivalence X
- Index ekvivalence \sim - počet tříd ekvivalence v Σ/\sim
 - o Pokud existuje nekonečno tříd ekvivalence, je index ∞
- Pravá kongruence – ekvivalence ∞ na abecedě Σ^* , kde $\forall u, v, w \in \Sigma^*$ platí:

$$u \sim v \Rightarrow uv \sim vw$$

- Prefixová ekvivalence pro L – relace \sim_L na množině Σ^* :

$$u \sim_L v \Leftrightarrow \forall w \in \Sigma^*: uw \in L \Leftrightarrow vw \in L$$

- L – libovolný jazyk nad abecedou Σ (nemusí být RJ)
- Pro každý jazyk L je prefixová ekvivalence pravou kongruencí

- Myhill-Nerodova věta: Nechť L je jazyk nad Σ . Pak následující tvrzení jsou ekvivalentní:

1. L je jazyk přijímaný DKA.
 2. L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.
 3. Relace \sim_L má **konečný index**.
- o 2. varianta M-N věty (minimalita DKA) - Počet stavů libovolného **minimálního DKA** přijímajícího jazyk L je roven indexu \sim_L . (Takový DKA existuje právě tehdy, když je index \sim_L konečný.)

- Důkaz neregularity pomocí Myhill-Nerodovy věty:

$$L = \{0^n1^n : n \geq 1\}$$

- Žádné řetězce $\epsilon, 0, 0^2, 0^3, \dots$ nejsou \sim_L -ekvivalentní, protože $0^i1^i \in L$, ale $0^j1^i \notin L$ pro $j \neq i$
- \sim_L má tedy nekonečně mnoho tříd – **nekonečný index**
- Podle Myhill-Nerodovy věty tedy nemůže být jazyk L přijímán žádným konečným automatem

Bezkontextové jazyky: Bezkontextové gramatiky, zásobníkové automaty a jejich varianty. Modely syntaktické analýzy bezkontextových jazyků

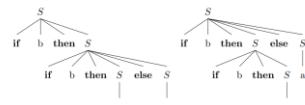
BI-AAG

- **Bezkontextový jazyk** – existuje bezkontextová gramatika, která ho generuje
 - o Přímán nedeterministickým zásobníkovým automate
 - o Rozpoznatelný zásobníkovým automatem
- **Bezkontextová gramatika** – každé pravidlo má tvar
$$A \rightarrow \alpha, \text{ kde } A \in N, \alpha \in (N \cup \Sigma)^*$$
 - o Rozpoznatelný lineárně omezeným Turingovým strojem
 - o Dokáží popsat převážnou většinu syntaktických struktur programovacích jazyků
- **Nejednoznačná BG G** – existuje věta $w \in L(G)$ taková, že je výsledkem alespoň dvou různých derivačních stromů
 - o V opačném případě – jednoznačná
- Mnohdy lze nejednoznačnost odstranit
- Jazyk nelze generovat jednoznačnou gramatikou \rightarrow inherentně (podstatně) nejednoznačný
- Neexistuje algoritmus na rozhodnutí o nejednoznačnosti BG
- Existuje algoritmus, který určí, zda jazyk generovaný danou BG je prázdný
- **Nedostupný symbol** $X \in N \cup \Sigma \in G = (N, \Sigma, P, S)$ - X se neobjeví v žádné větné formě
 - o Neexistuje derivace $S \Rightarrow^* \alpha X \beta, \alpha, \beta \in (N \cup \Sigma)^*$
- **Zbytečný symbol** – neexistuje $S \Rightarrow^* w X y \Rightarrow^* w x y$, kde $w, x, y \in \Sigma^*$
- **Redukovaná BG** – neobsahuje zbytečné symboly
- **Věta o vyloučení pravidla / dosazování:**

$G = (N, \Sigma, P, S) \dots \text{BG}$
 $(A \rightarrow \alpha B \beta) \in P, B \in N, A \neq B, \alpha, \beta \in (N \cup \Sigma)^*$.
 $B \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_k \dots$ všechna pravidla se symbolem B na levé straně v P.
 $G' = (N, \Sigma, P', S)$, kde $P' = P \cup \{A \rightarrow \alpha \gamma_1 \beta \mid \alpha \gamma_2 \beta \mid \dots \mid \alpha \gamma_k \beta\} \setminus \{A \rightarrow \alpha B \beta\}$.
Pak $L(G) = L(G')$.
- **BG bez cyklů** – není v ní možná derivace $A \Rightarrow^+ A$ pro žádné $A \in N$
- **BG $G = (N, \Sigma, P, S)$ bez ϵ -pravidel:**
 - a. P neobsahuje ϵ -pravidla
 - b. P obsahuje pouze pravidlo $S \rightarrow \epsilon$ a S se nevyskytuje na pravé straně žádného pravidla v P.
- **Derivační strom** – grafické vyjádření syntaktické struktury větné formy – strom s následujícími vlastnostmi:
 1. Uzly DS jsou ohodnoceny terminálními a neterminálními symboly a symbolem ϵ
 2. Kořen stromu je ohodnocen počátečním symbolem S
 3. Jestliže uzel má alespoň 1 následovníka, je ohodnocen neterminálem
 4. Jestliže $n_1 \dots n_k$ jsou bezprostřední následovníci uzlu n , který je ohodnocen s. A, a tyto uzly jsou zleva doprava ohodnoceny s. $A_1, \dots A_k$, pak $A \rightarrow A_1 A_2 \dots A_k$ je pravidlo v P
 5. Koncové uzly derivačního stromu tvoří zleva doprava větnou formu nebo větu v gramatice G, která je výsledkem derivačního stromu
- **Chomského normální tvar:**
 - o Každé pravidlo má jeden z tvarů:
 1. $A \rightarrow BC$, kde $A, B, C \in N$.
 2. $A \rightarrow a$, kde $a \in \Sigma, A \in N$.
 3. $S \rightarrow \epsilon$, pokud $\epsilon \in L(G)$ a S se nevyskytuje na pravé straně žádného pravidla.
- **Vlastní BG** – je bez cyklů, ϵ -pravidel a zbytečných symbolů
- Každý BK jazyk L je generovaný gramatikou v CHNK.

Příklad
 $G = (\{S\}, \{a, b, \text{if}, \text{then}, \text{else}\}, P, S)$, kde P :
 $S \rightarrow \text{if } b \text{ then } S \text{ else } S$
 $S \rightarrow \text{if } b \text{ then } S$
 $S \rightarrow a$

Gramatika je nejednoznačná: if b then if b then a else a



- **Rekurzivní neterminál A v BG G** – existuje derivace $A \Rightarrow^+ \alpha A \beta$ pro nějaké α a $\beta \in (N \cup \Sigma)^*$
 - o $\alpha = \epsilon \rightarrow A$ = symbol *rekurzivní zleva*
 - o $\beta = \epsilon \rightarrow A$ = symbol *rekurzivní zprava*
- **Rekurzivní gramatika** – obsahuje alespoň jeden rekurzivní neterminální symbol
- Každý BG může být generován gramatikou, která neobsahuje rekurzivitu zleva
- **Zásobníkový automat** = sedmice $R = (Q, \Sigma, G, \delta, q_0, Z_0, F)$, kde:
 - Q – konečná množina vnitřních stavů
 - Σ – konečná vstupní abeceda
 - G – konečná abeceda zásobníku
 - δ – zobrazení z konečné podmnožiny $Q \times (\Sigma \cup \{\epsilon\}) \times G^*$ do množiny konečných podmnožin $Q \times G^*$
 - $q_0 \in Q$ – počáteční stav
 - $Z_0 \in G$ – počáteční symbol v zásobníku
 - $F \subseteq Q$ – množina koncových stavů
- **Konfigurace ZA:**

$$(q, w, \alpha) \in Q \times \Sigma^* \times G^*$$
 - q – okamžitý vnitřní stav
 - w – dosud nezpracovaná část vstupního řetězce
 - α – obsah zásobníku
- Počáteční konfigurace ZA: (q_0, w, Z_0) , $w \in \Sigma^*$
- $\delta(q, a, \alpha) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$
 - ve stavu q přečte symbol a
 - přejde do stavu p_i , $i \in \{1, 2, \dots, m\}$
 - řetězec α na vrcholu zásobníku nahradí řetězcem γ_i
- $\delta(q, \epsilon, \alpha) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$
 - přechod do nového stavu
 - změnu obsahu zásobníku bez čtení vstupního symbolu
- **Přechod ZA** $R = (Q, \Sigma, G, \delta, q_0, Z_0, F)$:
 - o \vdash_R – relace nad $Q \times \Sigma^* \times G^*$ (podmnožina $(Q \times \Sigma^* \times G^*) \times (Q \times \Sigma^* \times G^*)$), taková, že
 - $(q, w, \alpha\beta) \vdash_R (p, w', \gamma\beta)$ právě tehdy, když $w = aw'$ a $(p, \gamma) \in \delta(q, a, \alpha)$ pro nějaké $a \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$, $\alpha, \beta, \gamma \in G^*$.
 - **Prvek relace \vdash_R** = přechod zásobníkového automatu R .
 - \vdash^k - k-tá mocnina relace \vdash
 - \vdash^+ - tranzitivní uzávěr relace \vdash
 - \vdash^* - tranzitivní a reflexivní uzávěr relace \vdash
- **Jazyk definovaný/přijímaný ZA** $R = (Q, \Sigma, G, \delta, q_0, Z_0, F)$:
 1. **Přechodem do koncového stavu**
 $L(R) = \{w : w \in \Sigma^*, \exists \gamma \in G^*, \exists q \in F, (q_0, w, Z_0) \vdash^* (q, \epsilon, \gamma)\}$
 2. **S prázdným zásobníkem**
 $L_\epsilon(R) = \{w : w \in \Sigma^*, \exists q \in Q, (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$
- Nechť $P = (Q, \Sigma, G, \delta, q_0, Z_0, F)$ je ZA. Pokud $(q, w, A) \vdash_P^n (q', \epsilon, \epsilon)$, pak $(q, w, A\alpha) \vdash_P^n (q', \epsilon, \alpha)$, $\forall A \in G$, $\forall \alpha \in G^*$
- Nechť L je jazyk. Pak \exists ZA P_ϵ přijímající L s prázdným zásobníkem právě tehdy, když \exists ZA P_f přijímající L přechodem do koncového stavu.
- Ze stavu Q , na vstup Σ nebo ϵ čtu ze zásobníku, jdu do stavu a přidávám do zásobníku (tzn. mám nějaký zásobník, kde si můžu něco ukládat)
- Podle hodnoty na zásobníku se mohou definovat přechody v daném stavu (0, in | out)
- V počáteční konfiguraci je zásobník prázdný $((q_0, w, Z_0), w \in \Sigma^*)$
- Koncový stav se liší dle typu
- Deterministický/nedeterministický

- **Levá/pravá derivace** $S \Rightarrow^* \alpha, \alpha \in (N \cup \Sigma)^*$ - při každém kroku byl nahrazován neterminální symbol nejvíce vlevo/vpravo ve větné formě
- **Rozklad** – lineární reprezentace derivačního stromu
 - o Derivačnímu stromu odpovídá jediná levá/pravá derivace a určité levé/pravé derivaci odpovídá jediný derivační strom
- **Levý/pravý rozklad větné formy** α v $G = (N, \Sigma, P, S)$:
 - = posloupnost čísel pravidel použitých v derivaci $S \Rightarrow^* \alpha$
 - pravidla v P jsou očíslována 1, 2, ..., $|P|$
 - *levý rozklad* – posloupnost čísel pravidel použitých v levé derivaci $S \Rightarrow^* \alpha$
 - *pravý rozklad* – obrácená posloupnost čísel pravidel použitých v pravé derivaci $S \Rightarrow^* \alpha$
- **Syntaktická analýza** (rozklad) – konstrukce derivačního stromu $G = (\{S, A\}, \{a, b\}, \{S \xrightarrow{1} AS, S \xrightarrow{2} b, A \xrightarrow{3} a\}, S)$
 - o Metoda **shora dolů** – proces nalezení levého rozkladu dané věty v dané gramatice
 - o Metoda **zdola nahoru** – proces nalezení pravého rozkladu dané věty v dané gramatice
- Je-li dána BG $G = (N, \Sigma, P, S)$, můžeme sestrojit ZA R takový, že $L(G) = L_e(R)$
- Konstrukce ZA – metoda shora dolů:

$R = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \emptyset)$, kde δ :

 1. **Expanze** – $\delta(q, \epsilon, A) \leftarrow \{(q, \alpha) : (A \rightarrow \alpha) \in P\}, \forall A \in N$
 - expanze neterminálu v zásobníku dle přechodu
 2. **Srovnání** – $\delta(q, a, a) \leftarrow \{(q, \epsilon)\}, \forall a \in \Sigma$
 - na vrcholu zásobníku je stejná sekvence terminálů, jako má slovo (začínají stejně), takže můžeme srovnat, že odebereme stejné terminály
 - o **Vrchol zásobníku** u tohoto typu automatu bude vždy **vlevo** – levý rozklad
 - o **Pro nalezení**
- Je-li dána BG $G = (N, \Sigma, P, S)$, můžeme sestrojit ZA R takový, že $L(G) = L(R)$
- Konstrukce ZA – metoda zdola nahoru
 1. **Přesun** – $\delta(q, a, \epsilon) \leftarrow \{(q, a)\}, \forall a, a \in \Sigma$
 - přesouvá jeden nejlevější terminál na zásobník
 2. **Redukce** – $\delta(q, \epsilon, \alpha) \leftarrow \{(q, A) : (A \rightarrow \alpha) \in P\}$
 - zásobník má posloupnost terminálů, které můžeme zredukovat dle pravidel
 3. **Přijetí** – $\delta(q, \epsilon, \#S) \leftarrow \{(r, \epsilon)\}$
 - o **Vrchol zásobníku** bude vždy **vpravo**
 - o Slovo přijato, až na zásobníku zůstane $|S|$
- **Deterministický zásobníkový automat** $P = (Q, \Sigma, G, \delta, q_0, Z_0, F)$:
 1. $|\delta(q, a, \gamma)| \leq 1, \forall q \in Q, \forall a \in (\Sigma \cup \{\epsilon\}), \forall \gamma \in G^*$
 2. Pokud $\delta(q, a, \alpha) \neq \emptyset, \delta(q, a, \beta) \neq \emptyset$ a $\alpha \neq \beta$, pak α není předponou β a β není předponou α (tzn. $\alpha\gamma \neq \beta, \alpha \neq \beta\gamma, \gamma \in G^*$)
 3. Pokud $\delta(q, a, \alpha) \neq \emptyset, \delta(q, \epsilon, \beta) \neq \emptyset$, pak α není předponou β a β není předponou α (tzn. $\alpha\gamma \neq \beta, \alpha \neq \beta\gamma, \gamma \in G^*$)
- Konstrukce DZA – shora dolů:
 - o Vstup = $G = (N, \Sigma, P, S)$, kde všechna pravidla jsou ve tvaru $A \rightarrow a\alpha, a \in \Sigma, \alpha \in (N \cup \Sigma)^*$ a pro každá dvě různá pravidla $\{A \rightarrow a\alpha, A \rightarrow b\beta\} \subset P$ platí $a \neq b$

$R \leftarrow (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \emptyset)$

 - $\delta(q, a, A) \leftarrow \{(q, \alpha) : (A \rightarrow a\alpha) \in P\}, \forall A \in N$
 - $\delta(q, a, a) \leftarrow \{(q, \epsilon)\}, \forall a \in \Sigma$

Základní pojmy teorie grafů. Grafové algoritmy: procházení grafu do šířky a do hloubky, určení souvislých komponent, topologické uspořádání, vzdálenosti v grafech, konstrukce minimální kostry a nejkratších cest v ohodnoceném grafu

BI-AG1

- **Neorientovaný graf** = uspořádaná dvojice $G = (V, E)$, kde V je nepr. konečná množina uzlů a E je množina hran

 - o Hrana = dvouprvková podmnožina V

 - o Množina všech možných hran: $\binom{V}{2}$

 - o **Úplný graf** na n vrcholech K_n = graf $(V, \binom{V}{2})$, kde $|V| = n$

 - o **Úplný bipartitní graf** K_{n_1, n_2} tvořený dvěma partitami o n_1 a n_2

vrcholek je graf $(A \cup B, \{(a, b) \mid a \in A, b \in B\})$, kde $A \cap B = \emptyset$,
 $|A| = n_1, |B| = n_2$

 - o **Cesta** délky m (s m hranami) P_m je graf $(\{0, \dots, m\}, \{(i, i+1) \mid i \in \{0, \dots, m-1\}\})$

 - o **Kružnice** délky n (s n vrcholy, $n \geq 3$) C_n je graf $(\{1, \dots, n\}, \{(i, i+1) \mid i \in \{1, \dots, n-1\} \cup \{1, n\}\})$

 - o Nechť $e = \{u, v\}$ je hrana v grafu G . Pak řekneme, že

 - Vrcholy u a v jsou **koncové vrcholy** hrany e
 - U je **sousedem** v G (a naopak)
 - U je **incidentní** s hranou e (i v)

- **Orientovaný graf** – uspořádaná dvojice $G = (V, E)$, kde V je neprázdná konečná množina uzlů a E je množina orientovaných hran

 - o **Orientovaných hrana** je dvouprvková podmnožina $V \{u, v\}$ - u je předchůdce, v následník

 - o **Zdroj** – vrchol orientovaného grafu, do kterého nevede žádná hrana (každý orientovaný acyklický graf má alespoň jeden zdroj)

 - o **Topologické uspořádání** – pořadí modulů m_1, m_2, \dots, m_n takové, že všechny orientované hrany vedou pouze od nižším k vyšším indexům

- **Doplňek grafu** \bar{G} grafu $G = (V, E)$ je graf $(V, \binom{V}{2} \setminus E)$

- **Izomorfismus** – Nechť G a H jsou dva grafy. Funkce $f: V(G) \rightarrow V(H)$ je izomorfismus grafů G a H , pokud:

 - o f je bijekce

 - o Pro každou dvojici vrcholů $u, v \in V(G)$: $\{u, v\} \in E(G) \Leftrightarrow \{f(u), f(v)\} \in E(H)$

- **Automorfismus** – graf je izomorfní sám se sebou – permutace vrcholů zachovávající „býtí hranou“ – „symetrie“ grafu

- Počty grafů – na n -prvkové množině vrcholů V je právě $2^{\binom{n}{2}}$ různých grafů

- Nechť $G = (V, E)$ je graf a $v \in V$ jeho vrchol.

 - o $\deg_G(v)$ = počet hran grafu G obsahujících vrchol v = **stupeň vrcholu** v v grafu G

 - o $N_G(v)$ = množina všech sousedů vrcholu v v grafu G = (otevřené) **okolí vrcholu** v v grafu G

 - o $N_G[v] = N_G(v) \cup \{v\}$ = uzavřené okolí vrcholu v v grafu G

- **R-regulární graf** – stupeň každého vrcholu je r

- **Regulární graf** – je r -regulární pro nějaké r

- **Izolovaný vrchol** – vrchol stupně 0

- **Princip sudosti** – Pro každý graf $G = (V, E)$ platí $\sum_{v \in V} \deg_G(v) = 2|E|$

 - o V každém grafu je počet vrcholů lichého stupně sudý

 - o Každý regulární graf lichého stupně musí mít sudý počet vrcholů

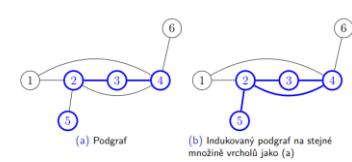
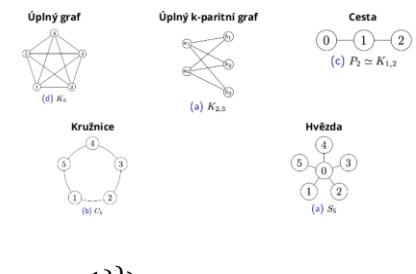
- **Podgraf** – Graf H je podgrafem grafu G ($H \subseteq G$), když $V(H) \subseteq V(G) \wedge E(H) \subseteq E(G)$

 - o Indukovaný podgraf ($H \leq G$) – $V(H) \subseteq V(G) \wedge E(H) = E(G) \cap \binom{V(H)}{2}$

 - o Klika – podmnožina vrcholů, z nichž každé dva jsou sousední

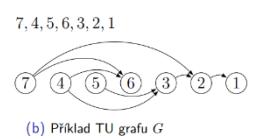
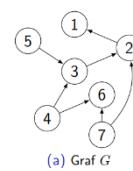
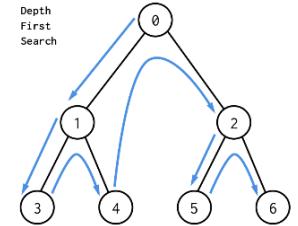
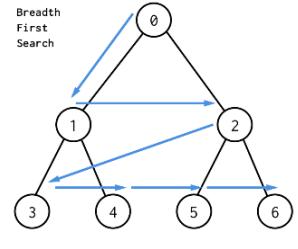
 - o Nezávislá množina – podmnožina vrcholů, z nichž žádné dva nejsou sousední

- **Cesta v grafu G** – podgraf grafu G izomorfní s nějakou cestou P



- Délka cesty je počet hran v cestě
 - Má-li cesta P v grafu G koncové vrcholy u a v , mluvíme do cestě z u do v , nebo o u - v -cestě
 - Cesta = posl. navzájem různých vrcholů v_0, \dots, v_k takových, že $\forall i \in \{0, \dots, k-1\}: \{v_i, v_{i+1}\} \in E(G)$
 - Kružnice (cyklus) v grafu G = podgraf grafu G izomorfní s nějakou kružnicí C
- **Souvislost grafu** – graf G je souvislý, jestliže v něm pro každé dva vrcholy u , v existuje u - v -cesta (jinak je graf nesouvislý)
 - **Vzdálenost mezi body u, v** – délka nejkratší cesty mezi těmito body (počet hran v cestě).
 - **Souvislá komponenta** – indukovaný podgraf H grafu G je souvislou komponentou, pokud je souvislý a neexistuje žádný souvislý podgraf F , $F \neq H$ grafu G takový, že $H \subseteq F$
 - Graf je souvislý právě tehdy, když obsahuje jednu komponentu
 - **Strom** – souvislý graf neobsahující kružnici
 - Graf G nazveme **lesem**, pokud neobsahuje žádnou kružnici
 - **List** – vrchol ve stromu se stupněm = 1
 - Každý strom T s aspoň 2 vrcholy obsahuje alespoň 2 listy
 - **Věta o trhání listů** – nechť $G = (V, E)$ je graf na alespoň 2 vrcholech a nechť $v \in V$ je jeho list. Pak jsou následující tvrzení ekvivalentní:
 - G je strom
 - $G - v$ je strom

→ pokud ze stromu list odebereme nebo do stromu list přidáme, zůstává stromem
 - **Kostra** – podgraf K souvislého grafu G nazveme kostrou grafu G , pokud $V(K) = V(G)$ a K je strom.
 - **Acyklický (orientovaný) graf** – graf neobsahující (orientovanou) kružnici.
 - **BFS** – hledání do šířky – u neorientovaného grafu vezmu vrchol a zařadím do pole FIFO. Vezmu všechny nezkontrolované sousedy a zařadím do pole. Označím původní vrchol za zkontrolovaný. Pokračuji se všemi vrcholy v poli. Výsledkem je procházení grafu po stupni hloubky.
 - Časová složitost – $O(|V| + |E|)$
 - Paměťová složitost – $O(|V| + |E|)$
 - **DFS** – hledání do hloubky – u neorientovaného grafu vezmu vrchol. Pokud nemá sousedy (kromě rodiče), označ ho jako zkontrolovaný. Jinak pro každého souseda zavolej DFS, tzn. od kořene jdu co nejhouběji to jde. Vracím se vždy po návštěvě listu, před zanořením značkuji. V případě možnosti opět jdu do hloubky.
 - Časová složitost – $O(|V| + |E|)$
 - Paměťová složitost – $O(|V| + |E|)$
 - **Topologické uspořádání** orientovaného acyklického grafu $G = (V, E)$ je takové pořadí vrcholů v_1, \dots, v_n grafu G , že pro každou hranu $(v_i, v_j) \in E: i < j$
 - **TopSort algoritmus** – cyklicky kontroluji zdroje (uzly bez vstupních hran), přidávám je do výstupu a odstraňuji všechny hrany, z něž vychází (vytvářím případné nové zdroje). V případě cyklické závislosti neexistuje řešení.
 - Výstupem je TU vstupního grafu, pokud existuje
 - Časová a paměťová složitost – $O(|V| + |E|)$
 - Algoritmy vhodné pro:
 - **Vzdálenosti v grafech** – neohodnocený graf BFS, ohodnocený Dijkstra
 - **Konstrukce minimální kostry** – Kruskal a Jarník
 - **Nejkratší cesta v grafu** – Dijkstra a Bellman-Ford (NP-úplná úloha)
 - **Hranově ohodnocený graf** – Takový souvislý neorientovaný graf $G = (V, E)$, jehož každé hraně e přiřadíme váhu $w(e)$ z reálných čísel.
 - Váhovou funkci můžeme rozšířit na podgrafy – váha $w(H)$ podgrafu H je součet vah jeho hran
 - Kostra hranově ohod. grafu G je **minimální**, pokud má mezi všemi jeho kostrami nejmenší váhu

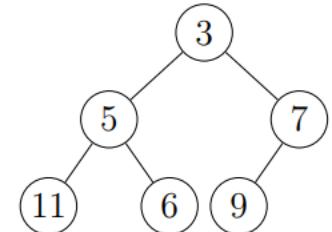


- **Nejkratší uv-cesta** – libovolná uv-cesta, jejíž délka je rovná vzdálenosti $d(u, v)$
 - o Pro lib. 2 body u a v ohodnoceného orientovaného grafu, vzdálenost $d(u, v)$ je minimum z délek všech uv-cest (cest z u do v), případně nekonečno, pokud žádná uv-cesta neexistuje
- **Sled** v orientovaném grafu G – posloupnost $(v_0, e_1, v_1, e_2, \dots, e_n, v_n)$ taková, že:
 - $v_i \in V \forall i \in \{0, \dots, n\}$
 - $e_i = (v_{i-1}, v_i) \in E(G) \forall i \in \{1, \dots, n\}$
 - o Délka $l(S)$ sledu S je součet ohodnocení hran, které na něm leží
- **Jarníkův algoritmus** – hladový algoritmus pro nalezení minimální kostry
 - o Počátek – strom s jedním vrcholem
 - o Nalezneme nejlehčí hranu incidentní se stromem a přidáme do stromu
 - o Opakujeme, dokud nevznikne celá kostra
 - o Časová složitost – $O(nm)$
 - o Paměťová složitost – $O(n + m)$, kde $n = |V|$ a $m = |E|$
- **Kruskalův algoritmus** – hladový algoritmus pro nalezení minimální kostry
 - o Opakujeme:
 - Vyberu nejlehčí hranu celého grafu
 - Pokud nevytvoří cyklus, přidám do kostry
 - o Časová složitost – $O(m \log n + n^2)$, pokud má keřkovou strukturu, tak v $O(m \log n)$
- **Dijkstrův algoritmus** – pro nalezení nejkratší cesty v ohodnoceném grafu
 - o Předpokládá celočíselné kladné hrany
 - o U všech vrcholů označím nejkratší vzdálenost $+inf$ z neznámého předchůdce
 - o Vyberu počáteční vrchol u a všem sousedům zapíšu délku cesty a předchůdce označím jako hotový
 - o Vezmu vrchol s nejmenší vzdáleností z nedokončených a sousedům upravím délku cesty a otevřu je (pokud tím snížím délku cesty souseda, jinak je nechám) - princip **relaxace vrcholů**
 - o Opakuju, dokud jsou vrcholy otevřené
 - o Časová složitost: $O((n + m) \cdot \log n)$.
- **Bellman-Fordův algoritmus** – pro nalezení nejkratší cesty v ohodnoceném grafu
 - o Předpokládá neexistenci záporných cyklů
 - o Princip jako Dijkstra, ale nepoužívá se prioritní fronta pro výběr uzel s nejkratší délkou, ale klasická fronta
 - o Časová složitost – $O(nm)$

Binární haldy, binomiální haldy. Vyhledávací stromy a jejich vyvažování. Tabulky s rozptylováním (hešováním)

BI-AG1

- **Zakořeněný strom** – jeden vrchol je označen jako **kořen**
- Leží-li u na (jediné) cestě z v do kořene, pak u je **předek** v a v je **potomek** u. Pokud je navíc $\{u, v\}$ z $E(T)$ hrana, u je **otec** v a v je **syn** u
- Vrcholy rozdělíme podle vzdálenosti od kořene do **hladin** – v 0. leží kořen
- **Hloubka vrcholu** = vzdálenost od kořene = číslo hladiny
- **Binární minimová halda**
 - o Datová struktura tvaru binárního stromu, v jehož každém vrcholu je uložen 1 prvek. Přitom platí:
 - a. **Tvar haldy**: Všechny hladiny kromě poslední jsou plně obsazeny, poslední hladina je zaplněna zleva.
 - b. **Haldové uspořádání**: Je-li v vrchol a s jeho syn, pro jejich klíče platí $k(v) \leq k(s)$.

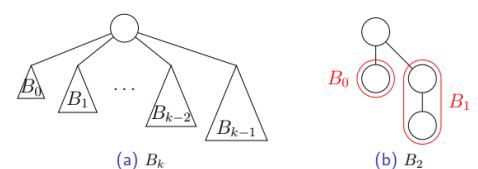


- o Pamatuje si množinu prvků opatřených klíči
- o Halda s n prvky má $\lfloor \log_2 n \rfloor + 1$ hladin, $\left\lceil \frac{n}{2} \right\rceil$ vnitřních vrcholů a $\left\lceil \frac{n}{2} \right\rceil$ listů
- o Reprezentace pomocí pole: otec $arr[n]$ má syny $arr[2n+1]$ a $arr[2n+2]$
- o Analogicky i maximová halda
- o Operace:
 - **INSERT(x)** – vloží prvek x do množiny
 - Nový list na konec poslední hladiny, pokud je plná, založení nové
 - Pokud je haldové uspořádání mezi novým listem l a jeho otcem o porušeno, prohodíme $k(l)$ a $k(o)$ – opakujeme, dokud uspořádání není splněno – $O(\log n)$ – **BUBBLE_UP(i)**
 - **MIN(x)** – najde prvek s nejmenším klíčem (nebo libovolný ze stejných nejmenších)
 - Kořen haldy
 - **EXTRACT_MIN(x)** – odeberete prvek s nejmenším klíčem a vrátí ho jako výsledek
 - Kořen smazat nemůžeme, ale poslední list ano – prohození a smazání posledního listu
 - Poškodí uspořádání – **BUBBLE_DOWN(i)** – prohazování prvku s menším z obou synů, dokud nebude zachováno uspořádání – $O(\log n)$
- o Nejrychlejší vytvoření haldy – vychází z toho, že listy jsou kořeny již seřazených podhal, takže se může řadit pouze horní polovina (vnitřní uzly) – $O(n)$

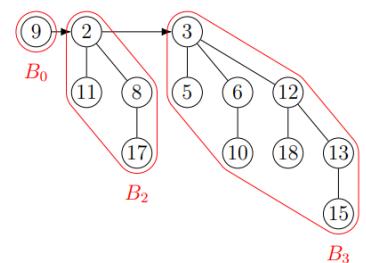
Binomiální halda

- o **Binomiální strom řádu k** – zakořeněný strom T, který splňuje:
 - a. Pokud je řád k roven 0, pak T obsahuje pouze kořen.
 - b. Pokud je řád k nenulový, pak T má kořen s právě k syny. Tito synové jsou kořeny podstromů, které jsou po řadě binomiálními stromy řádů 0, ..., k-1.
- o B_k je strom tvořen dvěma B_{k-1} stromy, přičemž kořen jednoho je navázán na kořen druhého
- o **Binomiální halda** pro danou množinu prvků se skládá ze souboru binomiálních stromů $T = T_1 \dots T_l$, kde:
 1. Každý strom T_i je binomiální strom.
 2. Uchovávané prvky jsou uloženy ve vrcholech stromů T_i . Klíč uložený ve vrcholu v z $V(T_i)$ je $k(v)$
 3. Pro každý strom T_i platí haldové uspořádání klíčů neboli pro každý vrchol v z $V(T_i)$ a libovolného jeho syna s je $k(v) \leq k(s)$.

```
HeapBuild
Vstup: prvky  $x_1, \dots, x_n$ 
Výstup: Halda  $H[1 \dots n]$ 
(1) vlož prvky do pole  $H$  tak, že  $H[i] := x_i$ 
(2) Pro  $i := \lfloor n/2 \rfloor, \dots, 1$ :
(3)     BubbleDown( $H[i]$ )
```



4. V souboru T se žádný řád stromu nevyskytuje více než jednou.
 5. Soubor stromů T je uspořádán vzestupně podle řádu stromu.
 6. Počet vrcholů stromu je součtem počtu vrcholů stromů souboru T.
- Opět počítáme s minimovou haldou
 - Binomiální strom B_i se vyskytuje v seznamu T n-prvkové BH právě tehdy, když ve dvojkovém zápisu b_k, b_{k-1}, \dots, b_0 čísla n je $b_i = 1$.
 - Binomiální strom s n vrcholy má **hloubku $O(\log n)$** a **počet synů kořene** je taktéž **$O(\log n)$** .
 - Binomiální strom T řádu k má **2^k vrcholů**, které jsou rozděleny do **$k + 1$ hladin**. Kořen stromu má právě k synů.
 - **Nalezení minima** – musí být v kořenech, stačí je projít $\rightarrow O(\log n)$
 - **Spojení binomiálních stromů** – pouze stromy stejného řádu
 - Jeden se zavěší pod druhý – připojení kořene druhého jako posledního syna pod kořenem prvního (větší klíč za menší)



- **Slévání binomiálních hald – Merge** – postupné přesouvání jednotlivých stromů nejnižších řádů do nové prázdné haldy – $O(\log n)$
- **Vkládání prvků** do binomiální haldy – vytvoříme novou binomiální haldu obsahující pouze vkládaný prvek a následně zavoláme Merge – $O(\log n)$
- **Odstranění minima** – průchodem stromů nalezneme binomiální strom M, jehož kořen je minimem haldy, tento strom z H odpojíme. Následně ze stromu M odtrhneme kořen a všechny jeho syny včetně jejich podstromů) vložíme do nové binomiální haldy H'. Poté pomocí Merge slijeme H a H'.

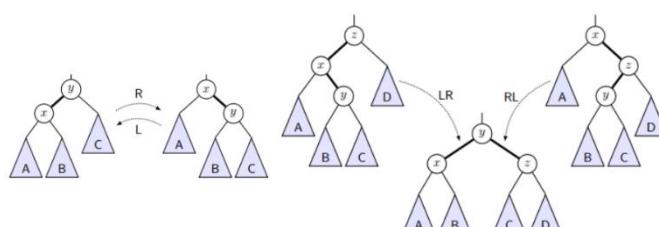
- Vyhledávací stromy a vyvažování

○ Binární vyhledávací strom – BVS

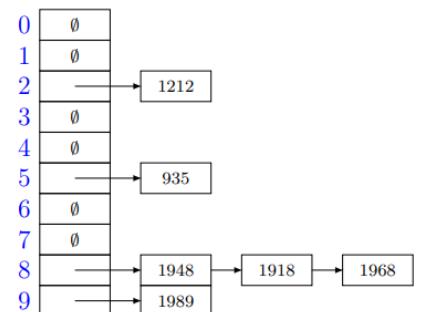
- Binární strom, v jehož každém vrcholu v je uložen unikátní klíč $k(v)$. Přitom \forall vrchol v platí:
 - Pokud a patří do $L(v)$ – levého podstromu, pak $k(a) < k(v)$.
 - Pokud b patří do $R(v)$ – pravého podstromu, pak $k(b) > k(v)$.
- Složitost je průměrně $O(\log n)$, při špatném vyvážení až $O(n)$
- BVS nazveme **dokonale vyvážený**, pokud pro každý jeho vrchol u platí: $| |L(u)| - |R(u)| | \leq 1$
= počet vrcholů levého a pravého podstromu se smí lišit nejvýše o 1
- **Vložení**: na vhodné místo, aby splňoval podmínky
- **Odstranění**:
 - 1 syn = přepojení svého syna na svého otce
 - 2 synové = nalezení svého předchůdce (nejpravější klíč levého podstromu)

○ AVL – hloubkově vyvážený BVS

- Pro každý vrchol v platí $|h(l(v)) - h(r(v))| \leq 1$
= Hloubka levého a pravého podstromu se vždy liší nejvýše o 1
- AVL strom na n vrcholech má hloubku $\Theta(\log n)$
- V každém vrcholu v je číslo $\delta(v) = h(r(v)) - h(l(v))$, které nazveme znaménko vrcholu:
 $\delta(v) = +1$ (pravý podstrom hlubší) – takový vrchol značíme \oplus ,
 $\delta(v) = -1$ (levý podstrom hlubší) – značíme \ominus ,
 $\delta(v) = 0$ (oba podstromy stejně hluboké) – značíme \circ .
- **Rotace AVL** – vyvažování – „otočení“ hrany mezi dvěma vrcholy a přepojení jejich podstromů tak, aby byli i nadále synové vzhledem k otcům správně uspořádáni – jednoduchá x dvojitá rotace



- Tabulky s rozptylováním (hešováním)
 - Cílem hešování je skloubit nízké paměťové nároky operací a přitom zachovat konstantní složitost operací, i když pouze v průměrném případě.
 - Pro univerzum klíčů volíme vhodnou *hash funkci* $h : U \rightarrow H$, která **každému klíči přiřadí hash hodnotu** (například modulení), dle které se bude vyhledávat. Snažíme se, aby bylo co nejméně kolizí.
 - **Hešování s řetízky**
 - Kolize jsou řešeny pomocí řetízků
 - Prvky jsou ukládány do pole či spojového seznamu odpovídajícího příslušnému hashi
 - Pokud množina tříděných klíčů je mnohem větší než počet unikátních hashů, tak se snažíme, aby všechny hashe měli přibližně stejně vzorů, aby nevznikaly dlouhé spojové seznamy s neefektivním $O(n)$ vyhledáváním.
 - **Hešování s otevřenou adresací**
 - Tentokrát se do každé příhrádky vejde pouze jeden prvek
 - Pro prvek neexistuje jen jedna hash jako v předchozím případě, ale posloupnost hashů (takže když nebude v prvním, může být v druhém atd.)
 - Při pokusu o uložení do obsazené příhrádky budeme postupně zkoušet nahradní příhrádky, dokud nenajdeme volnou
 - V případě mazání nahradíme prvek značkou (*tombstonem*) a v případě vysokého počtu tombstonů přepočítáme celou tabulku
 - Možnost dvojitého hashování, kde se počítá s počty neúspěšných pokusů



Tabulka:	0 1 2 3 4
	7 1 9 4
DeleteTombstone(4):	0 1 2 3 4

Nechť $h(1, 0) = 0, h(1, 1) = 3, h(1, 2) = 1, \dots$:

OpenFind(1): \Rightarrow úspěch	0 1 2 3 4 → [7 1 9 †]
---	----------------------------------

Asymetrické kryptosystémy (šifra RSA, Diffie-Hellman, RSA digitální podpis), hašovací funkce (SHA-2, HMAC)

BI-BEZ

- **Asymetrické kryptosystémy** – pro šifrování a dešifrování se používá pár rozdílných klíčů
 - o Veřejný klíč – veřejný, kdokoliv může zašifrovat zprávu a odeslat ji
 - o Soukromý klíč – nesdílený, pouze s tímto klíčem lze zprávu dešifrovat
- **RSA** – založeno na modulárním umocňování a faktorizaci velkých čísel
 - o **Definice RSA:**
 - Nechť p a q jsou prvočísla
 - Vypočítáme $n = pq$, $\phi(n) = (p-1)(q-1)$
 - Zvolíme e , $1 < e < n$, $\gcd(e, \phi(n)) = 1$ a spočítáme $d = |e^{-1}|_{\phi(n)}$, doporučení: $e > p, q$
 - Dvojici VK = (n, e) prohlásíme za veřejný klíč (a zveřejníme)
 - Dvojici SK = (n, d) prohlásíme za soukromý klíč
 - o Zašifrování OT (otevřeného textu):
 - Písmena → num. ekvivalenty, vytváříme bloky s největší možnou velikostí (sudý počet číslic)
 - Vztah pro zašifrování zprávy m na ŠT c:

$$E(m) = c = |m^e|_n, 0 < c < n$$
 - K dešifrování požadujeme znalost inverze d čísla e modulo $\phi(n)$, $\gcd(e, \phi(n)) = 1 \Rightarrow$ inverze existuje.
 - Pro dešifrování bloku c platí:

$$D(c) = |c^d|_n = |m^{ed}|_n = |m^{k\phi(n)+1}|_n = |(m^{\phi(n)})^k m|_n = |m|_n$$
 Kde $ed = k\phi(n) + 1$ pro nějaké celé číslo k ($|ed|_{\phi(n)} = 1$) a z Eulerovy věty platí $|p^{\phi(n)}|_n = 1$, kde $\gcd(p,n) = 1$

- o bezpečnost: se znalostí p, q lze použitím Euklidova algoritmu najít dešifrovací klíč d

- o **Problém faktORIZACE** – problém nalezení p, q
 - Bezpečnost RSA je postavena na předpokladu, že rozložit velké číslo na součin prvočísel (= faktorizace) je velmi obtížná úloha
 - Dosud nebylo prokázáno dešifrování zprávy zašifrované s použitím RSA bez faktorizace n!
 - **Výpočetní náročnost je tím větší, čím větší je modul**
 - Pokud je p, q 100číslicové, n je 200 a nejrychlejší algoritmus to dokáže dešifrovat za 250 let

Příklad

- Šifrovací modul je součinem dvou prvočísel 43 a 59. Potom dostáváme $n = 43 \cdot 59 = 2537$ jako modul.
- $e = 13$ je exponent, kde platí $\gcd(e, \phi(n)) = \gcd(13, 42 \cdot 58) = 1$.
- Dále platí $\Phi(2537) = (43 - 1) \cdot (59 - 1) = 42 \cdot 58 = 2436$.
- Pro zašifrování zprávy

 - PUBLIC KEY CRYPTOGRAPHY,
 - převedeme OT do číselných ekvivalentů písmen textu ⇒ vytvoříme bloky o délce 4 číslic (n je 4ciferné) a dostáváme:
1520 0111 0802 1004 2402 1724 1519 1406 1700 1507 2423,
Písmeno X = 23 je výplň (padding).
 - Pro šifrování bloku OT do bloku ŠT použijeme vztah $c = |m^e|_{2537}$.
Šifrováním prvního bloku OT 1520 dostáváme blok ŠT
 $c = |(1520)^{13}|_{2537} = 95$.

- Zašifrováním všech bloků OT dostáváme:
0095 1648 1410 1299 0811 2333 2132 0370 1185 1457 1084.
- Pro dešifrování zprávy, která byla zašifrována RSA šifrou, musíme najít inverzi $e = |13^{-1}|_{\Phi(n)}$, kde $\Phi(n) = \Phi(2537) = 2436$.
- S použitím Euklidova algoritmu získáme číslo $d = 937$, které je multiplikativní inverzí čísla 13 modulo 2436.
- K dešifrování bloku c ŠT použijeme vztah:
$$m = |c^{937}|_{2537}, 0 \leq m \leq 2537,$$

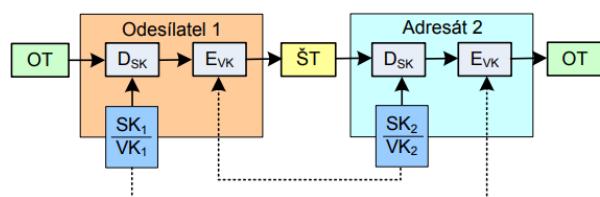
který platí, protože
$$|c^{937}|_{2537} = |(m^{13})^{937}|_{2537} = |m \cdot (m^{2436})^5|_{2537} = m,$$

kde jste použili Eulerovu větu
$$|m^{937}|_{2537} = |m^{2436}|_{2537} = 1,$$

když platí $\gcd(m, 2537) = 1$. Pokud to není splněno viz přednášky MI-KRY.

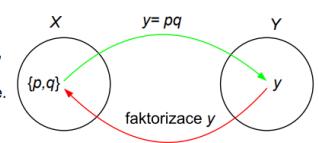
- **Digitální podpis a RSA** – RSA lze využít pro vyslání podepsané zprávy

- o Při použití podpisu se příjemce zprávy může ujistit, že zpráva přišla od oprávněného odesílatele, a že tomu tak je na základě nestranného a objektivního testu
- o Elektronická pošta, elektronické bankovnictví, elektronický obchod, ...
- o Postup:
 - otevřený text A zakóduje SK_a , pak VK_b a pošle
 - B dešifruje SK_b a pak VK_a

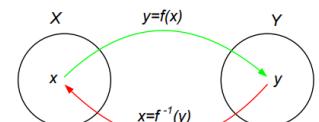


- **Diffie-Hellman** – algoritmus pro zízení společného klíče
 - o **Algoritmus:**
 - m ... velké prvočíslo (vhodné: $m = 2q + 1$, kde q je prvočíslo)
 - a ... generátor grupy Z_m^*
 - a, m jsou veřejné prvky
 - Každý subjekt si zvolí celé číslo k_j jako klíč, $\gcd(k_j, m - 1) = 1$
 - Subjekty si chtějí vygenerovat klíč:
 - Subjekt A pošle subjektu B celé číslo $y_A = |a^{kA}|_m$, $0 < y_A < m$
 - Subjekt B spočte společný klíč $K = |y_A^{kB}|_m = |a^{kAkB}|_m$, $0 < K < m$
 - B pošle A $y_B = |a^{kB}|_m$
 - A si spočte $K = |y_B^{kA}|_m = |a^{kAkB}|_m$
 - o Délka klíče je přímo úměrná kvalitě šifry
 - o Pro dva a více subjektů – může být využit jako šifrovací klíč v datové komunikaci, sestrojen tak, aby ho neautorizovaný subjekt nemohl rozluštit v rozumném čase
 - o **Problém diskrétního logaritmu (PDL)** – nechť G je grupa, $b \in G$ a $y \in G$ je mocninou prvku b. Potom diskrétní logaritmus prvku y o základu b je každé číslo k takové, že $b^k = y$, značíme ho $\log_b y$.
 - Problém nalezení k = problém diskrétního logaritmu.
 - Platí normální věty o logaritmech.
 - složitost $O(n)$, $n =$ počet prvků grupy
 - u D-H se používá grupa $GF(2^{1024})$ nebo $GF(2^{2048})$
- **Hašovací funkce** – libovolně velkému vstupu přiřazuje krátký hašovací kód o pevně definované délce, která má navíc vlastnost *jednosměrnosti* a *bezkoliznosti*
 - o **Jednosměrná funkce:** $f: X \rightarrow Y$, pro něž je snadné z jakékoli hodnoty $x \in X$ vypočítat $y = f(x)$, ale pro nějaký náhodně vybraný obraz $y \in f(X)$ nelze najít její vzor $x \in X$ tak, aby $y = f(x)$. Přitom víme, že minimálně jeden takový vzor existuje
 - o **Orákulum** – stroj, který na základě vstupu odpovídá nějakým výstupem
 - o **Náhodné orákulum** – orákulum, které na nový vstup odpovídá náhodným výběrem výstupu z množiny výstupů
 - o Hašovací funkce se musí chovat jako *náhodné orákulum* kvůli bezpečnosti
 - o **Bezkoliznost 1. řádu:** h je odolná proti kolizi 1. řádu, jestliže nemůžeme nalézt dvě různé zprávy M a N tak, že $h(M) = h(N)$ (tedy že sdílejí hash). Pokud se toto stane, říkáme, že jsme nalezli kolizi.
 - o Bezkoliznost – využití u *digitálních podpisů* – nepodepisuje se přímo zpráva, ale pouze její hash
 - o **Bezkoliznost 2. řádu:** h je odolná proti kolizi 2. řádu, jestliže pro jakýkoliv konkrétní vzor x nemůžeme nalézt 2. vzor y $\neq x$ tak, že $h(x) = h(y)$.
 - o Odolnost proti nalezení kolize 1. řádu – složitost nalezení kolize s 50 % pravděpodobností je $\approx 2^{n/2}$ (vyplývá z *narozeninového paradoxu*)
 - o Odolnost proti nalezení kolize 2. řádu $\approx 2^n$
 - o **Prolomení hašovací funkce** – pokud jsme schopni nalézt vzory jednodušeji, než říká odolnost
 - o Kryptografické využití hašovacích funkcí – kontrola integrity, autentizace původu dat, ukládání a kontrola přihlašovacích hesel, jednoznačná identifikace dat, prokazování znalosti, pseudonáhodné generátory, odvozování klíčů
 - o **Narozeninový paradox** – pro n-bitovou hašovací funkci nastává kolize s $\approx 50\%$ pravděpodobností v množině $2^{n/2}$ zpráv, namísto očekávaných 2^{n-1}
 - o **Konstrukce moderních hašovacích funkcí** – zpráva je většinou velmi dlouhá, proto se zpracovává v blocích, musí se zarovnat na celistvý počet bloků
 - musí umožňovat jednoznačné odejmutí (doplní se 1 a zbytek 0)
 - potřeba pro D-M konstrukci

Jednosměrné funkce 1. typu
Jednosměrnost je dána složitostí operace a její inverze, např. násobení dvou prvočísel (p a q) vs. faktORIZACE.



Jednosměrné funkce 2. typu
Jednosměrnost je dána znalostí „padacích vrátek“, například u asymetrické kryptografie je to klíč.



- Damgard-Merkelovo zesílení – doplnění o délku původní zprávy

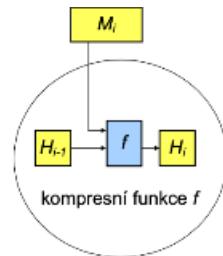
- U moderních hašovacích funkcí nutnost **hašování zprávy postupně po blocích** (kvůli možné velké délce zprávy) – nutnost zarovnání vstupní zprávy na celistvý počet bloků před hašováním
- Zarovnání musí být bezkolizní a také proto musí umožňovat jednozn. odejmutí (jinak by vznikaly kolize)
- Zpráva M je nejprve doplněna bitem 1 → bity 0 (může jich být 0 – 511) tak, aby do celistvého násobku 512 bitů zbývalo ještě 64 bitů
- Posledních 64 bitů je vyplňeno 64bitovou hodnotou # bitů původní zprávy M
- Délka zprávy je součástí hašovacího procesu – 64 bitů na délku zprávy umožňuje hašovat zprávy až do délky $d = 2^{64} - 1$ bitů.
- Začleněním informace o délce původní zprávy eliminujeme případné útoky
- Používá se D-M princip **iterativní hašovací funkce** s využitím **kompresní funkce f** (měla by být robustní)
 - f zpracovává aktuální blok zprávy M_i a výsledkem je určitá hodnota, tzv. **kontext** - H_i
 - Hodnota H_i nutně tvoří vstup do f v dalším kroku – f má tedy dva vstupy - H_{i-1} a aktuální M_i . Výstup = nové H_i .
 - **Damgard-Merkelova konstrukce** – iterativní konstrukce popsána vzorcem

$H_i = f(H_{i-1}, M_i)$
 $= \text{nový kontext je zpracování předchozího kontextu a aktuálního bloku zprávy}$
 $H_0 = IV$
 $= \text{počáteční hodnota } H_0 \text{ se nazývá inicializační hodnota} - IV - \text{dodefinována jako konstanta daná v popisu každé iterativní hašovací funkce}$

 - Po zahašování posledního bloku M_N dostáváme H_N , z něhož bereme buď celou délku nebo část jako výslednou haš (např. MD5 má šířku kontextu (= výsledné hodnoty kompresní funkce) 128 bitů a výslednou haš tvoří všech 128 bitů H_N)
 - Pokud je hašovací funkce bezkolizní, je bezkolizní i kompresní funkce
 - **Davies-Meyerova konstrukce** kompresní funkce – zesiluje vlastnost jednosměrnosti ještě přičtením (XOR) vzoru před výstupem

$H_i = f(H_{i-1} \oplus M_i) = E_{M_i}(H_{i-1}) \oplus H_{i-1}$

 - Navíc maskován vstupem
 - Blok zprávy M_i je obvykle větší než klíče blokových šifer → bloková šifra se aplikuje několikrát za sebou – v **rundách**
 - Rundovní klíč je postupně čerpán z bloku M



- SHA-x

- Kompresní a hašovací funkce založené na D-M
- SHA-1 umožňuje zpracovat zprávu M s maximální délkou $2^{64} - 1$ bitů a vrací výstupní hašový kód (kontext) s délkou 160b. Vstupní zpráva M je zpracována po blocích s velikostí 512b.
- Mezi verzemi SHA-x nejvýznamnější rozdíly jsou v délce hašového kódu, který určuje **odolnost hašového kódu vůči nalezení kolizí** 1. a 2. Řádu
- 80 rund (80x zašifrované - může být i různými funkciemi f, g, h, \dots)

	SHA-1	SHA-256	SHA-384	SHA-512
Délka haš. kódu	160	256	384	512
Délka zprávy	< 2^{64}	< 2^{64}	< 2^{128}	< 2^{128}
Velikost bloku	512	512	1024	1024
Velikost slova	32	32	64	64
# rund f	80	80	80	80
Bezpečnost v bitech	80	128	192	256

- HMAC

- Obecná konstrukce, která využívá obecnou hašovací funkci – podle konkrétní využité funkce se pak označuje např. HMAC-SHA-1, ...
- Zpracovávají hašováním nejen zprávu M, ale spolu s ní nějaký **tajný klíč K**
- Podobné autentizačnímu kódu zprávy MAC, ale místo blokové šifry využívají hašovací funkci
- Použití k **nepadělatelnému zabezpečení** zpráv a k autentizaci
 - Definujeme konstantní řetězce **ipad** jako řetězec $b/8$ bajtů s hodnotou $0x36$ (0011 0110) a **opad** jako řetězec $b/8$ bajtů s hodnotou $0x5C$ (0101 1100), kde b je velikost bloku v bitech.
 - Klíč K v případě, že $\log_2 K < b$, doplníme bity 0 vlevo od MSB bitu klíče do délky b -bitů a označíme ho K^+ .
 - Definujeme hodnotu $HMAC_K(M)$ jako

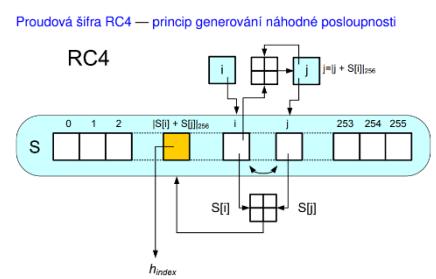
$$HMAC_K(M) = H((K^+ \oplus opad) \| H((K^+ \oplus ipad) \| M)),$$

kde $\|$ označuje zřetězení.

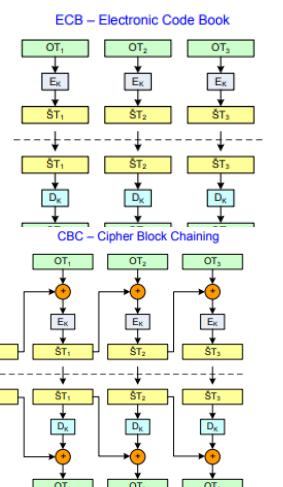
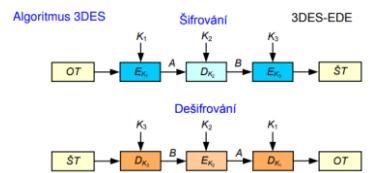
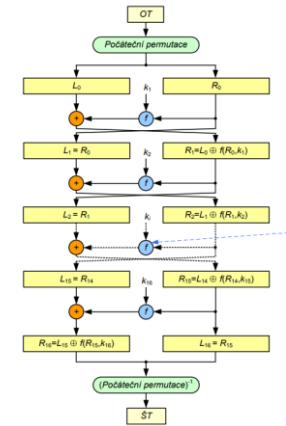
Symetrické šifry blokové a proudové (AES, 3DES, RC4) základní parametry, operační módy blokových šifer (ECB, CBC, CFB, OFB, CTR, MAC), jejich základní popis a slabiny

BI-BEZ

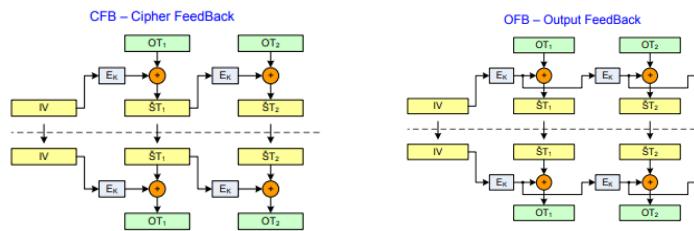
- **Symetrické šifry** – používají stejný klíč pro šifrování a dešifrování
 - o K šifrování a dešifrování používají stejný klíč
 - o Nižší výpočetní náročnost než u asymetrických šifer
 - o **Synchronní** = hesla nezávisí na ŠT ani OT, pokud něco vypadne, naruší se čitelnost
 - o **Asynchronní/samosynchronní** = proud hesla závisí na OT nebo ŠT, dokáže se samo opravit
- **Synchronní proudové šifry**
 - o Šifrují zvlášť jednotlivé znaky abecedy
 - o Z klíče k vygenerují posloupnost h_1, h_2, \dots a každý znak otevřeného textu šifrují jinou transformací E_{h_i}
 - o Vernamova šifra – náhodné heslo stejně dlouhé jako OT, ničilo se, vždy unikátní – absolutní bezpečnost
 - o **Algoritické proudové šifry:**
 - Heslo se vypočítá na základě tajného klíče
 - Aby klíč nemusel být měněn příliš často – princip náhodně se měnícího inicializačního vektoru (IV)
 - IV pro každou zprávu vybírá náhodně a přenáší před ŠT v otevřené podobě
 - IV nastavuje příslušný algoritmus vždy do jiného počátečního stavu \rightarrow i při stejném tajném klíči je generována pokaždé jiná heslová posloupnost
 - Za různost hesla odpovídá IV, za utajenost zodpovídá tajný šifrovací klíč
 - o **Synchronní proudové šifry** – když proud hesla nezávisí na OT ani ŠT – příjemce a odesílatel je přesně synchronizován
 - o Mají nedostatečnou vlastnost difúze (operují nad jednotlivými znaky, ne bloky)
 - o Jednotlivé bity proudu hesla jsou postupně slučovány s jednotlivými bity proudu OT binárním sčítáním (většinou XOR)
 - o Malá propagace chyby (pouze v jednom znaku)
 - o **RC4** – považována za prolomenou
 - Nevyužívá IV – na každý spojený generuje náhodně nový tajný klíč (pomocí asymetrické metody)
 - Umožňuje volit délku klíče, nejčastěji 40b a 128b
 - Šifrovací klíč se používá pouze k vygenerování tajné substituce $0, \dots, 255 \rightarrow 0, \dots, 255$, tedy substituci bajt za bajt
 - Pomocí tabulky S se pak konečným automatem generují jednotlivé bajty hesla, které se xorují na OT nebo ŠT
 - Nejprve se podle tajného klíče vytvoří náhodná permutace, poté se začíná šifrovat OT a při každém kroku se mění stav pole (pořád swapuju $S(i)$ a $S(j)$)
- **Blokové symetrické šifry**
 - o Šifrování po blocích textu, poslední doplněn o padding
 - o Všechny bloky OT jsou šifrovány toutéž transformací a všechny bloky ŠT jsou dešifrovány toutéž transformací
 - o Šifrovací systém (M, C, K, E, D), kde E a D jsou zobrazení definující pro každé $k \in K$ transformaci zašifrování E_k a dešifrování D_k tak, že zašifrování bloků OT m_1, m_2, \dots , kde $m_i \in M$, probíhá podle vzoru $c_i = E_k(m_i)$ a dešifrování podle vztahu $m_i = D_k(c_i)$ pro každé $i \in N$
 - o Nejčastěji blok 128 bitů
 - o Někdy využití principu **algoritmů Feistelova typu** – umožňující postupnou aplikací relativně jednoduchých transformací vytvořit složitý kryptografický algoritmus
 - V symetrických šifrovacích algoritmech – DES
 - Vezmu zprávu m , tu rozdělím na dvě, proženu jednu část nějakou funkcí (třeba změnou permutace), sečtu (např. naXORuji) se druhou částí a vznikne mi nová část textu. Opakuju v h rundách
 - o Lze používat v různých operačních módech



- Dobré vlastnosti konfúze i difúze
- **DES – iterovaná šifra**
 - 16 rund, 64b bloky OT a ŠT
 - Šifrovací klíč má délku 56b – v průběhu expandován na 16 rundovních klíčů, které jsou řetězci 48 bitů, každý z nich je některým bitem původního klíče k
 - Místo počátečního zašumění OT se používá bezklíčová pevná permutace a místo závěrečného zašumění permutace k ní inverzní
 - Rundovní funkce – binární načtení klíče kj na vstup
 - Pevná nelineární substituce 6b znaků do 4b s následnou bitovou transpozicí, pomocí S-boxů, díky tomu dobrá difúze i konfúze
 - lehce prolomitelný, dnes nahrazen AESem
 - rozdělení počáteční permutace na $2 \times 32b$
 - ve smyslu feistela je $2n = 64$ a $h = 16$
 - dešifrování probíhá jako šifrování, ale obrátí se pořadí rundovních klíčů
- **3DES – Triple DES**
 - Prodlužuje klíč originální DES tím, že používá DES jako stavební prvek celkem 3×8 s 2 různými klíči
 - Klíč buď 112b (2 klíče) nebo 168b (3 klíče)
 - nejčastěji se používá EDE
- **AES – není feistelovského typu**
 - Délka bloku 128 bitů
 - Nemá slabé klíče, odolná proti hrubému útoku
 - 3 délky klíče – 128, 192, 256b – částečně se mění algoritmus
 - Platný šifrovací standard
 - 32b rundovní klíče, které se odvozují ze šifrovacího klíče
 - 4 fáze rundy (16B vstup):
 - SubBytes – substituce bytů pevnou tabulkou
 - ShiftRows – posun řádků v matici
 - MixColumns – další substituce, neprovede se v poslední rundě
 - AddRoundKey – naxorování s rundovním klíčem
 - Odolný proti útokům
 - Vhodný pro paralelní zpracování a má malé nároky na paměť
 - Pracuje s prvky Galoisova tělesa $GF(2^8)$
- **Operační módy blokových šifer** – způsoby použití blokových šifer v daném kryptosystému
 - **ECB (Electronic Codebook)** – základní mód, každý blok se šifruje zvlášť
 - Nevýhoda – stejně bloky OT mají vždy stejný šifrový obraz
 - útočník může bloky prohazovat
 - **CBC (Cipher Block Chaining)** – Nejpoužívanější mód
 - Pro rozšíření difúze – řetězení šifrového textu (proto cipher block chaining) – každý blok OT se nejprve modifikuje předchozím blokem ŠT, a teprve poté se šifruje
 - Běžný šifrový blok závisí na celém předchozím OT
 - 1. blok OT modifikován náhodnou hodnotou IV
 - Pozitiva: samosynchronizace, dokáže se opravit už po dvou blocích
 - **CFB a OFB (Cipher Feedback, Output Feedback)**
 - Převádí blokovou šifru na proudovou
 - Inicializační hodnota IV nastavuje konečný automat do náhodné polohy
 - Automat produkuje posloupnost hesla, které se jako u proudových šifer xoruje na OT, první blok hesla se získá zašifrováním IV

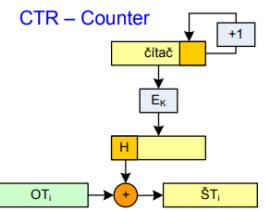


- Vzniklé heslo (OFB) nebo vzniklý ŠT (CFB) se přivádějí na vstup blokové šifry a jejich zašifrováním je získán další blok hesla
- OFB má vlastnost synchronní proudové šifry – heslo generováno bez vlivu OT a ŠT
- CFB – kombinace vlastností CBC a proudové šifry, samosynchronní



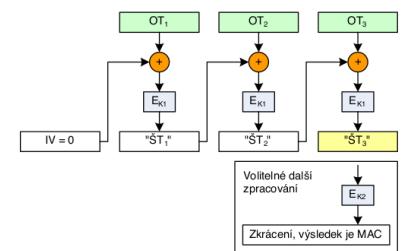
○ CTR (Counter Mode)

- Podobný OFB – převádí blokovou šifru na synchronní proudovou
- Délka periody hesla dána předem
- IV se načte do vstupního registru (čítače) T. Po jeho zašifrování vzniká první blok hesla. Poté dojde k aktualizaci čítače T, nejčastěji přičtením jedničky a ke generování dalšího bloku hesla
- Obsah čítače se nesmí opakovat, jinak by dvojí použití hesla znamenalo rozluštění OT
- Výhoda – heslo může být vypočítáno na základě pozice OT a IV nezávisle na ostatním
- smyslem je zaručení maximální periodicity hesla (pomocí čítače)



○ MAC (Message Authentication Code) – podobný CBC

- Mód B. Š., který řeší zajištění neporušnosti dat
- MAC = zabezpečovací kód který autentizuje původ zprávy a řeší obranu proti náhodným i úmyslným změnám na komunikačním kanálu
- Vznikne zpracováním zprávy s tajným klíčem, který by měl být jiný, než ten k šifrování zprávy
- Zajišťuje autentizaci původu dat, ne nepopiratelnost
- proudové a blokové šifry zaručují důvěrnost, a ne integritu zpráv
- krátký kód, který je využíván pro digitální podpis pro symetr. Šifry



○ Metoda solení

- Komunikujícímu protějšku se předává hodnota IV, ale k šifrování se použije jiná, „osolená“ hodnota IV'
- Tato hodnota se na obou stranách vypočítá z IV a klíče K nějakým definovaným způsobem, např zřetězení IV a K

Infrastruktura veřejného klíče, distribuce klíčů, digitální podpis. Certifikáty, certifikační autority. Kryptograficky bezpečné generátory náhodných čísel

BI-BEZ

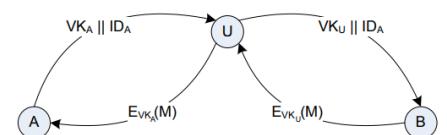
- **Infrastruktura veřejných klíčů (PKI)** – systém digitálních certifikátů, certifikačních autorit (CA) a dalších registračních úřadů, které slouží k ověřování platnosti každé strany zúčastněné v určité elektronické transakci.

- o jedná o specifikaci technických a organizačních opatření pro vydávání, správu, používání a odvolávání klíčů a certifikátů
 - o hlavní cíl: zabezpečení kompatibility SW pro Internet

- **Distribuce klíčů** – nutné pro komunikaci subjektů

- o S distribucí VK souvisí hrozba podvržení veřejného klíče – ohrožení bezpečnosti

- o Hrozí podvržení útočníkem prostřednictvím útoku man-in-the-middle. Je to útok, kde komunikace probíhá přes prostředníka, který zaměňuje klíče za své a tím může číst obsah zpráv a subjekty to nezjistí. Existuje více technik, jak tento problém řešit.



- o **Způsob podvržení VK:**

- A pošle svůj VK_A a svůj identifikátor ID_A, tj. zprávu VK_A || ID_A subjektu B
 - Útočník U má aktivní přístup k veřejnému kanálu → zachytí zprávu VK_A || ID_A a vytvoří novou zprávu VK_U || ID_A a odešle ji B – podvržení VK_U subjektu B
 - B si myslí, že VK_U, co dostal je od A → B zašifruje každou zprávu M klíčem VK_U, tedy vytvoří E_{VK_U}(M) a odešle ji A
 - U zachytí E_{VK_U}(M), dešifruje ji SK_U a získává zprávu M
 - U zná VK_A → zašifruje M na E_{VK_A}(M) a pošle ji A

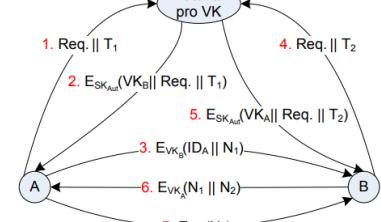
→ U čte korespondenci zaslanou subjektem B subjektu A, může tyto zprávy měnit a nemusí to být B detekováno

- o **Realizace distribuce VK:**

- **Zveřejnění veřejných klíčů (public announcement)**
 - Zasláním VK individuálně nebo hromadně v rámci komunity, rychlé a iži, ale nízká bezpečnost – není odolné proti podvržení VK
 - **Veřejně dostupný adresář (public available direktory)**
 - Distribuci VK zabezpečuje důvěryhodná autorita – správce adresáře, účastníci prostřednictvím ní registrují svůj VK – osobně nebo přes bezpečnou komunikaci, změna VK podle potřeby
 - vyšší stupeň bezpečnosti
 - pokud útočník získá SK správce adresáře, může ho modifikovat dle libosti



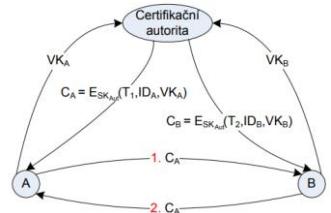
- **Autorita pro veřejné klíče (public-key authority)**
 - Přísnější dohled na distribuci VK → vyšší bezpečnost, autorita vykonává činnost správce adresáře VK, každý účastník zná veřejný klíč autority VK_{Aut}, která vlastní odpovídající soukromý klíč SK_{Aut}
 - **1,2 a 4,5** – A získá VK B, B získá VK A – autorita šifruje svým private klíčem, aby se potvrdilo, že lze rozšifrovat public klíčem
 - **3,6,7** – jistý handshaking, že druhý subjekt dokáže přijmout a rozšifrovat zprávu (N1, N2) svým klíčem



- **Certifikace veřejných klíčů (public-key certification)**

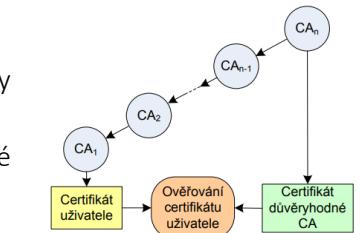
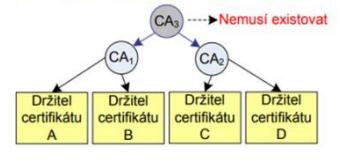
- Distribuce VK bez kontaktu s třetím důvěryhodným subjektem
 - Vyžaduje se certifikát a certifikační autorita
 - **Certifikát** – struktura, která obsahuje:
 - o Veřejný klíč žadatele/držitele certifikátu

- Identifikační údaje držitele certifikátu
- Dobu platnosti certifikátu
- Další údaje vytvořené certifikační autoritou
- Certifikát je podepsán *soukromým klíčem certifikační autority SK_{Aut}* a každý účastník může verifikovat obsah certifikátu pomocí *veřejného klíče certifikační autority VK_{Aut}*
- **Certifikační autorita CA** – důvěryhodná třetí strana, která na základě žádosti vydává a aktualizuje certifikáty, každý účastník může verifikovat to, že certifikát byl vytvořen CA, pomocí jejího veřejného klíče VK_{Aut}
- Doporučení X.509 – určuje formáty certifikátů, definuje autentizační protokoly
 - Formát certifikátu, sériové číslo certifikátu, algorit. vytvoření podpisu certifikátu, identifikační údaje CA, doba platnosti certifikátu, identifikační údaje uživatele, veřejný klíč uživatele, jednoznačný identifikátor CA, jednoznačný identifikátor uživatele, rozšíření, digitální podpis CA

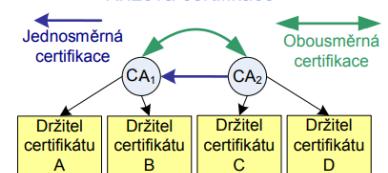


- **Certifikační strom** – propojuje více vytvořených CA formou stromové struktury
 - Každý strom reprezentuje **kořenová CA**, která vlastní **kořenový certifikát**
 - **Řetězec certifikátů** – posloupnost certifikátů od certifikátu uživatele až k certifikátu kořenové CA
 - Certifikát je platný \Leftrightarrow platné jsou všechny certifikáty v řetězci
 - Kořenový certifikát musí být ověřen jinou bezpečnou cestou (např. *křížová certifikace*)
 - pokud existuje více CA pro různé okruhy lidí, vznikají oddělené stromy certifikátů
 - **Křížová certifikace** – při nutnosti získání certifikátů uživatelů jiné stromové struktury
 - Pokud C chce komunikovat s A, musí A poslat C množinu cert.:
 - Certifikát A, podepsaný CA₁
 - Certifikát CA₁, podepsaný CA₁ – *kořenový certifikát CA₁*
 - Certifikát CA₁, podepsaný CA₂ – *křížový certifikát*
 - **PKI** – Public Key Infrastructure – norma v síti Internet, specifikace technických a organizačních opatření pro vydávání, správu, používání a odvolávání klíčů a certifikátů, hlavní cíl = zabezpečení kompatibility SW pro Internet
 - **Distribuce tajných klíčů**
 - pokud A chce komunikovat s B, musí oba realizovat uvedené kroky
 - A pošle B zprávu obs. VK_A a identifikátor ID_A – poté obdrží A od B tajný klíč k_s zašifrovaný VK_A
 - Podvrzením veřejného klíče lze získat tajný klíč k_s – provádí se stejně jako u podvrzení veřejného klíče

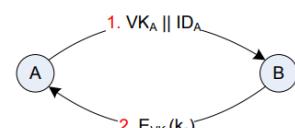
Stromová struktura certifikačních autorit



Křížová certifikace



- **Digitální podpis** – mechanismy bezpečnosti využívající kryptografických transformací na zabezpečení autentizace zdroje zprávy a integrity dat
 - funguje jako potvrzení, že přijatá zpráva (i nezašifrovaná) je pravá a nepozměněná
 - Forma *asymetrického* kryptografického schématu
 - Soukromý klíč – podepsání
 - Veřejný klíč – ověření
 - Vlastnosti digitálního podpisu:
 - **Nezfalšovanost, autentizace** – podpis se nedá napodobit nikým jiným než podepisujícím
 - Ověřitelnost – příjemce dokumentu musí být schopen ověřit, že je podpis platný
 - **Integrita** – podepsaná zpráva se nedá změnit, aniž by se zneplatnil podpis
 - **Nepopiratelnost** – podepisující nesmí mít později možnost poprít, že dokument nepodepsal



- Kategorie – přímé (problém s popiratelností), verifikované (využívá důvěryhodnou třetí stranu)
 - DSA – digital signature algorithm
 - dělí se na:
 - **přímé** – mezi 2 subjekty, příjemce zná VK odesílatele, problém s popiratelností (není nikdo třetí kdo by ho usvědčil)
 - **verifikované** – využívá důvěryhodnou třetí stranu, který ověřuje podpisy
 - **Generátory náhodných čísel**
 - **Náhodné číslo** – číslo vygenerované procesem, který má nepředvídatelný výsledek a jehož průběh nelze přesně reprodukovat = generátor náhodných čísel
 - Od náhodných posloupností jsou očekávány vlastnosti:
 - *rovnoměrné rozdělení* (stejná pravděpodobnost)
 - generované hodnoty jsou na sobě *nezávislé* (není mezi nimi korelace)
 - **PRNG** – generátor pseudonáhodných čísel – algoritmus, jehož výstupem je posloupnost, která není náhodná, ale zdá se být náhodná, pokud nejsou známy některé parametry generátoru
 - rychlé, snadno realizovatelné, dobré statistické vlastnosti
 - **Lineární kongruenční generátor**
- $X_{n+1} = (aX_n + c) \bmod m$
- X ... posloupnost pseudonáhodných čísel
 m ... modul
 a ... násobitel
 c ... inkrement
 X_0 ... počáteční hodnota - **seed**
- **Kryptograficky bezpečné PRNG** – musí splňovat požadavky:
 - **Next-bit test** – je-li známo prvních k bitů náhodné posloupnosti, neexistuje žádný algoritmus s polynomiální složitostí, který by dokázal předpovědět $(k+1)$. bit s pravděpodobností úspěchu vyšší než 50%
 - **State compromise** – i když je zjištěn vnitřní stav generátoru, nelze zpětně zrekonstruovat dosavadní vygenerovanou náhodnou posloupnost
 - Příklady kryptograficky bezpečných PRNG:
 - **Bezpečná bloková šifra v režimu čítače** – náhodně se zvolí klíč (seed) a počáteční hodnota čítače i . Zvoleným klíčem se postupně šifrují hodnoty $i, i + 1, \dots$, perioda 2^n , nesmí dojít k vyzrazení klíče ani počáteční hodnoty čítače
 - **Kryptograficky bezpečná hešovací funkce aplikovaná na čítač** – náhodně se zvolá počáteční hodnota čítače i , postupně se hashuje $i, i+1 \dots$, nesmí dojít k prozrazení počáteční hodnoty čítače
 - **Proudové šifry** – PRNG, s jehož výstupem se XORuje plaintext
 - **Algoritmy založené na teorii čísel** – u kterých byl proveden nějaký důkaz bezpečnosti
 - **Blum-Blum-Shub** – pomalý, silný důkaz bezpečnosti
 - **Seed** – náhodný a tajný vstup PRNG – musí být skutečně náhodný, udává entropii výstupu PRNG
 - **TRNG** – True Random Number Generator – využívají zdroj entropie, který je zpravidla nějaký fyzikální jev nebo vnější vliv (*radioaktivní rozpad, atmosférický/teplotní šum, chování uživatele, ...*)
 - složitější implementace
 - Výstup není předvídatelný, i když známe všechny parametry, má horší statické vlastnosti → nutné následné zpracování – *post-processing*
 - **Post-processing** – odstranění nevyváženosti jedniček a nul (*bias*) a zajištění rovnoměrného rozdělení, extrakce entropie – zvýšení entropie výstupních bitů za cenu snížení rychlosti jejich generování (*bitrate*)
 - **John von Neumannův dekolerátor** – eliminuje nevyváženosť a snižuje korelovanost výstupu
 - bity se odebírají po dvou
 00, 11 → - (vstup se zahodí)
 01 → 0
 10 → 1

Relační databáze, dotazování v relační algebře, základní koncepce jazyka SQL (SELECT, DDL, DML, DCL, TCL), vyjádření integritních omezení v DDL

BI-DBS

- Relační databáze

- **Databáze** = soubor záznamů (zpráv), jako jsou znaky, čísla, diagramy, jejichž systematická struktura umožňuje, aby tyto zprávy mohly být vyhledávány pomocí počítače
- Zabývá se řízením velkého množství, perzistentních, spolehlivých a sdílených dat
 - **velkého množství**
 - pro data nestačí operační paměť,
 - **perzistentních**
 - data přetrvávají od zpracování ke zpracování,
 - **spolehlivých**
 - data lze rekonstruovat po chybě,
 - **sdílených**
 - data jsou přístupná více uživatelům:
 - ▶ užívání na základě přístupových práv,
 - ▶ koordinované současné využívání stejných dat více uživateli.
- **DBMS** = systém řízení bází dat (database management system)
- **Smysl a přínos DB**
 - Nezávislost dat na aplikaci
 - Efektivní přístup k datům
 - Urychlení vývoje aplikací
 - Integrita a ochrana dat
 - Správa a zálohování dat
 - Transakční zpracování
 - Paralelní přístup k datům
 - Zotavení po chybě
- **Relace** – množina n-tic
 - Dvourozměrná struktura (obsahující řádky a sloupce)
 - Jména atributů [A1,A2,A3, ...,An]
 - Domény atributů Di – atomické
 - N-tice (a1, a2, ..., an) (prvek relace)
 - Množina n-tic $\subset D_1 \times D_2 \dots \times D_n$
 - Jméno relace R
 - Schéma relace R(A) = záhlaví tabulky
 - Neobsahuje duplicitní n-tice
- **Tabulka** – záhlaví, jméno sloupce, sloupec, řádek tabulky
- **Relační databáze**
 - (R, I) je schéma relační databáze, kde:
 - R = {R1...Rn} je množina relací
 - I je množina integritních omezení
 - Přípustná relační databáze se schématem (R, I) = množina relací takových, že jejich n-tice vyhovují tvrzením v I
- **Klíč schématu** R(A) = taková minimální podmnožina atributů z A, která jednoznačně určí každou n-tici relace R*
- **Dotaz nad schématem** S = výraz, který vrací odpověď se schématem T, přičemž:
 - Def. Oborem jsou všechna úložiště se schématem S
 - Oborem hodnot jsou všechny relace se schématem T
 - Data v odpovědi pocházejí z databáze
 - Odpověď nezávisí na fyzickém uložení dat
- **Dotazovací jazyk** = množina všech použitelných výrazů pro konstrukci dotazu
- **Relační algebra** – dotazovací jazyk (neřeší DML a DDL)
 - Je vyšší jazyk – nespecifikuje jak, ale co má být výsledkem
 - Dotazovací jazyk, který umožňuje realizovat RA je relačně úplný (např SQL)
 - Výsledkem dotazu je relace, která může být vstupem do dalšího dotazu – lze řetězit
 - Výrazy – dotazy – z operací a operandů (operand je vždy celá relace)

- **Selekce** (restrikce) relace R dle podmínky $\phi = R(\phi)$ – výběr **řádku** tabulky podle kritéria
 - Definice: $R(\phi) =_{\text{def}} \{u \mid u \in R \wedge \phi(u)\}$, kde ϕ je logický výraz
 - Relace R dle podmínky ϕ
 - Např. table(id = 1)
- **Projekce** relace R na množinu atributů C, kde $C \subseteq A = R[C]$ – výběr **součtu** tabulky podle názvů
 - Definice: $R[C] =_{\text{def}} \{u[C] \mid u \in R\}$, kde $C \subseteq A$
 - Např. table[id, name]
- **Přirozené spojení** relací R(A) a S(B) s výsledkem T(C) = R * S
 - $R * S =_{\text{def}} \{u \mid u[A] \in R \wedge u[B] \in S\}$, kde $C = A \cup B$ a výběr n-tic pro spojení je dán rovností na všech průnikových atributech A a B
 - Pravé/levé (s tím, že ve výsledku jsou jen atributy pravé/levé relace)
- **Přejmenování atributu**
 - $t \rightarrow \text{alias}$
- **Obecné spojení** ($R[t_1\Theta t_2]S$)
 - $R[t_1\Theta t_2]S =_{\text{def}} \{u \mid u[A] \in R, u[B] \in S, u.t_1\Theta u.t_2\}$, kde $\Theta \in \{<, >, =, \leq, \geq, !=\}$ nebo logická spojka
 - výsledek má všechny atributy, společné se opakují (třeba přejmenovat)
 - pravé/levé (s tím, že ve výsledku jsou jen atributy pravé/levé relace)
- **Množinové operace**
 - **sjednocení** – $R \cup S$
 - **průnik** – $R \cap S$
 - množinový **rozdíl** – $R \setminus S$ (někdy též klasicky $R - S$)
 - **kartézský součin** – $R \times S$
- **Antijoin** (R neg $<*$ S)
 - $R <* S =_{\text{def}} R \setminus \{R <* S\}$
 - Podmnožina n-tic z R, které nejsou spojitelné s žádnou n-ticí z S.
- **SQL** – dotazovací jazyk pro práci s relačními DB
 - Říkáme, co chceme získat, ne jak se to má dělat
 - Intuitivní zápis, připomíná anglické věty
 - Syntaxe úzce spojená s RA
 - **DDL** (Data Definition Language)
 - Jazyk pro definici dat
 - Např. manipulace s tabulkami, nebo integritní omezení createtable
 - *CREATE TABLE teachers (id int, name varchar(255)); ALTER TABLE, DROP TABLE*
 - **DML** (Data Manipulation Language)
 - Jazyk pro manipulaci dat a dotazování
 - Příkazy INSERT, UPDATE, DELETE, MERGE
 - Pro upravování tabulek
 - MERGE – kombinace příkazů INSERT a UPDATE
 - Na základě referenční tabulky provádíme buď insert nebo update záznamů v cílové tabulce
 - Při zpracování DML příkazů DBMS automaticky hlídá platnost integritních omezení
 - **Pohledy** – VIEW – virtuální relace
 - V systémovém katalogu je uložen ve formě SELECT příkazu, kterým je definován
 - Z hlediska dotazování je pohled zaměnitelný s tabulkou
 - DML operace nad pohledy v omezené míře
 - Jsou k odstínění informací, které uživatel/role nemá vidět, zpřehlednění složitých dotazů a snazšímu vývoji aplikací
 - **WITH** – pro dočasné pohledy, i pro rekursivní dotazování
 - **DCL** (Data Control Language)
 - Jazyk pro přiřazení přístupových práv – GRANT přidává právo, REVOKE odebírá
 - Např. *GRANT INSERT, UPDATE, REVOKE*

- Grantovat lze (dle typu objektu): SELECT, INSERT, UPDATE, DELETE, ALTER, EXECUTE, INDEX, REFERENCE
- Kdo vytvoří objekt, je jeho vlastníkem a může s ním manipulovat
- **TCL** (Transaction Control Language)
 - Jazyk pro řízení transakcí
 - Pro admin správu databáze, pro přiřazení práv pro použití/úpravu DB
 - Session – jedno navázané spojení s DB serverem – v session probíhají transakce
 - Lze mít otevřeno více session i pod jedním DB uživatelem
 - Např. *commit, rollback*
 - **COMMIT** – potvrzení změn – všechny provedené změny jsou persistentně uloženy v databázi, ostatní session od této chvíle provedené změny vidí
 - **ROLLBACK** – odvolání změn – všechny změny provedené v příslušné transakci jsou odvolány
 - **SAVEPOINT** – možnost definování „značky“ uvnitř transakce, ke které lze vztáhnout ROLLBACK
 - AUTOCOMMIT ON/OFF – řídí chování na úrovni session
 - ON – každý DML příkaz je automaticky potvrzen (COMMIT)
 - OFF – musí přijít explicitní COMMIT nebo ROLLBACK
- **SELECT** – základní syntaxe SQL, zobrazuje tabulku na základě vybraných atributů a podmínek
 - **Základní syntaxe:**

```
SELECT specifikace_sloupců
      FROM specifikace_zdroje
      [WHERE podmínka_selekce]
      [ORDER BY specifikace_řazení]
```

 - Specifikace_sloupců = {DISTINCT | ALL} * | {jm_sloupce [, jm_sloupce]...}
 - DISTINCT – vztahuje se vždy na celou klauzuli SELECT
 - Podmínky selekce – AND, BETWEEN AND, =, <, IN, LIKE, EXISTS, ...
 - WHERE vyhodnocuje výraz jako TRUE, FALSE nebo NULL
 - **Spojení tabulek – JOIN**
 - [{LEFT | RIGHT | FULL} [OUTER]] JOIN ON/JOIN USING/NATURAL JOIN
 - CROSS JOIN
 - Aliasy – použití v joinech mezi tabulkami pro nalezení stejného atributu
 - FROM Filmy JOIN Predstaveni ON (Filmy.nazev = Predstaveni.nazev)
 - **Vnější spojení** – outer join – SQL – normální spojení + levá/pravá/obě relace dodají n-tice, které na druhé straně spojení nemají partnera (chybějící sloupce musí být doplněny pomocí NULL)
 - **Operace s hodnotou NULL** – všechny datové typy ji mají
 - Pokud alespoň jeden člen porovnání má hodnotu NULL, je výsledek NULL
 - Vyhýbání se NULL hodnotám pomocí operátorů IS NULL, IS NOT NULL, COALESCE
 - **Agregační funkce** – syntaxe: aggregační_funkce ({ALL | DISTINCTS} sloupec | výraz)
 - COUNT, SUM, MAX, MIN, AVG
 - COUNT(A) ignoruje NULL, COUNT(*) započte NULL
 - **Seskupování řádků** – GROUP BY, HAVING (např. HAVING COUNT (herec) > 1)
 - **CASE** – něco jako if:


```
CASE ...
      WHEN ... THEN ...
      ELSE ...
      END
```
 - **LIKE** – dohledávání podobných výrazů pomocí zástupných symbolů (jejich význam ruší \'')

- % – skupina znaků (i prázdná)
- _ – právě jeden znak
- **Vnořené výrazy**
 - IN – množinový predikát


```
SELECT osobni_c, jmeno
FROM Zamestnanci
WHERE plat > ALL (SELECT Z.plat
                     FROM Zamestnanci Z
                     WHERE Z.adresa LIKE '%Praha%');
```
 - ANY, ALL, SOME
 - UNIQUE
 - Kvantifikace – [NOT] EXISTS
 - Nezáleží na tom, co se vybere v klauzuli SELECT vnořeného výrazu
- **Množinové operace** – UNION, INTERSECT, EXCEPT (v Oracle MINUS), UNION ALL, UNION CORRESPONDING
 - Větší vyjadřovací možnosti než RA (agregace, vnější spojení, ...)
 - Příklad – Vypiš seznam všech filmů a u každého uved" počet jeho kopií, včetně filmů bez kopií

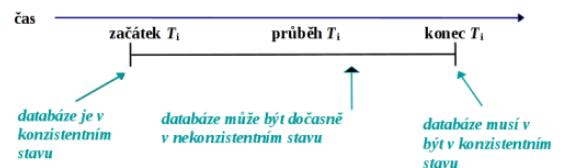

```
SELECT jmeno_f, COUNT (c_kopie) as pocet_kopii
FROM Kopie K RIGHT OUTER JOIN Filmy USING(jmeno_f)
GROUP BY jmeno_f;
```
 - **Normální formy**
 - 1NF – atributy jsou atomické
 - 2NF – žádny neklíčový atribut není závislý na části klíče
 - 3NF – neklíčové atributy závisí na klíci přímo (ne tranzitivně)
 - BCNF – levá strana každé FZ obsahuje klíč schématu
- **Integritní omezení v DDL**
 - Způsob pro zavedení pravidel pro určité sloupce, tabulky...
 - Možnosti vyjádření (a kontroly) IO:
 - Deklarativní:
 - Při vytvoření schématu (relace)
 - Hlídá je automaticky DBMS
 - Výše jsme zavedli PK a FK
 - V SQL – UK, NOT NULL, CHECK
 - IO sloupce: NOT NULL, DEFAULT, UNIQUE, PRIMARY KEY, REFERENCES, CHECK
 - IO tabulky – stejně, jako pro sloupce
 - Procedurální
 - Na straně serveru
 - Na straně klienta
 - TRIGGER
 - **Okamžik kontroly** (DDL)
 - Dají se dočasně vypnout (např. během transakce odstraňování/přidávání dat)
 - Nastavit čas kontroly
 - Odložitelné na konec transakce
 - Zpětné zapnutí IO může/nemusí vyžadovat kontrolu platnosti dat již vložených do DB
 - **Systémový katalog**
 - Metadata – informace o obsahu databáze
 - V relační databázi má podobu pevně daných tabulek, nad nimi obvykle k dispozici pohledy

Transakce a jejich vlastnosti – ACID

BI-DBS

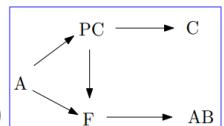
- Transakce

- Požadavky na DMBS:
 - Chránit data – odolnost vůči haváriím serveru
 - Poskytnutí korektního, rychlého a asynchronního přístupu pro větší množství uživatelů najednou
- Modul řízení souběžného zpracování (concurrency control)
 - zajišťuje uživatelům, že každý vidí konzistentní stav DB bez ohledu na to, že ke stejným údajům přistupuje asynchronně více uživatelů najednou
- Modul zotavení z chyb (recovery)
 - zajišťuje, že stav DB není narušen v případě chyby SW, systému nebo fyzického média v průběhu zpracování úlohy
- Transakce
 - sekvence akcí, které spolu logicky souvisí
 - skupina příkazů, která převádí jeden konzistentní stav databáze do jiného konzistentního stavu, přičemž mezistavy nemusí být konzistentní
 - z pohledu DB je to atomický příkaz, provede se buď celý, nebo vůbec.
 - **Transakční zpracování** – dodržení ACID zajistí, že po (jakémkoliv) skončení zůstane DB konzistentní
 - **Konec transakce** – Explicitní (COMMIT x ROLLBACK) x implicitní (ukončení session)
 - **Začátek transakce** – obvykle vymezen skončením transakce předchozí nebo vznikem session
- Stavový diagram transakce
 - A – **aktivní** – od začátku (probíhají DML příkazy)
 - PC – **částečně potvrzený** – po provedení poslední operace transakce
 - C – **potvrzený** – po úspěšném zakončení (tzn. po potvrzení operace COMMIT)
 - F – **chybný** – nelze v normálním průběhu transakce pokračovat
 - AB – **zrušený** – po skončení operace ROLLBACK, uvedený do stavu před započetím transakce



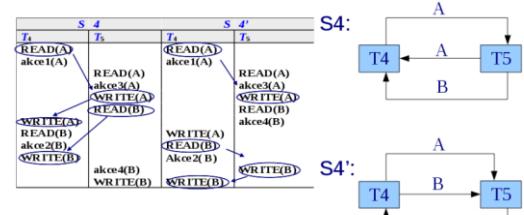
- Vlastnosti transakcí

- ACID
 - **Atomicita (Atomicity)** - transakce musí buď proběhnout celá nebo vůbec
 - **Konzistence (Consistency)** - transakce transf. databázi z konzist. stavu do jiného konzist. stavu
 - **Nezávislost (Independence)** - dílčí efekty jedné transakce nejsou viditelné jiným transakcím
 - **Trvanlivost (Durability)** - efekty úspěšně ukončené (potvrzené) transakce jsou trvale uloženy (persistence)
- Žurnál
 - obnovuje DB po pádu
 - obsahuje sekvenci změnových vektorů a checkpoint
 - zajišťuje Atomicity a Durability
 - operace použité při obnově: UNDO a REDO
- **Zotavení z chyb**
 - **Synchronizační body** (v žurnálu a v hlavičkách db souborů) – slouží k nalezení místa (v žurnálu) odkud je třeba začít s rekonstrukcí databáze
 - Třídy možných chyb:
 - Globální – mají vliv na více transakcí (např. spadnutí systému serveru)
 - Lokální – logické chyby by se měly dát odchytit ROLLBACKem (např. dělení nulou)



- **Rozvrhy**

- Transakce se skládají z dílčích operací $T_1 = \{T_{11}, \dots, T_{1n}\}$ a $T_2 = \{T_{21}, \dots, T_{2m}\}$. Provádět je můžeme celé sériově (např. nejdřív celá T_1 , pak celá T_2), nebo sériově míchat dílčí operace.
- **Rozvrh** = stanovení provádění dílčích akcí více transakcí v čase
- **Rozvrhovač** = se snaží o maximalizaci paralelismu zpracování
- Špatný rozvrh může vést ke ztrátě aktualizace dat (při „prokládání“ transakcí)
- nebezpečí dočasné aktualizace při chybě systému
- **Uspořádatelnost rozvrhu** – rozvrh je uspořádatelný, jestliže existuje sériový rozvrh s ním ekvivalentní
 - Lze vytvořit rozvrh, kde jsou operace navzájem prokládány – **paralelní rozvrh**
 - Rozvrh je **korektní**, když je v nějakém smyslu ekv. kterémukoliv sériovému rozvrhu
 - **Sériové rozvrhy** zachovávají operace každé transakce pohromadě – pro N transakcí existuje $N!$ různých sériových rozvrhů
 - Máme-li rozvrhy S_1 a S_2 pro množinu transakcí $T = T_1, \dots, T_N$, pak S_1 s S_2 jsou ekvivalentní (vzhledem ke konfliktům), jsou-li splněny dvě podmínky:
 - Jestliže se v prvním rozvrhu vyskytuje $\text{READ}(A)$ v T_i a tato hodnota vznikla z $\text{WRITE}(A)$ v T_j , potom totéž musí být zachováno v druhém rozvrhu
 - Jestliže se v prvním rozvrhu vyvolá poslední operace $\text{WRITE}(A)$ v T_i , pak totéž musí být i v druhém rozvrhu
= Relativní pořadí konfliktních operací nad stejnými objekty je v obou rozvrzích stejné
 - **Paralelní zpracování transakcí** – testování uspořádatelnosti
 - Uzamykání (LOCK TABLE)
 - Problémy:
 - Neopakovatelné čtení – T_1 provede SELECT a použije načtené hodnoty, T_2 poté změní hodnoty některých řádků, když T_1 později provede tentýž select, dostane jiné hodnoty
 - Fantóm – jako neopakovatelné čtení, ale T_2 smaže nebo přidá řádky – T_1 pak obdrží jinou sadu dat
- **Stupně izolace** – různé druhy zámků – určují, jestli jsou povoleny anomálie (fantóm, neopakovatelné čtení, dočasná aktualizace)
 - Read uncommitted, read committed (nejčastější), repeatable, serializable (implicitně, všechny a. zakázány)
- **Precedenční graf** – orientovaný graf
 - *vrchol*: jednotlivá transakce
 - *hrana*: konfliktní operace nad stejnými daty, orientace je dána pořadím přístupu k datům v jednotlivých transakcích
 - rozvrh je uspořádatelný, pokud *precedenční graf neobsahuje kružnice* (tzn. obrázek má 2 neuspořádatelné rozvrhy)
 - *dva rozvrhy jsou ekvivalentní, jestliže mají stejný precedenční graf*
- **Uzamykací protokoly** – soubor pravidel pro konstrukci transakce tak, že za určitých předpokladů bude každý její rozvrh uspořádatelný
 - **Jednoduchý model** – Objekt může být v daném čase uzamčen nejvýše jednou transakcí – $\text{Lock}(A)$, $\text{Unlock}(A)$. Transakce, která uzamkla objekt, má právo na něm provádět operace READ a WRITE (a další)



- **Dvoufázový uzamykací protokol**
 - 1. fáze: uzamyká se, nic se neodemyká
 - 2. fáze: od prvního odemknutí se do konce už nic nezamyká
- Uváznutí – deadlock – řešení = ROLLBACK jedné tr. – co ta uzamkla, bude odemknuto – odblok. druhé tr.
- **Legální rozvrh**
 - Objekt bude nutné mít v transakci uzamknutý, kdykoliv k němu chce tato transakce přistupovat
 - Transakce se nebudou pokoušet uzamknout objekt již uzamknutý jinou transakcí
 - Samotná legálnost rozvrhu nezaručuje uspořádatelnost
- **Dobře formované transakce** – podporují přirozené požadavky na transakce
 - Transakce zamyká objekt, chce-li k němu přistupovat
 - Transakce nezamyká objekt, když ho již zamkla
 - Transakce neodmyká objekt, který nezamkla
 - Na konci transakce nezůstane žádný objekt zamčený
- Jestliže všechny transakce v dané množině transakcí T jsou dobře formované a dvoufázové, pak každý jejich legální rozvrh je uspořádatelný

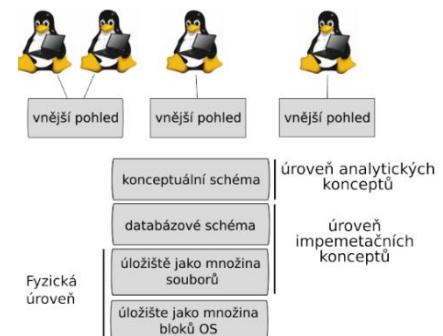
3 úrovně pohledu na data (konceptuální, implementační, fyzická). Struktury pro ukládání dat v relačních databázích s ohledem na rychlý přístup k nim (speciální způsoby uložení, indexy apod.)

BI-DBS

- 3 úrovně pohledu na data

o Konceptuální

- Zabývá se model. reality, komunikace se zákazníkem
- Snaží se nebýt ovlivněna budoucími prostředky řešení
- Používá se grafická notace (např. UML diagram, ER modelem)
- Integrace různých uživatelských pohledů
- Výsledek je vstupem pro realizaci databáze
- Slouží jako dokumentace



o Implementační (databázový/logický)

- Vztahuje se ke konkrétnímu databázového modelu a používá jeho konstrukční dotazovací a manipulační prostředky (relační, objektová, síťová, hierarchická, XML, grafová...)
- Realizace datové struktury popsané v konceptuálním modelu
- Model je transformován do modelu, který odpovídá konkrétní technologii
- Popisuje, čím je datový obsah systému, popsaný koncept. a struktur. modelem, realizován

o Fyzická

- Fyzické uložení dat (sekvenční soubor, indexy, clustery...)
- Uživatelé jsou od ní odstíněni databázovou vrstvou
- Pro přístup se využívá přístupový interface (dotazovací jazyk, API...)

- Oracle ROWID – unikátní identifikátor řádku

- o potřebný pro implementaci indexů – „pseudosloupec“ a „implementační trik“
- o odkazuje na fyzické uložení řádky
- o může se měnit

- Index – pomocná (sekundární) struktura pro rychlejší zpracování požadavku a přístup k datům (řádkům) pomocí klíče indexu

- o Váže se k jednomu nebo k více sloupcům tabulky, nad kterou se často dotazuje
- o Snižuje se rychlosť zápisu, ale zvyšuje rychlosť čtení
- o Různé typy: B-strom, hash i bitmapové tabulky...
- o Jsou nutné pro větší data, bez nich nefunguje DB rozumně
- o Index může být složený/jednoduchý, unikátní/neunikátní
- o Vytváří se automaticky, je třeba je udržovat

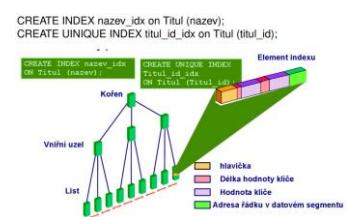
- DSS = decission support system – velká rozhodnutí, založena na historických datech

- OLTP = online transaction processing – aktuální data, každodenní transakce

- o Nejčastěji indexy na bázi B-stromů – tam, kde jsou data (skoro) unikátní
 - Data velmi neunikátní (pohlaví, ...) a potřebuji indexovat – bitmapové indexy

- Typy indexů:

- o **B*-Tree** = vyvážený m -ární strom
 - Data ve vnitřním uzlu jsou organizována
 - Listy obsahují úplnou množinu klíčů a mohou se lišit strukturou
 - Indexy se mohou nacházet ve vnitřních uzlech.
 - Faktor větvení (většinou 100) + hloubka (< 4)
 - Pro OLTP
- o **Bitmapový index** = binární matice
 - narození od B-stromu pro sloupce s nízkou kardinalitou (mohutností)



- výběrové podmínky – binární operátory (AND, OR, NOT) a jejich kombinace
- předvypočítané binární odpovědi pro každý záznam
- DML operace velmi drahé
- spíš pro DSS (ne OLTP)
- vhodné pro záznamy s neunikátními položkami
- **Indexově organizovaná tabulka** – ještě rychlejší přístup přes klíč indexu (ale je jen jeden)
- **Shluk (cluster)** – join tabulek do jednoho shluku podle cluster key
 - Clustered index – max jeden pro tabulku, většinou s váže k primárnímu klíči, fyzicky třídí data a vzniká stromová struktura přímo obsahující údaje
 - Non-clustered index – může existovat více pro tabulku, unikátní klíč, fyzicky samostatný strom, ukazuje na výsledná reálná data s pomocí pointerů
- **Další techniky zrychlení přístupu k datům:**
 - Sledování frekventovaných SQL příkazů a jejich optimalizace (indexy atd.)
 - Speciální struktury pro uložení dat
 - **Heap** (nové záznamy jsou přidány kamkoliv, není uspořádané)
 - Heap s indexy (záznamy jsou uspořádány)
 - Materializované pohledy
 - Partitions (partitioned tables)
 - Horizontální škálování
 - Zavedení redundancy
 - Distribuované databáze
 - Optimalizace aplikace a databáze
 - Možnosti ladění jednoho DB serveru:
 - Systémové zdroje (paměť – velikost a struktura)
 - Konfigurace serveru (disky, parametry databáze, zálohování)

Soustavy lineárních rovnic: Frobeniova věta a související pojmy, vlastnosti a popis množiny řešení, Gaussova eliminační metoda

BI-LIN

- Základní pojmy lineární algebry:

- **Vektorový prostor** – nechť T je libovolné komutativní těleso, jeho neutrální prvky vůči operacím sčítání, resp. násobení označme 0 , resp. 1 . Nechť je dále dána neprázdná množina V a dvě zobrazení

$$\oplus : V \times V \rightarrow V, \quad \odot : T \times V \rightarrow V.$$

Řekneme, že V je **vektorový prostor nad tělesem** T s vektorovými operacemi \oplus a \odot , právě když platí následující **axiomy vektorového prostoru**:

1. $\forall a, b \in V : a \oplus b = b \oplus a,$
2. $\forall a, b, c \in V : (a \oplus b) \oplus c = a \oplus (b \oplus c),$
3. $\forall \alpha, \beta \in T, \forall a \in V : \alpha \odot (\beta \odot a) = (\alpha\beta) \odot a,$
4. $\forall \alpha \in T, \forall a, b \in V : \alpha \odot (a \oplus b) = (\alpha \odot a) \oplus (\alpha \odot b),$
5. $\forall \alpha, \beta \in T, \forall a \in V : (\alpha + \beta) \odot a = (\alpha \odot a) \oplus (\beta \odot a),$
6. $\forall a \in V : 1 \odot a = a,$
7. $\exists \theta \in V, \forall a \in V : \theta \odot a = \theta.$

Prvky vektorového prostoru nazýváme **vektory**, prvky tělesa T nazýváme **skaláry** a prvek θ z axiomu 7 nazýváme **nulový vektor**.

- **Podprostor** – Nechť V je vektorový prostor nad tělesem T a nechť $\emptyset \neq P \subseteq V$ (P je neprázdná podmnožina V). Říkáme, že P je **podprostor** prostoru V , právě když platí:

1. $\forall x, y \in P : x + y \in P,$
2. $\forall \alpha \in T, \forall x \in P : \alpha x \in P$

(tedy P je množina uzavřená na obě vektorové operace $+$, \cdot). Vztah „být podprostorem“ pak značíme $P \subset\subset V$.

- Nechť V je vektorový prostor nad tělesem T , nechť $P \subset\subset V$. Potom P se zúžením operace sčítání vektorů $+$ na $P \times P$ a operace násobení vektorů skalárem \cdot na $T \times P$ je také vektorový prostor nad T .

- **Lineární kombinace** – Nechť V je vektorový prostor nad T , $x \in V$ a (x_1, \dots, x_n) je soubor vektorů z V . Říkáme, že vektor x je **lineární kombinací** souboru (x_1, \dots, x_n) právě když existují čísla $\alpha_1, \dots, \alpha_n \in T$ taková, že

$$x = \sum_{i=1}^n \alpha_i x_i.$$

Čísla $\alpha_i, i \in \hat{n}$, nazýváme **koefficienty lineární kombinace**. Jestliže $\forall i \in \hat{n} : \alpha_i = 0$, nazýváme takovou lineární kombinaci **triviální**. V opačném případě jde o lineární kombinaci **netriviální**.

- **Lineární nezávislost** – Nechť (x_1, \dots, x_n) je soubor vektorů z V . Řekneme, že (x_1, \dots, x_n) je **lineárně nezávislý** (LN) soubor, právě když pouze triviální lineární kombinace tohoto souboru je rovna nulovému vektoru θ . V opačném případě nazýváme soubor **lineárně závislý** (LZ). Jinými slovy:

- (x_1, \dots, x_n) je LN \Leftrightarrow

$$\forall \alpha_1, \dots, \alpha_n \in T : \left(\sum_{i=1}^n \alpha_i x_i = \theta \Rightarrow (\forall i \in \hat{n})(\alpha_i = 0) \right)$$

- (x_1, \dots, x_n) je LZ \Leftrightarrow

$$\exists \alpha_1, \dots, \alpha_n \in T, \exists k \in \hat{n}, \alpha_k \neq 0 : \left(\sum_{i=1}^n \alpha_i x_i = \theta \right)$$

- **Lineární obal** – Bud' (x_1, \dots, x_n) soubor vektorů z V . Množinu všech lineárních kombinací tohoto souboru nazveme **lineárním obalem souboru** (x_1, \dots, x_n) a značíme ji $\langle x_1, \dots, x_n \rangle$. Bud' $\emptyset \neq M \subseteq V$. Množinu všech lineárních kombinací všech souborů vektorů z množiny M nazveme **lineárním obalem množiny** M a značíme ji $\langle M \rangle$.

- O množině vektorů M z vektorového prostoru V řekneme, že **generuje prostor V** , právě když platí: $\langle M \rangle = V$.
- **Báze VP** – Existuje-li ve V uspořádaná množina vektorů B taková, že
 - i. B je LN,
 - ii. B generuje V ,
 nazýváme B **bází vektorového prostoru V** .
- **Dimenze VP** – Bud' V vektorový prostor nad T . Řekneme, že **dimenze vektorového prostoru V** je rovna
 - 0, pokud ve V neexistuje LN soubor délky 1.
 - $n \in \mathbb{N}$, pokud ve V existuje LN soubor délky n , ale každý soubor délky $n+1$ už je nutně LZ.
 - ∞ , pokud ve V existuje LN soubor libovolné délky.

Dimenzi vektorového prostoru V označujeme symbolem $\dim V$. Je-li $\dim V = \infty$, říkáme, že V má **nekonečnou dimenzi**, naopak pokud $\dim V < \infty$ říkáme, že V má **konečnou dimenzi**.

- **Souřadnice vektoru v bázi** – Nechť $\mathcal{X} = (x_1, \dots, x_n)$ je báze V_n . Potom ke každému $z \in V_n$ existuje právě jedna uspořádaná n-tice $(\alpha_1, \dots, \alpha_n) \in T^n$ taková, že

$$z = \sum_{i=1}^n \alpha_i x_i.$$

Souřadnicemi vektoru $z \in V_n$ v bázi \mathcal{X} nazveme uspořádanou n-tici (sloupcový vektor)

$$(z)_{\mathcal{X}} := \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \in T^{n,1}.$$

Číslo $\alpha_i \in T$ je itá souřadnice vektoru z v bázi \mathcal{X} , často značíme

$$x_i^\#(z) := \alpha_i.$$

- Nechť $a, b \in \mathbb{R}$ a $a \neq 0$. Potom $x = a^{-1}b$ je jediné reálné číslo splňující rovnici $ax = b$.
- **Geometrická interpretace množiny řešení** – rovina v prostoru $R \mathbb{R}^3; x, y, z \in \mathbb{R}$:

$$ax + by + cz = d, \quad a, b, c, d \in \mathbb{R}$$
- **Soustava lineárních rovnic** – nechť n a m jsou přirozená čísla a pro všechna $i \in \{1, \dots, m\}, j \in \{1, \dots, n\}$ platí, že $a_{ij} \in \mathbb{R}, b_i \in \mathbb{R}$. Systém rovnic

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & = & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}$$

nazýváme **soustavou m lineárních rovnic o n neznámých** $x_1, \dots, x_n \in \mathbb{R}$. Číslu a_{ij} říkáme j -tý koeficient i-té rovnice.

- Množinu všech uspořádaných n-tic $(x_1, \dots, x_n) \in \mathbb{R}^n$, pro které po dosazení do rovnic je splněno všechny m rovnic, nazýváme **množinou řešení** soustavy a značíme ji S .
- Platí-li $b_1 = \dots = b_m = 0$, říkáme, že je soustava homogenní. Není-li soustava **homogenní**, je **nehomogenní**.
- Množina řešení soustavy je rovna průniku množin řešení jednotlivých rovnic.
- Homogenní SLR má vždy alespoň jedno řešení $(0, \dots, 0) \in \mathbb{R}^n$, to zároveň nikdy není řešením nehomogenní SLR.
- Má-li homogenní SLR i jiné řešení než $(0, \dots, 0) \in \mathbb{R}^n$, pak má nekonečně mnoho řešení.

- **Úpravy SLR:**

- Prohození dvou rovnic
- Vynásobení jedné rovnice nenulovým číslem
- Přičtení libovolného násobku jedné rovnice k jiné rovnici

- Maticový zápis SLR

Nechť $m, n \in \mathbb{N}$, $\mathbb{A} \in \mathbb{R}^{m,n}$, $\mathbb{b} \in \mathbb{R}^m$. Rovnici

$$\mathbb{A}\mathbf{x} = \mathbb{b} \quad (7)$$

nazýváme **soustavou m lineárních rovnic pro n neznámých** x_1, x_2, \dots, x_n . Vektor $\mathbf{x} \in \mathbb{R}^{n,1}$, kde $\mathbf{x}^T = (x_1 \ x_2 \ \dots \ x_n)$ nazýváme **vektorem neznámých** a vektor $\mathbb{b} \in \mathbb{R}^{m,1}$, kde $\mathbb{b}^T = (b_1 \ b_2 \ \dots \ b_m)$ vektorem **pravých stran**. Matici \mathbb{A} nazýváme **maticí soustavy** a matici

$$(\mathbb{A} \mid \mathbb{b}) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array} \right)$$

rozšířenou maticí soustavy. Je-li $\mathbb{b} = \theta \in \mathbb{R}^m$, mluvíme o **homogenní soustavě**. Soustava $\mathbb{A}\mathbf{x} = \theta$ je **přidruženou homogenní soustavou lineárních rovnic k soustavě** $\mathbb{A}\mathbf{x} = \mathbb{b}$.

Množinu všech řešení značíme S (pro nehomogenní) nebo S_0 (pro homogenní).

Nechť $\tilde{\mathbf{x}}$ je partikulární řešení soustavy $\mathbb{A}\mathbf{x} = \mathbb{b}$, potom pro soustavu platí, že $S = \tilde{\mathbf{x}} + S_0$.

- Hodnost matice – Nechť $\mathbb{A} \in T^{m,n}$. Hodností matice \mathbb{A} nazýváme **dimenzi lineárního obalu** souboru řádků matice \mathbb{A} (jako vektorů z $T^{1,n}$) a značíme ji $h(\mathbb{A})$. Tedy:

$$h(\mathbb{A}) = \dim(\mathbb{A}_{1:}, \dots, \mathbb{A}_{m:}).$$

- $h(\mathbb{A}) \leq m$ pro každou $\mathbb{A} \in T^{m,n}$
- Nechť $\mathbb{A} \in T^{m,n}$ je matice v **horním stupňovitém tvaru** s právě k nenulovými řádky. Pak $h(\mathbb{A}) = k$.
- $h(\mathbb{A}) = h(\mathbb{A}^T)$

- Frobeniova věta

Nechť $\mathbb{A} \in T^{m,n}$.

- ➊ **Soustava $\mathbb{A}\mathbf{x} = \mathbb{b}$ je řešitelná, tj. $S \neq \emptyset$, právě tehdy, když**

$$h(\mathbb{A}) = h(\mathbb{A} \mid \mathbb{b}).$$

- ➋ **Je-li $h(\mathbb{A}) = h$, pak množina řešení soustavy $\mathbb{A}\mathbf{x} = \theta$ je podprostor dimenze $n - h$, tedy existuje LN soubor vektorů $(\mathbf{z}_1, \dots, \mathbf{z}_{n-h})$ v $T^{n,1}$ takový, že**

$$S_0 = \begin{cases} \{\theta\}, & \text{pokud } n = h, \\ \langle \mathbf{z}_1, \dots, \mathbf{z}_{n-h} \rangle, & \text{pokud } h < n. \end{cases}$$

Je-li navíc $h(\mathbb{A} \mid \mathbb{b}) = h$, pak $S = \tilde{\mathbf{x}} + S_0$, kde $\tilde{\mathbf{x}}$ je tzv. **partikulární řešení: $\mathbb{A}\tilde{\mathbf{x}} = \mathbb{b}$.**

- Vlastnosti a popis množiny řešení SLR

- **Horní stupňovitý tvar matice SLR**

O matici $\mathbb{D} \in \mathbb{R}^{m,n}$ řekneme, že je v **horním stupňovitém tvaru**, jestliže všechny řádky jsou nulové, nebo existuje $k \in \hat{n}$ tak, že řádky 1 až k matice \mathbb{D} jsou nenulové a řádky $k+1$ až m jsou nulové a jestliže platí následující:
Označme pro každé $i \in \hat{k}$ index nejlevějšího nenulového prvku v itém řádku jako j_i , tj.

$$j_i = \min\{\ell \in \hat{n} \mid \mathbb{D}_{i\ell} \neq 0\}.$$

Potom platí $1 \leq j_1 < j_2 < \dots < j_k$.

Je-li matice v horním stupňovitém tvaru, potom sloupcům s indexy j_1, j_2, \dots, j_k říkáme **hlavní sloupce**, ostatním říkáme **vedlejší sloupce**.

O soustavě $\mathbb{A}\mathbf{x} = \mathbb{b}$ řekneme, že je v horním stupňovitém tvaru, pokud matice této soustavy $(\mathbb{A} \mid \mathbb{b})$ je v horním stupňovitém tvaru.

- **Počet řešení**

Mějme soustavu lineárních rovnic $\mathbb{A}\mathbf{x} = \mathbb{b}$, kde $\mathbb{A} \in \mathbb{R}^{m,n}$. Je-li tato soustava v horním stupňovitém tvaru, platí následující:

- ➊ Je-li poslední sloupec matice $(\mathbb{A} \mid \mathbb{b})$ hlavní, soustava nemá řešení.
- ➋ Je-li poslední sloupec matice $(\mathbb{A} \mid \mathbb{b})$ jediný vedlejší sloupec, má soustava právě jedno řešení.
- ➌ Je-li poslední sloupec matice $(\mathbb{A} \mid \mathbb{b})$ vedlejší a existuje-li ještě jiný vedlejší sloupec, má soustava více než jedno řešení.

- Postup řešení SLR

Řešíme soustavu $\mathbb{A}\mathbf{x} = \mathbf{b}$ s rozšířenou maticí $(\mathbb{A} \mid \mathbf{b})$, kde $\mathbb{A} \in T^{m,n}$ je v HST (na něj lze vždy převést pomocí GEM).

- Pokud tím neporušíme HST, můžeme prohodit pořadí sloupců v matici \mathbb{A} (stejně prohodíme i příslušné proměnné).
- Pokud $h(\mathbb{A}) < h(\mathbb{A} \mid \mathbf{b})$, řešení neexistuje. Jinak postupujeme dále.
- Za vázané proměnné označíme proměnné příslušející hlavním sloupcům matice. Zbývající volné proměnné označíme (t_1, \dots, t_{n-h}) .
- Pro nalezení $\tilde{\mathbf{x}}$ zvolme za (t_1, \dots, t_{n-h}) libovolná a ze soustavy $(\mathbb{A} \mid \mathbf{b})$ dopočítajme vázané proměnné.
- Pro nalezení S_0 zvolíme libovolnou bázi T^{n-h} (různé volby volných proměnných). **Vynulujeme** pravou stranu soustavy.
- Pro každý bazický vektor $(t_1, \dots, t_{n-h}) \in T^{n-h}$ reprezentující volbu volných proměnných dopočítáme z $(\mathbb{A} \mid \theta)$ vázané proměnné a dostáváme bázi S_0 .
- Řešením je $S = \tilde{\mathbf{x}} + S_0$.

- Gaussova eliminační metoda – vychází z Úprav SLR, cíl = převést matici do horního stupňovitého tvaru

(G1) Prohození dvou řádků.

(G2) Vynásobení jednoho řádku matice nenulovým číslem, přesněji nahrazení řádku

$$\begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{pmatrix}$$

řádkem

$$\begin{pmatrix} \alpha a_{i1} & \alpha a_{i2} & \cdots & \alpha a_{in} \end{pmatrix},$$

pro nějaké $1 \leq i \leq m$ a $\alpha \in \mathbb{R} \setminus \{0\}$.

(G3) Přičtení libovolného násobku jednoho řádku k jinému, přesněji nahrazení řádku

$$\begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{pmatrix}$$

řádkem

$$\begin{pmatrix} (a_{i1} + \alpha a_{j1}) & (a_{i2} + \alpha a_{j2}) & \cdots & (a_{in} + \alpha a_{jn}) \end{pmatrix},$$

pro nějaká různá čísla $i, j \in \{1, 2, \dots, m\}$ a $\alpha \in \mathbb{R}$.

- Přivedeme-li rozšířenou matici jedné soustavy na rozšířenou matici jiné pomocí jedné z úprav (G1), (G2) nebo (G3), mají obě soustavy stejnou množinu řešení

Matice: součin matic, regulární matice, inverzní matice a její výpočet, vlastní čísla matice a jejich výpočet, diagonalizace matice

BI-LIN

- **Matice** – nechť $m, n \in \mathbb{N}$. Uspořádaný soubor mn čísel zapsaný do tabulky o m řádcích a n sloupcích nazýváme **matice** typu $m \times n$. Matice obvykle značíme takto:

$$\mathbb{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

kde a_{ij} jsou **prvky matice** (někdy značíme \mathbb{A}_{ij} a nazýváme **jj-té prvky**). Číslu i říkáme **řádkový** a číslu j **sloupcový index**.

- Množinu všech matic typu $m \times n$ (s reálnými prvky) značíme $\mathbb{R}^{m,n}$.
- Jako $\mathbb{A}_{\cdot j} \in \mathbb{R}^{m,1}$ značíme j-tý sloupec, podobně $\mathbb{A}_{i\cdot} \in \mathbb{R}^{1,n}$ značíme i-tý řádek matice \mathbb{A} .
- **Násobení matice číslem a součet matic** – buděte $m, n \in \mathbb{N}; \alpha \in \mathbb{R}; \mathbb{A}, \mathbb{B} \in \mathbb{R}^{m,n}$ matice s prvky a_{ij} , resp. b_{ij} .

Součin matice \mathbb{A} a reálného čísla α definujeme:

$$\alpha\mathbb{A} := \begin{pmatrix} \alpha a_{11} & \alpha a_{12} & \cdots & \alpha a_{1n} \\ \alpha a_{21} & \alpha a_{22} & \cdots & \alpha a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{m1} & \alpha a_{m2} & \cdots & \alpha a_{mn} \end{pmatrix}$$

Součet matic \mathbb{A}, \mathbb{B} , definujeme:

$$\mathbb{A} + \mathbb{B} := \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{pmatrix}$$

Matici $\mathbb{A} + (-1)\mathbb{B}$ nazýváme rozdíl matic \mathbb{A}, \mathbb{B} a značíme $\mathbb{A} - \mathbb{B}$, matici $(-1)\mathbb{A}$ značíme jako $-\mathbb{A}$.

- **Součin matic** – buděte $m, n, p \in \mathbb{N}; \mathbb{A} \in \mathbb{R}^{m,n}$ matice s prvky a_{ij} a $\mathbb{B} \in \mathbb{R}^{n,p}$ matice s prvky b_{ij} . Součinem matic \mathbb{A}, \mathbb{B} je matice $\mathbb{D} \in \mathbb{R}^{m,p}$ s prvky d_{ij} , pro kterou platí

$$d_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Značíme $\mathbb{D} = \mathbb{A}\mathbb{B}$.

- Podmínka – počet **sloupců** matice \mathbb{A} = počet **řádků** \mathbb{B} (násobí se řádky x sloupce)
 - Záleží na pořadí (není komutativní) a velikosti
 - Nechť $m, n, s, t \in \mathbb{N}$. Pro libovolné matici $\mathbb{A} \in \mathbb{R}^{m,n}, \mathbb{B} \in \mathbb{R}^{n,s}, \mathbb{D} \in \mathbb{R}^{s,t}$ platí
- $$\mathbb{A}(\mathbb{B}\mathbb{D}) = (\mathbb{A}\mathbb{B})\mathbb{D}$$
- V následujících tvrzeních jsou rozměry matic \mathbb{A}, \mathbb{B} a \mathbb{D} vždy takové, aby obsažené výrazy měly smysl a $\alpha \in \mathbb{R}$. Platí:
 - Distributivní zákon – $\mathbb{A}(\mathbb{B} + \mathbb{D}) = \mathbb{A}\mathbb{B} + \mathbb{A}\mathbb{D}$
 - Distributivní zákon – $(\mathbb{A} + \mathbb{B})\mathbb{D} = \mathbb{A}\mathbb{D} + \mathbb{B}\mathbb{D}$
 - $\alpha(\mathbb{A}\mathbb{B}) = (\alpha\mathbb{A})\mathbb{B} = \mathbb{A}(\alpha\mathbb{B})$
 - $(\mathbb{A}\mathbb{B})^T = \mathbb{B}^T\mathbb{A}^T$

- **Transponovaná matice** – buděte $m, n \in \mathbb{N}$ a $\mathbb{A} \in \mathbb{R}^{m,n}$ matice s prvky a_{ij} . **Transpozicí matice** \mathbb{A} nazýváme matici z $\mathbb{R}^{n,m}$, jejíž prvek v jtém řádku a itém sloupci je roven a_{ij} . Tuto matici značíme \mathbb{A}^T .
- **Regulární a inverzní matice** – bud' $\mathbb{A} \in \mathbb{T}^{n,n}$. Existuje-li matice $\mathbb{B} \in \mathbb{T}^{n,n}$ taková, že platí

$$\mathbb{A}\mathbb{B} = \mathbb{B}\mathbb{A} = \mathbb{E}$$

nazýváme matici \mathbb{A} **regulární** a \mathbb{B} **inverzní maticí** k matici \mathbb{A} . Značíme $\mathbb{B} = \mathbb{A}^{-1}$.

- Pokud \mathbb{A} není regulární, nazýváme matici \mathbb{A} **singulární**.
- Matice \mathbb{E} je jednotková čtvercová matice (tzn. leží v $\mathbb{T}^{n,n}$ která má na diagonále 1, jinak 0)
- Pokud je A regulární, tak je inverze určená jednoznačně

- **Výpočet inverzní matice** – buď $\mathbb{A} \in T^{n,n}$. Následující tvrzení jsou ekvivalentní:
 - \mathbb{A} je regulární.
 - Soubor řádků matice \mathbb{A} je LN.
 - $h(\mathbb{A}) = n$
 - $\mathbb{A} \sim \mathbb{E}$
- **Algoritmus ověření regularity a nalezení inverzní matice** – nechť $\mathbb{A} \in T^{n,n}$. Ověřte, zda je matice regulární a pokud ano, nalezněte k ní matici inverzní \mathbb{A}^{-1} .
 - Hledáme matici \mathbb{A}^{-1} s vlastností $\mathbb{A}^{-1}\mathbb{A} = \mathbb{A}\mathbb{A}^{-1} = \mathbb{E}$
 - Doplňním zadané matice o jednotkovou matici stejného rozměru sestavme dvoublokovou rozšířenou matici $(\mathbb{A} | \mathbb{E}) \in T^{n,2n}$.
 - Na celou $(\mathbb{A} | \mathbb{E})$ používáme řádkové úpravy GEM, pro libovolnou posloupnost řádkových úprav \mathbb{P} pak platí:
$$(\mathbb{A} | \mathbb{E}) \sim (\mathbb{P}\mathbb{A} | \mathbb{P}\mathbb{E}) = (\mathbb{P}\mathbb{A} | \mathbb{P})$$

\mathbb{A} je možné převést na jednotkovou matici právě tehdy, když je \mathbb{A} regulární.

 - Je-li \mathbb{A} regulární, pak pro úpravy \mathbb{P} vedoucí k převedení levého bloku matice $(\mathbb{A} | \mathbb{E})$ na jednotkovou matici platí $\mathbb{P} = \mathbb{A}^{-1}$, tedy
$$(\mathbb{A} | \mathbb{E}) \sim (\mathbb{E} | \mathbb{A}^{-1})$$

$$\left(\begin{array}{cc|cc} 1 & 0 & -2 & 1 \\ 0 & 1 & \frac{3}{2} & -\frac{1}{2} \end{array} \right) \Rightarrow \left(\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 3 & 4 & 0 & 1 \end{array} \right)$$

- **Permutace** – buď $n \in \mathbb{N}$. Každé zobrazení $\pi: \hat{n} \rightarrow \hat{n}$, které je bijekcí, nazýváme permutací množiny \hat{n} . Množinu všech permutací množiny \hat{n} značíme S_n .
 - Celkem $n!$ Permutací
- **Inverze a znaménko permutace**
 - V permutaci $\pi = (3, 1, 5, 2, 4)$ existují 4 inverze: za číslem 3 se vyskytují menší čísla 1 a 2 a za číslem 5 se vyskytují menší čísla 2 a 4. Platí tedy, že $\text{sgn}\pi = (-1)^4 = 1$.
- **Determinant matice** – buď $\mathbb{A} \in T^{n,n}$ se složkami $\mathbb{A}_{ij} = a_{i,j}$. Determinant matice \mathbb{A} je číslo definované vztahem

$$\det \mathbb{A} = \sum_{\pi \in S_n} \text{sgn}\pi a_{1,\pi(1)} a_{2,\pi(2)} \dots a_{n,\pi(n)}.$$

- Pro dané π si sčítanec $a_{1,\pi(1)} a_{2,\pi(2)} \dots a_{n,\pi(n)}$ můžeme představit tak, že v každém řádku $i = 1, 2, \dots, n$ zvolíme prvek, který je v $\pi(i)$ tém sloupci a všechny tyto prvky vynásobíme.

$$\begin{array}{c} \left(\begin{array}{ccc} \color{red}{a} & b & c \\ d & \color{red}{e} & f \\ g & h & \color{red}{j} \end{array} \right) \quad \left(\begin{array}{ccc} a & b & \color{red}{c} \\ \color{red}{d} & e & f \\ g & \color{red}{h} & j \end{array} \right) \quad \left(\begin{array}{ccc} a & \color{red}{b} & c \\ d & e & \color{red}{f} \\ \color{red}{g} & h & j \end{array} \right) \\ \downarrow \qquad \downarrow \qquad \downarrow \\ \det \left(\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & j \end{array} \right) = \quad \color{red}{aej} \quad + \quad \color{red}{dhc} \quad + \quad \color{red}{gbf} \\ - \quad \color{blue}{gec} \quad - \quad \color{blue}{haf} \quad - \quad \color{blue}{dbj} \\ \uparrow \qquad \uparrow \qquad \uparrow \\ \left(\begin{array}{ccc} \color{blue}{a} & b & \color{red}{c} \\ d & \color{blue}{e} & f \\ \color{blue}{g} & h & j \end{array} \right) \quad \left(\begin{array}{ccc} a & b & \color{red}{c} \\ d & e & \color{blue}{f} \\ g & \color{blue}{h} & j \end{array} \right) \quad \left(\begin{array}{ccc} a & \color{blue}{b} & c \\ \color{blue}{d} & e & f \\ g & h & \color{red}{j} \end{array} \right) \end{array}$$

- **Vliv sloupcových a řádkových úprav na determinant** – buď $\mathbb{A} \in T^{n,n}$.
 - Buď \mathbb{B} matice, která vznikne z matice \mathbb{A} prohozením itého a jtého sloupce (nebo řádku), $i \neq j$. Potom
$$\det \mathbb{B} = -\det \mathbb{A}$$
- Buď \mathbb{B} matice, která vznikne z matice \mathbb{A} vynásobením itého řádku číslem $\alpha \in T$, potom:
$$\det \mathbb{B} = \alpha \det \mathbb{A}$$
- Budě \mathbb{B} a \mathbb{D} matice, které mají shodné prvky s maticí \mathbb{A} až na itý řádek, pro který platí $\mathbb{A}_{i,:} = \mathbb{B}_{i,:} + \mathbb{D}_{i,:}$, potom:

$$\det \mathbb{B} + \det \mathbb{D} = \det \mathbb{A}$$

- **Determinant transponované matice** – pro matici $\mathbb{A} \in T^{n,n}$ platí: $\det \mathbb{A} = \det \mathbb{A}^T$
- **Vlastnosti determinantu** – následující tvrzení jsou pravdivá:
 - i. Obsahuje-li matice dva stejné řádky, pak je její determinant nulový.
 - ii. Bud' \mathbb{B} matice, která má všechny řádky stejné jako \mathbb{A} s výjimkou i-tého řádku, kde platí $\mathbb{B}_{i,:} = \mathbb{A}_{i,:} + \alpha \mathbb{A}_{j,:}$ pro nějaké $\alpha \in T$, potom
$$\det \mathbb{B} = \det \mathbb{A}$$

Speciálně, přičteme-li k nějakému řádku matice jiný řádek té samé matice, pak se determinant matice nezmění.

 - iii. **Kroky GEM** mohou měnit hodnotu i znaménko determinantu, ale zachovávají nenulovost.
Platí-li $\mathbb{A} \sim \mathbb{B}$, pak $\det \mathbb{A} \neq 0$, právě když $\det \mathbb{B} \neq 0$.
- Matice $\mathbb{A} \in T^{n,n}$ je regulární, právě když má nenulový determinant.
- Pro matice $\mathbb{A}, \mathbb{B} \in T^{n,n}$ platí: $\det(\mathbb{A}\mathbb{B}) = \det \mathbb{A} \cdot \det \mathbb{B}$.
 - Pro každou matici $\mathbb{D} \in T^{n,n}$ a libovolnou matici \mathbb{P} reprezentující elementární krok GEM platí:
$$\det(\mathbb{P}\mathbb{D}) = \det \mathbb{P} \cdot \det \mathbb{D}$$
- **Cramerovo pravidlo** – nechť $\mathbb{A} \in \mathbb{R}^n$ je regulární matice a $\mathbb{b} \in T^n$. Potom řešení soustavy $\mathbb{A}\mathbf{x} = \mathbb{b}$, kde $\mathbf{x} = (x_1, \dots, x_n)$ splňuje

$$x_i = \frac{\det \mathbb{A}_i}{\det \mathbb{A}}$$

kde matice \mathbb{A}_i vznikla z matice \mathbb{A} nahrazením itého sloupce vektorem \mathbb{b} , neboli

$$\mathbb{A}_i = \begin{pmatrix} \mathbb{A}_{11} & \mathbb{A}_{12} & \dots & \mathbb{A}_{1,i-1} & \mathbb{b}_1 & \mathbb{A}_{1,i+1} & \dots & \mathbb{A}_{1n} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \mathbb{A}_{n1} & \mathbb{A}_{n2} & \dots & \mathbb{A}_{n,i-1} & \mathbb{b}_n & \mathbb{A}_{n,i+1} & \dots & \mathbb{A}_{nn} \end{pmatrix}$$

- Vlastní čísla

- **Operátor** – buď V vektorový prostor. Zobrazení $E: V \rightarrow V$ definované vztahem

$$\forall x \in V: Ex = x$$

Je lineární operátor a nazýváme ho identický operátor na V . Izomorfismem nazveme jakékoli zobrazení $A \in \mathcal{L}(P, Q)$, které je bijekce.
- **Vlastní číslo operátoru** – řekneme, že $\lambda \in \mathbb{C}$ je **vlastní číslo operátoru** $A \in \mathcal{L}(V)$ právě když existuje $x \in V, x \neq \theta$, takový, že $Ax = \lambda x$. Vektor x pak nazýváme **vlastním vektorem operátoru** A příslušejícím vlastnímu číslu λ . Množinu všech vlastních čísel A nazýváme **spektrem** A a značíme $\sigma(A)$.
 - Každý operátor má alespoň jedno vlastní číslo
 - Intuitivně – $Ax = \lambda x$, vlastní číslo operátoru (lineárního zobrazení do stejného VP = matice přechodu) lambda „nahrazuje“ matici jedním číslem. K němu patří vlastní vektor, který musí být **nutně nenulový**.
- **Vlastní číslo matice** – řekneme, že $\lambda \in \mathbb{C}$ je vlastní číslo matice $\mathbb{A} \in \mathbb{C}^{n,n}$, právě když existuje $\mathbf{x} \in \mathbb{C}^n, \mathbf{x} \neq \theta$, takový, že $\mathbb{A}\mathbf{x} = \lambda \mathbf{x}$. Vektor \mathbf{x} pak nazýváme vlastním vektorem matice \mathbb{A} příslušejícím vlastnímu číslu λ . Množinu všech vlastních čísel \mathbb{A} nazýváme spektrem \mathbb{A} a značíme $\sigma(\mathbb{A})$.
- **Charakteristický polynom** matice \mathbb{A} (ozn. $p_{\mathbb{A}}$) definujeme předpisem $p_{\mathbb{A}}(\lambda) := \det(\mathbb{A} - \lambda \mathbb{E})$.
 - Nechť $A \in \mathcal{L}(V_n), \lambda_0 \in \sigma(A)$. Násobnost čísla λ_0 jako kořene charakteristického polynomu p_A operátoru A nazýváme **algebraickou násobností** vlastního čísla λ_0 a značíme ji $\nu_a(\lambda_0)$.
 - Podprostor $\ker(A - \lambda_0 E)$ nazýváme **vlastním podprostorem operátoru** A příslušejícím **vlastnímu číslu** λ_0 .
 - Číslo $d(A - \lambda_0 E)$ nazýváme **geometrickou násobností** vlastního čísla λ_0 a značíme ji $\nu_g(\lambda_0)$
 - $\nu_g(\lambda_0) \leq \nu_a(\lambda_0)$
 - $\nu_g(\lambda_0)$ je tedy maximální délka LN souboru vlastních vektorů k vlastnímu číslu λ_0
- **Výpočet vlastních čísel** – Pro danou matici $\mathbb{A} \in \mathbb{C}^{n,n}$, hledáme nenulové vektory \mathbf{x} a čísla $\lambda \in \mathbb{C}$ splňující rovnici $\mathbb{A}\mathbf{x} = \lambda \mathbf{x}$.
 - To je ekvivalentní hledání λ takové, že homogenní soustava rovnic $(\mathbb{A} - \lambda \mathbb{E})\mathbf{x} = \theta$ má **nenulové řešení**.

- To nastává ale tehdys a jen tehdys (Frobeniova věta), když je matice $\mathbb{A} - \lambda\mathbb{E}$ singulární (neregulární).
- To je zase ekvivalentní tomu, že **determinant matice $\mathbb{A} - \lambda\mathbb{E}$ je roven nule** – abychom tedy našli vlastní číslo, řešíme rovnici

$$\det(\mathbb{A} - \lambda\mathbb{E}) = 0$$

- Pro zadané vlastní číslo λ už najdeme vlastní vektory snadno jako řešení homogenní soustavy uvedené výše
- Intuitivně – máme matici. Od její diagonály odečteme λ (od matice odečítáme $\lambda\mathbb{E}$). Z této matice spočítáme determinant. Determinant bude rovnice s λ proměnnými, která je rovna nule. To je charakteristický polynom. Hledáme nulové λ kořeny. Máme vlastní čísla.

- **Diagonalizace matice**

- **Podobné matice** – značí se $\mathbb{A} \sim \mathbb{B}$ – matice $\mathbb{A}, \mathbb{B} \in \mathbb{C}^{n,n}$ nazveme podobné, právě když existuje $\mathbb{P} \in \mathbb{C}^{n,n}$ regulární tak, že platí

$$\mathbb{A} = \mathbb{P}^{-1}\mathbb{B}\mathbb{P}$$

- Podobné matice mají stejné charakteristické polynomy a spektra

- **Diagonalizace**

- **Diagonální matice** = čtvercová matice, která má nenulové prvky pouze na diagonále
- **Báze** = množina vektorů, která je LN a generuje celý vektorový prostor
- **Operátor $A \in \mathcal{L}(V_n)$** je diagonalizovatelný, právě když každé z vlastních čísel λ operátoru A má stejnou algebraickou a geometrickou násobnost, tj.

$$(\forall \lambda \in \sigma(A))(\nu_a(\lambda) = \nu_g(\lambda)).$$

- *Operátor $A \in \mathcal{L}(V_n)$ nazveme **diagonalizovatelný**, jestliže existuje báze \mathcal{X} prostoru V_n taková, že matice ${}^{\mathcal{X}}A$ je diagonální. Matici $\mathbb{A} \in \mathbb{C}^{n,n}$ nazveme **diagonalizovatelnou**, jestliže je podobná diagonální matici.*

Výroková logika: splnitelnost formulí, logická ekvivalence a důsledek, universální systém logických spojek, disjunktivní a konjunktivní normální tvary, úplné normální tvary.

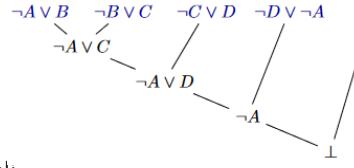
BI-MLO

- **Prvotní výrok** – jednoduchá oznamovací věta, u které má smysl se ptát, zda je či není pravdivá
 - o **Prvotní formule** – prvotní výroky označeny A, B, ...
 - **Pravidlostní ohodnocení** množiny prvotních výroků – funkce v z množiny prvotních formulí do množiny {0,1}

$$v: \{A_1, \dots, A_n\} \rightarrow \{0,1\}$$
 - o $v(A) = 1 \rightarrow A$ je **pravdivý při ohodnocení v**
 - o $v(A) = 0 \rightarrow A$ je **nepravdivý při ohodnocení v**
 - **Splnitelnost formulí** – Formule A je:
 - o **Tautologie** – je pravdivá ($v(A)=1$) pro každé ohodnocení - $\models A$
 - o **Kontradikce** – je nepravdivá ($v(A)=0$) pro každé ohodnocení
 - o **Splnitelná** – existuje ohodnocení, pro které je pravdivá
 - **Logická ekvivalence** A \equiv B – formule A a B jsou logicky ekvivalentní, právě když pro každé ohodnocení v je $v(A) = v(B)$. A \equiv B, právě když $A \Leftrightarrow B$ je tautologie.
 - **Logický důsledek** $A \models B$ – B je logickým důsledkem A, právě když pro každé ohodnocení v, pro které $v(A) = 1$, je i $v(B) = 1$. Říkáme, že B vyplývá z A.
 - **Univerzální systém logických spojek** – množina logických spojek tvoří univerzální systém, právě když ke každé formuli existuje logicky ekvivalentní formule, která obsahuje pouze tyto spojky
 - o Příklady: $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$; $\{\neg, \wedge, \vee, \Rightarrow\}$; $\{\neg, \wedge, \vee\}$; $\{\neg, \vee\}$; $\{\neg, \wedge\}$; $\{\neg, \Rightarrow\}$
 - o Shefferův symbol a Peirceova šipka – Oba tvoří **jednoprvkový univerzální systém** spojek (každý zvlášť)
 - **Shefferův symbol** $\uparrow = \text{NAND} = A \uparrow B \models \neg(A \wedge B)$
 - **Peirceova šipka** $\downarrow = \text{NOR} = A \downarrow B \models \neg(A \vee B)$
 - **Literál** – prvotní formule nebo negace prvotní formule – např. A ; $\neg A$; B
 - **Klausule** – literál nebo **disjunkce** několika literálů – např. $A \vee \neg B$; $\neg A \vee C \vee B$; $\neg C$
 - **Implikant** – literál nebo **konjunkce** několika literálů – např. $A \wedge \neg B$; $\neg A \wedge C \wedge B$; $\neg C$
 - **Konjunktivní normální tvar** – Formule je v KNT, jestliže je klausulí, nebo konjunkcí několika klausulí
 - o Příklad KNT – $(A \vee \neg B) \wedge C$
 - **Disjunktivní normální tvar** – Formule je v DNT, jestliže je implikant nebo je disjunkce několika implikantů
 - o Příklad DNT – $(A \wedge B \wedge C) \vee (\neg A \wedge \neg C)$
 - Formule, které jsou DNT i KNT (příkady): A ; $\neg A$; $A \wedge \neg B$; $A \vee B$
 - **Minterm** formule A – implikant, který obsahuje všechny prvotní formule vyskytující se v A
 - **Maxterm** formule A – klausule, která obsahuje všechny prvotní formule vyskytující se v A
 - **Úplný disjunktivní normální tvar** – disjunkce mintermů – např. $(A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C)$
 - **Úplný konjunktivní normální tvar** – konjunkce maxtermů – např. $(A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C)$
 - Tzn. pokud mám např. prvotní formule A, B, C a výsledný DNT implikant $(A \wedge \neg B)$, tak do úplného DNT to musím rozšířit o všechna C, co chybí $\rightarrow (A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C)$.
 - o Negace KNF je DNF, negace DNF je KNT.
 - o Minimální tvar DNT nebo KNT získáme s pomocí **Karnaughovy mapy**:

		C			
		00	01	11	10
00		0 0	0 1	0 3	0 2
01		0 4	0 5	0 7	0 6
11		0 12	0 13	1 15	1 14
10		0 8	0 9	1 11	1 10

- **Rezoluční metoda** – umožňuje rozhodnout o splnitelnosti formule a nalézt logické důsledky
 - o Vycházíme z KNT, hledáme jednoduché logické důsledky plynoucí z klauzulí
 - o **Rezoluční pravidla:**
 - $A, \neg A \vee B \vDash B$
 - $A \vee B, \neg A \vee C \vDash B \vee C$
 - $A, \neg A \vDash \perp$



- **Sémantické stromy** – grafická reprezentace formulí

- o Konstrukce sémantických stromů:

Konjunkce	Disjunkce	Implikace	Ekvivalence
$A \wedge B$	$A \vee B$	$A \Rightarrow B \vDash \neg A \vee B$	$A \Leftrightarrow B \vDash (A \wedge B) \vee (\neg A \wedge \neg B)$
$\begin{array}{c} \\ A \end{array}$	$\begin{array}{c} A \\ / \quad \backslash \\ A \quad B \end{array}$	$\begin{array}{c} A \Rightarrow B \\ / \quad \backslash \\ \neg A \quad B \end{array}$	$\begin{array}{c} A \Leftrightarrow B \\ / \quad \backslash \\ A \quad \neg A \\ \quad \\ B \quad \neg B \end{array}$
Negace konjunkce	Negace disjunkce	Negace implikace	Negace ekvivalence
$\neg(A \wedge B) \vDash \neg A \vee \neg B$	$\neg(A \vee B) \vDash \neg A \wedge \neg B$	$\neg(A \Rightarrow B) \vDash A \wedge \neg B$	$\neg(A \Leftrightarrow B) \vDash (A \wedge \neg B) \vee (\neg A \wedge B)$
$\begin{array}{c} / \quad \backslash \\ \neg A \quad \neg B \end{array}$	$\begin{array}{c} \\ \neg A \vee B \\ / \quad \backslash \\ \neg A \quad \neg B \end{array}$	$\begin{array}{c} \\ \neg(A \Rightarrow B) \\ / \quad \backslash \\ A \quad \neg B \end{array}$	$\begin{array}{c} / \quad \backslash \\ A \quad \neg A \\ \quad \\ \neg B \quad B \end{array}$

- **Logické zákony**

- o Tautologie
 - Zákon vyloučeného třetího $\neg A \vee \neg \neg A$
 - Vyloučení sporu $\neg(A \wedge \neg A)$
 - Zákon dvojí negace $\neg \neg \neg A \Leftrightarrow A$

- o Algebraické vlastnosti spojek

- $A \wedge A \vDash A$
- $A \vee A \vDash A$
- $A \wedge B \vDash B \wedge A$ – komutativní zákon pro \wedge
- $A \vee B \vDash B \vee A$ – komutativní zákon pro \vee
- $(A \wedge B) \wedge C \vDash A \wedge (B \wedge C)$ – asociativní zákon pro \wedge
- $(A \vee B) \vee C \vDash A \vee (B \vee C)$ – asociativní zákon pro \vee
- $(A \wedge B) \vee C \vDash (A \vee C) \wedge (B \vee C)$ – distributivní zákony
- $(A \vee B) \wedge C \vDash (A \wedge C) \vee (B \wedge C)$
- $A \wedge (A \vee B) \vDash A$ – zákony absorpcie
- $A \vee (A \wedge B) \vDash A$

- o Asociativní a komutativní zákony

- $A \wedge (B \wedge C) \vDash (A \wedge B) \wedge C$
- $A \vee (B \vee C) \vDash (A \vee B) \vee C$

- o Distributivní zákony

- $(A \wedge B) \vee C \vDash (A \vee C) \wedge (B \vee C)$
- $(A \vee B) \wedge C \vDash (A \wedge C) \vee (B \wedge C)$

- o Logicky ekvivalentní dvojice formulí

- $\neg(A \wedge B) \vDash \neg A \vee \neg B$ – de Morganovy zákony
- $\neg(A \vee B) \vDash \neg A \wedge \neg B$
- $A \Rightarrow B \vDash \neg A \vee B$ – "zlaté pravidlo"
- $\neg(A \Rightarrow B) \vDash A \wedge \neg B$ – "stříbrné pravidlo"
- $A \Rightarrow B \vDash \neg B \Rightarrow \neg A$ – zákon kontrapozice
- $A \Leftrightarrow B \vDash (A \Rightarrow B) \wedge (B \Rightarrow A)$ – význam ekvivalence

- Logické ekvivalence implikace:
 - $A \Rightarrow B$
 - $\neg A \vee B$
 - $\neg(A \wedge \neg B)$
 - $\neg B \Rightarrow \neg A$
- Algebraické vlastnosti implikace
 - Implikace není komutativní ani asociativní
 - Nemůžeme vynechat závorky
- Logické ekvivalence ekvivalence:
 - $A \Leftrightarrow B$
 - $(A \Rightarrow B) \wedge (B \Rightarrow A)$
 - $(A \Rightarrow B) \wedge (\neg A \Rightarrow \neg B)$
 - $(A \wedge B) \vee (\neg A \wedge \neg B)$
 - $(A \vee \neg B) \wedge (\neg A \vee B)$
 - $\neg A \Leftrightarrow \neg B$
- Algebraické vlastnosti ekvivalence
 - $A \Leftrightarrow B \text{ } \textcolor{brown}{\models} \text{ } B \Leftrightarrow A$ – komutativní zákon
 - $A \Leftrightarrow (B \Leftrightarrow C) \text{ } \textcolor{brown}{\models} \text{ } (A \Leftrightarrow B) \Leftrightarrow C$ – asociativní zákon
 - $\neg(A \Leftrightarrow B) \text{ } \textcolor{brown}{\models} \text{ } \neg A \Leftrightarrow B$
 - $A \Leftrightarrow A \text{ } \textcolor{brown}{\models} \text{ } \top$
 - $A \Leftrightarrow \neg A \text{ } \textcolor{brown}{\models} \text{ } \perp$
 - $A \Leftrightarrow \top \text{ } \textcolor{brown}{\models} \text{ } A$
 - $A \Leftrightarrow \perp \text{ } \textcolor{brown}{\models} \text{ } \neg A$
- Vlastnosti tautologie \top a kontradikce \perp
 - $A \wedge \top \text{ } \textcolor{brown}{\models} \text{ } A$
 - $A \vee \top \text{ } \textcolor{brown}{\models} \text{ } \top$
 - $A \wedge \perp \text{ } \textcolor{brown}{\models} \text{ } \perp$
 - $A \vee \perp \text{ } \textcolor{brown}{\models} \text{ } A$

Predikátová logika: jazyk, interpretace, splnitelnost formulí, logická platnost, logická ekvivalence a důsledek, teorie, model.

BI-MLO

- Jazyk predikátové logiky

- o Logické symboly

- Logické spojky - $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
 - Proměnné – x, y, z, \dots
 - Kvantifikátory – vždy následovány proměnnou
 - Obecný - \forall - pro všechny, všichni
 - Existenční - \exists - některé, existuje

- o Mimologické symboly L

- Konstanty – K, S, ...
 - Predikáty – p, q, r, ... – dána četnost
 - Funkce – f, g, ... – dána četnost

$$(\forall \epsilon)(\epsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))).$$

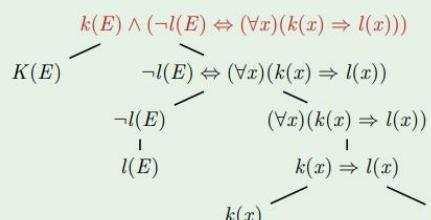
- logické spojky: \Rightarrow, \wedge
- proměnné: x, ϵ, δ
- kvantifikátory: $(\forall), (\exists)$
- konstanty: $a, 0$
- binární funkce: rozdíl –
- unární funkce: f
- unární funkce: absolutní hodnota ||
- binární predikát: $>$

$$L = \{a, 0, f, ||, -, >\}$$

- Logické symboly jsou univerzální pro všechny jazyky, proto se jazyk definuje jako množina mimologických symbolů: $L = \{K, \dots, p, \dots, f, \dots\}$. (K = konstanty, p = predikáty, f = funkce)
- **Term** – řetězec symbolů v jazyce predikátové logiky, jež vznikl použitím následujících pravidel v konečně mnoha krocích:
 - o Každá proměnná a konstanta je term.
 - o Jsou-li $t_1 \dots t_n$ termy a f je n-ární funkční symbol, potom $f(t_1 \dots t_n)$ je term
- **Formule** – posloupnost symbolů v jazyce predikátové logiky, která vznikla aplikací následujících pravidel v konečně mnoha krocích:
 - o Je-li p n-ární predikátový symbol a $t_1 \dots t_n$ jsou termy, pak $p(t_1 \dots t_n)$ je formule. Takto vzniklou formuli nazýváme atomická formule
 - o Jsou-li A, B formule, pak $\neg A, (A \wedge B), (A \vee B), (A \Rightarrow B), (A \Leftrightarrow B)$ jsou formule
 - o Je-li x proměnná a A formule, pak $(\forall x)A$ a $(\exists x)A$ jsou formule
- **Podformule** B – část formule A, která je sama formulí
- Proměnná x má **vázaný výskyt** v A právě, když se vyskytuje v její podformuli ve tvaru $(\forall x)B(x)$ nebo $(\exists x)B(x)$
 - o **Volný výskyt** – výskyt proměnné A, který není vázaný
 - o **Uzavřená formule** – obsahuje pouze vázané proměnné – sentence
 - o **Otevřená formule** – obsahuje pouze volné proměnné
- **Interpretace** (realizace, struktura) $\mathcal{M} = \langle M, \dots, K_{\mathcal{M}}, \dots, p_{\mathcal{M}}, \dots, f_{\mathcal{M}}, \dots \rangle$ jazyk L obsahuje
 - o Neprázdnou množinu M – **universum** interpretace
 - o Je-li K konstanta, pak její interpretaci $K_{\mathcal{M}} \in M$
 - o Je-li p n-ární predikát, pak n-ární relaci $p_{\mathcal{M}} \subseteq M^n$ jako jeho interpretaci
 - o Je-li f funkce mající n argumentů, pak funkci $f_{\mathcal{M}}: M^n \rightarrow M$ jako její interpretaci
- **Formalizační strom** formule – zachycuje strukturu formule
 - o Koncové uzly = atomické formule
 - o Formační strom do vyšších pater se postupně dostáváme užitím pravidel 2 a 3 z definice formule

$$k(E) \wedge (\neg l(E) \Leftrightarrow (\forall x)(k(x) \Rightarrow l(x)))$$

$L = \{l(x), k(x), E\}$, kde l, k jsou unární predikáty, E je konstanta.



- Formalizace matematických tvrzení v daném jazyce (nejspíš tam nebude)
 - Definujeme jazyk, predikáty (vstupní parametr je volný), vhodně kombinujeme
 - Formalizace v aritmetice:
 - V jazyku $L = \{+, \cdot, 0, =\}$, kde $+$, \cdot jsou binární funkční symboly, 0 je konstanta a $=$ je binární predikátový symbol zapište v interpretaci $\mathcal{N} = (\mathbb{N}, +_{\mathcal{N}}, \cdot_{\mathcal{N}}, 0_{\mathcal{N}}, 0_{\mathcal{N}})$:

Pro každá dvě čísla existuje největší společný dělitel.

 - x dělí y $(\exists u)(y = x \cdot u)$
 - x je společný dělitel y a z - $(\exists u)(y = x \cdot u) \wedge (\exists v)(z = x \cdot v) - d(x, y, z)$
 - x je menší nebo rovno y $(\exists u)(y = x + u) - x \leq y$
 - x je největší společný dělitel y a z - $d(x, y, z) \wedge (\forall w)(d(w, y, z) \Rightarrow w \leq x)$
 - Existuje největší spol. děl. y a z - $(\exists x)(d(x, y, z) \wedge (\forall w)(d(w, y, z) \Rightarrow w \leq x))$
 - Pro každá dvě čísla existuje největší společný dělitel.

$(\forall y)(\forall z)((\exists x)(d(x, y, z) \wedge (\forall w)(d(w, y, z) \Rightarrow w \leq x)))$
 - Formalizace tvrzení o reálných funkčích:
 - V jazyku $L = \{f, \leq\}$ zformalizujte tvrzení o reálných číslech:

Na každém uzavřeném intervalu funkce f nabývá maximum.

 - Funkce nabývá maxima v bodě z . $(\forall x)(f(x) \leq f(z))$
 - Funkce nabývá maxima. $(\exists z)(\forall x)(f(x) \leq f(z))$
 - Funkce nabývá maxima v uzavřeném intervalu $< a, b >$.
 $(\exists z)((a \leq z \wedge z \leq b) \wedge (\forall x)(a \leq x \wedge x \leq b) \Rightarrow (f(x) \leq f(z)))$
 - Funkce f nabývá maxima na každém uzavřeném intervalu.
 $(\forall a)(\forall b)(\exists z)((a \leq z \wedge z \leq b) \wedge (\forall x)(a \leq z \wedge z \leq b) \Rightarrow (f(x) \leq f(z)))$
 - Formalizace teorie množin:
 - $L = \{\in\}$, \in binární predikátový symbol, $x \in y$, termy x, y, z
 - $u \subseteq v$ - $(\forall x)((x \in u) \Rightarrow (x \in v))$
 - $u = v$ - $(\forall x)(x \in u \Leftrightarrow x \in v)$
 - $u = v \cap w$ - $(\forall x)(x \in u \Leftrightarrow ((x \in v) \wedge (x \in w)))$
 - $u = v \cup w$ - $(\forall x)(x \in u \Leftrightarrow ((x \in v) \vee (x \in w)))$
 - $u = \emptyset$ - $(\forall x)\neg(x \in u)$
 - $u = \{x\}$ - $(\forall v)(v \in u \Leftrightarrow v = x)$
 - $u = \{x, y\}$ - $(\forall v)(v \in u \Leftrightarrow (v = x \vee v = y))$
 - Negace kvantifikátorů – znegujeme kvantifikátory a podformule
 - Nechť A je formule predikátové logiky. Potom platí:
 - (i) $\neg(\forall x)A \Leftrightarrow (\exists x)\neg A$
 - (ii) $\neg(\exists x)A \Leftrightarrow (\forall x)\neg A$
 - Příklad: Negace tvrzení „Funkce f není spojitá v bodě a “.
 - $\neg(\forall \epsilon)(\epsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)\neg(\epsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge \neg(\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge (\forall \delta)\neg(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge (\forall \delta)(\neg(\delta > 0) \vee \neg(\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge (\forall \delta)((\delta > 0) \Rightarrow (\exists x)\neg(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge (\forall \delta)((\delta > 0) \Rightarrow (\exists x)(|x - a| < \delta \wedge \neg|f(x) - f(a)| < \epsilon))) \Leftrightarrow$
 - $\neg(\exists \epsilon)(\epsilon > 0 \wedge (\forall \delta)(\delta > 0 \Rightarrow (\exists x)(|x - a| < \delta \wedge |f(x) - f(a)| \geq \epsilon))) \Leftrightarrow$
 - Aristotelův čtverec
 - $\neg(\forall x)(k(x) \Rightarrow s(x)) \Leftrightarrow (\exists x)\neg(k(x) \Rightarrow s(x)) \Leftrightarrow (\exists x)(k(x) \wedge \neg s(x))$
 - $\neg(\exists x)(k(x) \wedge s(x)) \Leftrightarrow (\forall x)(\neg k(x) \vee \neg s(x)) \Leftrightarrow (\forall x)(k(x) \Rightarrow \neg s(x))$

	Kladné	Záporné
Obecné	A $(\forall x)(k(x) \Rightarrow s(x))$ Všechny kočky jsou šelmy	E $(\forall x)(k(x) \Rightarrow \neg s(x))$ Zádné kočky nejsou šelmy
Částečné	I $(\exists x)(k(x) \wedge s(x))$ Některé kočky jsou šelmy	O $(\exists x)(k(x) \wedge \neg s(x))$ Některé kočky nejsou šelmy

- **Ohodnocení proměnných** = funkce z množiny proměnných, která každé proměnné přiřazuje nějaký prvek universa
- **Pravdivost formule A v interpretaci \mathcal{M} při ohodnocení e: $\mathcal{M} \models A[e]$**
 - Formule A je **pravdivá (platná) v interpretaci \mathcal{M}** ($\mathcal{M} \models A$), právě když pro každé ohodnocení e je pravdivá ($\mathcal{M} \models A[e]$)
 - Formule A je **logicky pravdivá** (tautologie PL), právě když pro každou interpretaci \mathcal{M} platí $\mathcal{M} \models A$. Značíme $\models A$
 - Formule A je **splnitelná**, právě když v nějaké interpretaci \mathcal{M} pro nějaké ohodnocení e platí $\mathcal{M} \models A[e]$
 - Formule A je **kontradikce**, právě když není splnitelná
 - Věta o uzávěru – Pro každou formuli A predikátové logiky platí:
 - A(x) je logicky pravdivá, právě když $(\forall x)A(x)$ je logicky pravdivá
 - A(x) je splnitelná, právě když $(\exists x)A(x)$ je splnitelná
 - Příklady:

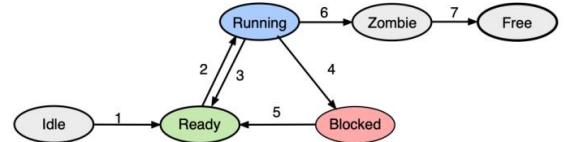
Formule v jazyce $L = \{r(x, y), =\}$, kde $r(x, y)$ je binární predikát.

1. $(\forall x)(\forall y)(r(x, y) \Rightarrow r(y, x))$ **symetrie**
2. $(\forall x)(\forall y)(r(x, y) \Rightarrow \neg r(y, x))$ **antisymetrie**
3. $(\forall x)r(x, x)$ **reflexivita**
4. $(\forall x)\neg r(x, x)$ **ireflexivita**
5. $(\forall x)(\forall y)(\forall z)((r(x, y) \wedge r(y, z)) \Rightarrow r(x, z))$ **transitivita**
6. $(\forall x)(\exists y)r(x, y)$
7. $(\forall x)(\exists y)r(y, x)$
8. $(\exists x)(\forall y)r(x, y)$
- **Logická ekvivalence** – formule A a B jsou logicky ekvivalentní ($A \equiv B$), právě když pro každou interpretaci \mathcal{M} a pro každé ohodnocení e platí: $\mathcal{M} \models A[e]$, právě když $\mathcal{M} \models B[e]$
- **Logický důsledek** – B je logickým důsledkem A ($A \models B$), právě když pro každou interpretaci \mathcal{M} a pro každé ohodnocení e platí: jestliže $\mathcal{M} \models A[e]$, pak i $\mathcal{M} \models B[e]$
 - $A \models B$, právě když $A \Rightarrow B$ je **logicky pravdivá**.
 - $A \models B$, právě když $A \wedge \neg B$ je **kontradikce**.
 - $A \models B$, právě když $A \models B$ a zároveň $B \models A$.
 - $A \models B$, právě když $A \Leftrightarrow B$ je **logicky pravdivá**.
- Logická ekvivalence VL, kde prvotní formule jsou nahrazeny formulami PL, je logická ekvivalence PL
- Logický důsledek VL, kde prvotní formule jsou nahrazeny formulami PL, je logický důsledek PL
- **Teorie** = množina uzavřených formulí $T = \{A_1, \dots, A_n\}$ jazyka L
 - **Interpretace \mathcal{M} jazyka L je modelem teorie T**, jestliže každá formule T platí v \mathcal{M} . Příšeme $\mathcal{M} \models T$
 - Formule A je **logický důsledek teorie T** (A logicky vyplývá z T), jestliže v každém modelu teorie T platí A . Příšeme $T \models A$.
 - Teorie T je **splnitelná**, právě když **má model**.

Procesy a vlákna, jejich implementace. Synchronizační nástroje. Klasické synchronizační úlohy. Plánování vláken. Přidělování prostředků, Coffmanovy podmínky, způsoby řešení uváznutí

BI-OSY

- Program = posloupnost instrukcí definující chování procesu
- **Proces** – instance spuštěného programu
 - Entita, v rámci které jsou alokovány prostředky (paměť, vlákna, otevřené soubory, sokety, ...)
 - Implicitně každý proces má jedno výpočetní „main“ vlákno
 - Jádro OS pro každý proces udržuje řadu datových struktur pro:
 - Identifikaci – číslo procesu (PID), číslo rodiče procesu (PPID), ...
 - Bezpečnost – identita procesu (USER, ...), ...
 - Správu paměti – info. Pro překlad virt. Adres (page table), ...
 - Správu FS – tabulka deskriptorů souborů, ...
 - Při vzniku nového procesu, část datových struktur je zděděna od rodiče. proc., část nastavena na nové hodnoty spec. pro nový proces
 - **Implementace procesů**
 - **Tabulka procesů** – jádro OS udržuje info o procesech pomocí zřetězeného seznamu struktur
 - **Process control block** – PCB – jedna položka tabulky, která reprezentuje všechny nezbytné informace o 1 procesu (identifikace, identita procesu/bezpečnosti – vlastník procesu, alokované prostředky – paměť, otevřené soubory)
- **Vlákno** – výpočetní entita (průběh instrukcí), které je přidělováno jádru CPU
= „light-weight process“
 - Vlákna vytvořená v rámci procesu sdílí většinu prostředků alokovaných v tomto procesu
 - Jádro OS pro každé vlákno udržuje dat. struktury pro:
 - Identifikaci – číslo vlákna (TID)
 - Zásobník – lokální proměnné, historie volání
 - Inf. nutné pro přepínání kontextu – čítač instrukcí, aktuální hodnoty registrů
 - Inf. pro plánování vláken – priorita, čas strávený na CPU
 - **Stavy vláken** – popisují, co se s daným vláknenem právě děje
 - *Idle* – vznik nového vlákna
 - *Ready* – vlákno čeká na přidělení jádra CPU
 - *Running* – vlákno zpracováváno jádrem CPU
 - *Blocked* – vlákno čeká na událost
 - *Zombie* – vlákno je ukončováno, ale zatím ještě nebylo vše dokončeno
 - *Free* – vlákno bylo kompletně zrušeno (pouze teoretický stav)
- **Implementace vláken**
 - *V uživatelském prostoru* – kompletně podporována/spravována v už. Prost. Pomocí „run-time“ systému
 - Už. Vlákna používají kooperativní plánování
 - Jádro OS nemá „zádnou“ informaci o uživatelských vláknech v jednotlivých procesech a spravuje je jako proces s jedním vláknem
 - jednoduchá/rychlá správa – vytvoření, přepínání, synchronizace vláken je v uživatelském prostoru bez zásahu jádra OS
 - Vlákna jednoho procesu jsou mapována na jedno jádro CPU
 - Blokující volání v jednom vláknu zablokuje i ostatní vlákna procesu
 - = model many-to-one – více uživatelských vláken namapováno na jedno kernel vlákno
 - *V jádru OS* – spravována/podporována přímo jádrem OS
 - Jádro OS



- Spravuje jeden PCB pro každý proces
 - Spravuje jeden TCB pro každé vlákno
 - Poskytuje systémová volání pro vytváření/správu vláken z uživatelského prostoru
 - Jádro má všechny informace o všech vláknech – jádro OS přiděluje jednotlivá jádra CPU jednotlivým vláknům na určitou dobu
 - Pokud zavolá vlákno blokující systémové volání – zablokuje se pouze toto vlákno
 - Vytvoření/ukončení vlákna, systémové volání, přepnutí kontextu – představuje přepnutí z user modu do kernel modu
 - = model one-to-one – jedno uživatelské vlákno namapováno na jedno kernel vlákno
 - Hybridní implementace – model many-to many, m uživatelských vláken namapováno na n kernel vláken
- **Plánování vláken** – v okamžiku, kdy se uvolní jádro CPU, musí OS vybrat „vhodné“ vlákno ve stavu „ready“, a umožní mu používat toto jádro po určitou dobu
- V OS je obvykle implementováno několik různých plánovacích strategií a uživatel/administrátor může pro různé aplikace nastavit vhodnou strategii, popřípadě určí, na kterých jádrech CPU poběží
 - **Typy aplikací z hlediska plánování vláken**
 - Dávkové úlohy (aplikace běžící na pozadí) – obvykle zpracovávají data, která načítají ze/zapisují do souborů, a nekom. přímo s uživatelem, vyžadují hodně CPU výkonu
 - Interaktivní aplikace – reagují na události, které přicházejí z GUI, CLI, síťového rozhraní,...
 - Úlohy reálného času – většinou nevyžadují mnoho CPU výkonu, ale je nutné zajistit/garantovat co nejrychlejší reakci na událost
 - **Typy vláken** z hlediska plánování OS
 - **Vlákna orientovaná na CPU (CPU-bound)**
 - CPU využíváno po dlouhou dobu, málo blokujících operací
 - **Vlákna orientovaná na V/V (I/O-bound)**
 - CPU využito po krátkou dobu, hodně blokujících operací
 - **Vlákna reálného času (realtime)** – vlákno musí zareagovat na událost během daného intervalu
- **Přepínání kontextu** – mechanismus, při kterém se vlákna vystřídají v používání jádra CPU
- **Kontext** = všechny nezbytné inf. nutné pro pozdější spuštění přeruš. vlákna od okamžiku přerušení
 - Vlákna žijí v iluzi, že běží bez přerušení od začátku do konce
 - Několik kroků:
 - a. Uloží se kontext původního vlákna
 - b. Jádro OS naplánuje další vlákno
 - c. Nastaví se kontext tohoto vlákna
- **Minimalizace přepnutí kontextu** – v okamžiku uvolnění jádra CPU OS musí vybrat vhodné v. ve stavu Ready
- **Plánování s odnímáním (preemptive scheduling)**
 - Jádro OS přidělí vláknu jádro CPU pouze na určitou dobu, po uplynutí mu ho odebere
 - Vhodná pro vlákna v interaktivních systémech
 - Umožňuje rozumně sdílet CPU výkon systému
 - Dochází k častějšímu přepínání kontextu – horší využití CPU, lepší doba odezvy
 - **Round-robin plánování (RR)** – vlákna ve stavu „Ready“ čekají ve frontě FIFO, až jim bude přiděleno jádro CPU
 - Všem vláknům je jádro CPU propůjčováno na stejně velkou dobu (**časové kvantum**). Po uplynutí doby je jádro CPU vláknům odebráno, změní se jejich stav na „ready“ a vlákna se zařadí na konec fronty (Ready queue)
 - **Prioritní RR se statickou prioritou** – každému vláknu přiřazena statická priorita (číslo 1,...,N), která se během existence vlákna nemění (vyšší číslo = vyšší priorita)
 - Volné jádro CPU je vždy přiřazeno vláknu s nejvyšší prioritou, které je na začátku příslušné fronty (Ready queue)
 - „ready“ fronty implementovány jako RR -> FIFO
 - Časové kvantum může být pro různé priority různě velké

- **Prioritní RR s dynamickou prioritou** – výchozí plánovací strategie běžných OS
 - Priorita i velikost časového kvanta se může během existence vlákna měnit
 - Cíl minimalizovat problém hladovění vláken, zlepšit odezvu a snížit počet přepnutí kontextu
 - Strategie:
 - Priorita se zvýší a časové kvantum sníží – pokud vlákno v posledním běhu nevyužilo celé své časové kvantum nebo dlouho čeká na CPU
 - Priorita se sníží a časové kvantum se zvýší – pokud vlákno v posledním běhu využilo celé své časové kvantum
 - **Kooperativní plánování** (cooperative scheduling) – není vhodná pro běžná uživatelská vlákna
 - Jádro OS přidělí vláknu jádro CPU a vlákno ho používá, dokud ho samo neuvolní
 - First-come-first-served (FCFS) – vlákna ve stavu „ready“ čekají ve FIFO frontách na přidělení jádra CPU, když běžící vlákno uvolní jádro CPU, první vlákno z Ready fronty s nejvyšším číslem bude pokračovat
 - Vlákna ve stavu blocked čekají ve frontě Blocked
 - Jednoduché na implementaci, minimalizuje počet přepnutí kontextu
- **Časově závislé chyby** (race conditions) – situace, kdy dvě nebo několik vláken používá společné sdílené prostředky a výsledek deterministického algoritmu je závislý na rychlosti jednotlivých vláken, které používají tyto prostředky
 - Deterministický alg. – vždy ze stejných vých. (vstupních) podm. svým během vytvoří stejné výsledky
 - Náhodný výskyt => špatná detekce
 - Předcházet časově závislým chybám bychom měli správný návrhem paralelního algoritmu
- **Kritické sekce** (critical regions) – části programu, kde vlákna používají sdílené prostředky (sdílená proměnná, sdílený soubor, ...)
 - Sdružené kritické sekce – kritické sekce dvou (či více) vláken, které se týkají stejného sdíleného prostředku
 - Vzájemné vyloučení – vláknům není dovoleno sdílet stejný prostředek ve stejném čase => vlákna se nesmí nacházet ve stejné sdružené kritické sekci současně
- **Synchronizační nástroje**
 - Bez použití synchronizace – časově závislé chyby (race conditions) – více vláken používá společné sdílené prostředky během deterministického algoritmu a jeho výsledek je závislý na rychlosti vláken
 - Při použití synchronizace
 - Uváznutí (deadlock) – několik vláken čeká na událost, kterou může vyvolat jen jeno z čekajících vláken
 - Livelock – několik vláken vykonává neužitečný výpočet, ale nemohou dokončit výpočet
 - Hladovění (starvation) – vlákno ve stavu „ready“ je předbíháno a nedostane se dlouho k prostředkům
- **Synchronizace kritické sekce**
 - Pouze jedno vlákno může být uvnitř kritické sekce – ostatní musí počkat, dokud se neuvolní
 - Pomocí aktivního čekání nebo blokujících systémových volání/knihovních funkcí
 - **Aktivní čekání** (busy waiting, spinning)
 - Sdílená proměnná indikuje obsazenost kritické sekce (zamčená/odemčená)
 - Před vstupem do:
 - Zamčené sekce – vlákno ve smyčce „aktivně“ testuje akt. Hodnotu proměnné do okamžiku, než se sekce uvolní
 - Odemčené sekce – vlákno změní hodnotu sdíl. Proměnné (zamkne krit. sekci) a vstoupí do sekce
 - Po opuštění krit. sekce – vlákno změní hodnotu sdíl. Proměnné (odemkne sekci)
 - **Blokující volání** – implementováno pomocí datových struktur, které umožňují pamatovat si stav krit. sekce, udržovat seznam vláken, která čekají na vstup do krit. sekce
 - Před vstupem do krit. sekce:

- Zamčená sekce – vlákno provede sys. Volání/knihovní funkci, které ho zablokuje a tím pádem vláknu přestane být přidělován procesor => pasivně čeká na uvolnění sekce
 - Odemčená sekce – vlákno provede sys. Volán, které ho nezablokuje ale pouze si pamatuje, že sekce je zamčená, a vlákno vstoupí do sekce
 - Po opuštění krit. sekce vlákno pomocí sys. Volání probudí čekající vlákno/vlákna, v případě, že žádná vlákna nečekají si pamatuje, že sekce je odemčená
 - **Zámek – mutex** (MUTual EXclusion lock) – pamatuje si svůj stav, kdo je jeho vlastníkem (vlákno, které ho zamklo), množinu vláken, která jsou na něm blokovaná
 - **Podmíněné proměnné** – pamatuje si, která vlákna jsou na nich blokovány
 - **Semafor** – datový typ obsahující celočíselný čítač, pamatuje si množinu vláken, která jsou na něm zablokována
 - **Bariéra** – umožňuje jednoduše synchronizovat iterační výpočty, obsahuje čítač definující sílu bariéry (počet vláken, které bariéru prolomí) a frontu vláken, která jsou na bariéře blokovány
- **Klasické synchronizační úlohy**
1. **Večeřící filozofové**
 - Reprezentuje situaci, kdy několik vláken soutěží o omezený počet prostředků
 - V systému N filozofů sedících kolem kulatého stolu, před každým talíř s jídlem a mezi sousedy vždy jedna vidlička (celkem N). Pokud dostane filozof hlad, musí získat obě vidličky, které leží napravo a nalevo od něj. F může být ve 3 stavech – přemýšlí, má hlad a snaží se získat prostředky, jí
 - Optimální řešení – v jeden okamžik může jít až floor(N/2) filozofů
 - Správné optimální řešení – mutexy a N semaforů/podmíněných proměnných
 2. **Čtenáři – písáři**
 - 2 typy vláken soutěží o přístup ke společnému prostředku
 - Máme 1 sd. Prostředek a 2 typy vláken – čtenáři (pouze read) a písáři (mohou write), pouze jeden písář může modifikovat prostředek v jednom okamžiku.
 - Optimální řešení – více čtenářů může číst současně, pokud žádný písář nepřistupuje k prostředku
 - Spravedlivé řešení – pokud p/c čeká, pak by ho nikdo neměl předběhnout
 - Problém hladovění – řešení pomocí dvou mutexů
 - Optimální spravedlivé řešení – musíme si pamatovat v jakém pořadí čtenáři a písáři začínají čekat
 3. **Spící holiči**
 - V holičství je N holičů, N křesel k holení a M křesel k čekání pro zákazníka. Pokud není žádný zákazník v holičství, holič sedne do křesla k holení a usne. Pokud přijde zákazník: holič je volný (spí) – zákazník se nechá ostříhat / holič není volný, ale je volné místo v čekárně – z. počká / jinak opustí holičství
 - Správné optimální řešení – jeden mutex a dva semafory
 4. **Producent – konzument**
 - Několik vláken si vyměňuje data prostřednictvím např. sdílené paměti s omezenou velikostí
 - Producent – produkuje data a vkládá je do sdílené paměti (fronty) s omezenou velikostí
 - Konzument – vybírá data ze sdílené paměti (fronty)
 - Problémy:
 - Musíme zajistit výlučný přístup při vkládání/vybírání dat z fronty
 - Pokud je fronta prázdná, musíme zablokovat konzumenta
 - Pokud je fronta plná, musíme zablokovat producenta
- **Uváznutí vláken**
- **Výpočetní prostředky** – fyzické x logické
 - Sdílené výp. Prostředky – pokud je p. sdílen více vláknem současně, mohou vznikat časově závislé chyby – nutno zajistit výlučný přístup k prostředku
 - Paralelní přístup k více s. p. – vlákna často potřebují přistupovat k více p. současně
 - Typy sdílených prostředků:
 - Odnímatelné – již alokovaý p. může být vláknu odebrán bez rizika dalších problémů
 - Neodnímatelné – nemohou být odebrány bez rizika, většina prostředků
 - **Sekvence kroků při použití sdíleného prostředku vláknem:** *alokace – použití – uvolnění*

- Alokace prostředku – vlákno žádá o prostředek prostřednictvím alokační funkce
 - Pokud je p. volný, je přidělen vláknu
 - Pokud je p. alokovaný, tak:
 - A. Vlákno bude blokováno, dokud prostředek nebude k dispozici
 - B. Vlákno bude blokováno max. po určitý čas
 - C. Vlákno nebude blokováno
 - V příp. B a C vlákno zjistí např. na zákl. návratové hodnoty fce, zda mu byl p. přidělen, nebo ne, potom musí rozhodnout, jak bude výpočet pokračovat
 - Uvolnění prostředku – vlákna by sama měla uvolňovat alok. Prostředky
 - **Uváznutí (deadlock)** – několik vláken čeká na událost/prostředek, kterou může vyvolat/uvolnit pouze jedno z čekajících vláken
- **Coffmanovy podmínky** – uváznutí nastane v případě jejich splnění (pokud alespoň 1 není splněna, nenastane)
 - A. Vzájemné vyloučení – každý prostředek je buď přidělen právě jednomu vláknmu, nebo je volný
 - B. Podmínka neodnímatelnosti – prostředek, který byl již přidělen vláknu mu nemůže být násilím odebrán
 - C. Podmínka „drž a čekej“ – vlákno, které má přiděleno prostředky, může zařádat o další
 - D. Podmínka kruhového čekání – musí existovat smyčka dvou nebo více vláken, ve které každé vlákno čeká na prostředek přidělený dalšímu vláknu ve smyčce
 - A – C jsou nutné, ale ne dostačující – k uváznutí může dojít, D představuje samotné uváznutí
- **Strategie pro řešení uváznutí**
 - A. *Pštrosí strategie* – ignorování celého problému
 - Problém uváznutí se neřeší/řeší se částečně – při uváznutí nutný zásah uživatele/admina
 - B. *Prevence uváznutí* – pomocí nespolnění aspoň jedné z C.P.
 - C. *Předcházení vzniku uváznutí* – na základě pečlivé alokace prostředků – předem známe všechny požadavky vláken na prostředky a vytvoříme bezpečný stav, který garantuje postupné uspokojení potřeb všech vláken
 - D. *Detekce uváznutí a zotavení* – k uváznutí může dojít, ale je detekováno a odstraněno
 - 2 fáze – v systému jsou prováděny pravidelné kontroly, které se snaží **odhalit uváznutí** (jednoduché) a **zotavuje** se tím, že se část prostředků uvolní, tím se poruší čekání ve smyčce (složité, např. ukončení vláken)
 - **Detekce** – vytvoříme **kopii vektoru volných prostředků C**, na začátku jsou všechna **vlákna neoznačená** → označíme vlákna, která nechtějí žádný prostředek → existuje vlákno, které lze uspokojit prostředky z C? Pokud ano, označíme ho a alokované prostředky přičteme k vektoru C a ptáme se znova, dokud žádná nezbydou → byla **označena všechna vlákna**? Pokud ano, k uváznutí nedošlo, pokud ne, uváznutí nastalo
 - **Nástroje pro detekci** – příkazy pro zobrazení stavu vláken, příkazy zobrazující zásobníky vláken, aplikace pro ladění a profilování programů (Valgrind)
 - **Zotavení**
 - Ukončení všech uvázlých vláken/Postupné ukončování uvázlých vláken
 - Zotavení pomocí návratu restartu – znova spuštění uvázlých vláken z některého z předchozích stavů (nutná implementace mech. návratu – rollback a restartu)
- **Tvorba vláken v závislosti na systému**
 - OS API/Knihovny
 - Proprietární knihovny jednotlivých OS – Microsoft Windows API, Solaris thread library, ...
 - POSIX thread library – MS Windows, GNI/Linux, FreeBSD, ...
 - OpenMP – MS Windows, GNU/Linux, FreeBSD, Solaris
 - Programovací jazyky s vestavěnou podporou vláken – C++ (od C++11), Java, ...
- **Tvorba procesu** v závislosti na systému
 - Nový proces může být kopie/klon původního procesu nebo nový proces
 - OS unixového typu – fork(), execve(), ...
 - MS Windows – CreateProcessA(), ...

Virtualizace paměti pomocí stránkování, principy překladu logických adres na fyzické, algoritmy pro nahrazování stránek

BI-OSY

- **Viruální adresový prostor procesu (VAS)**
 - Každý proces má svůj VAS, který se mapuje do fyzické paměti
 - Jakým způsobem bude VAS procesu mapován do gyz. Paměti závisí na konkrétní architektuře CPU a použitém OS
- **Virtuální paměť**
 - VAS každého procesu automaticky rozdělen na menší kousky a ve fyzické paměti musí být pouze kousky aktuálně používané, zbytek používaného VAS je na disku
- **Virtuální paměť se stránkováním**
 - Proces používá adresy, kterým se říká virtuální adresy a které adresují virtuální adresový prostor
 - VAS rozdělen na stejně velké souvislé oblasti – virtuální stránky
 - Korespondující úseky ve fyz. paměti stejně velikosti = rámce stránky
 - V hlavní paměti musí být aspoň stránky aktuálně používané
- **Problém:** Fyzická paměť je po určitém čase fragmentovaná (obsahuje mnoho malých volných oblastí, např. když v buddy systému (varianta zřetězených seznamů) mám malé zabrané oblasti a uvolním každý lichý, aby to nešlo mergnout).
- Řešení:** Virtuální adresový prostor (VAS) budeme alokovat jako množinu malých oblastí, tím odpadne problém s nalezením velké souvislé oblasti ⇒ stránkování
- **Stránkování** – množina stejně velkých oblastí fyzické paměti
 - Fyzická paměť (hlavní paměť) – úseky stejně velikosti – rámce (frames)
 - VAS – rozdělený na úseky stejně velikosti – stránky (pages)
 - Velikost rámce a stránky je stejná
 - Jednotlivé stránky VAS se nahrávají do volných rámců fyzické paměti
 - OS si musí pamatovat rámce přidělené procesům a volné rámců
- **Memory management unit (MMU)**
 - Zajišťuje, že proces má iluzi, že celý jeho VAS je v hlavní paměti a pro adresaci instrukcí/dat používá virtuální adresy
 - Výpadek stránky (page fault) – pokud není virtuální stránka v hl. paměti, MMU způsobí přerušení a „požádá“ OS o nahrání příslušné stránky do fyz. paměti. OS nejdřív najde vhodný rámec f. p., pak do něj nahraje požadovanou virtuální stránku s disku nebo na něm „vytvoří“ novou stránku
- **Překlad virtuálních adres na fyzické**
 - OS si musí udržovat inf. o tom, do kterých rámců fyzické paměti se namapovaly jednotlivé stránky VAS jednotlivých procesů, a které rámce fyz. paměti jsou zatím volné
 - Tuto informaci si OS udržuje v následujících strukturách, které jsou závislé na ISA použitého procesoru
 - Tabulka stránek
 - Víceúrovňová tabulka stránek
 - Invertovaná tabulka stránek
 - Pro urychlení překladu MMU využívá „Translation lookaside buffer“ – TLB, ve kterém jsou informace o naposledy překládaných virtuálních adresách
- **Tabulka stránek**
 - Obsahuje pro každou stránku VAS daného procesu jednu řádku, ve které jsou uložené informace:
 - Číslo rámce, do kterého se stránka namapovala
 - Kontrolní bity
 - Present bit – zda je stránka v hlavní paměti
 - Reference bit – zda se ke stránce přistupovalo
 - Modify bit – zda byl obsah stránky modifikován
 - Přístupová práva

- Cache disabled/enabled – zda jsou registry periferií mapovány přímo do paměti
 - Read/write – zda lze stránku modifikovat
 - User/supervisor – zda lze na stránku přistupovat v uživatelském módu
 - Číslo stránky = index tabulky
 - OS si musí udržovat pro každý proces jednu tabulku
 - Problémy
 - Přestože proces používá pouze zlomek místa ze svého VAS, tabulka stránek obsahuje informaci i o nepoužitých stránkách
 - Pro každý proces musí OS držet jednu tabulku stránek – plýtvání pamětí
 - Řešení – víceúrovňová tabulka stránek, která umožňuje se držet v paměti pouze informace o používaných stránkách – u procesů, které alokují málo paměti, můžeme ušetřit
- **Víceúrovňová tabulka stránek**
- Pro n úrovňovou tabulku stránek platí
 - Virtuální adresa se skládá z n indexů, které ukazují do tabulek jednotlivých úrovní, a offsetu
 - Fyzická adresa se skládá z čísla rámce a offsetu
 - Tabulky stránek úrovní 1, ..., n – 1 obsahují „present bit“ a číslo rámce, ve kterém je uložena/začíná tabulka následující úrovně
 - Tabulka stránek úrovně n obsahuje „present bit“ a číslo rámce, ve kterém je uložena samotná virtuální stránka
 - V hlavní paměti je vždy „top level tabulka“ (úr. 1)
 - Tabulky ostatních úrovní v paměti být nemusí, pokud proces nepoužívá stránky z oblastí VAS, která tyto tabulky pomáhají adresovat – šetří se fyzická paměť
 - Za ušetřené místo platíme pomalejším překladem – pro urychlení se používá společně s TLB
- **Invertovaná tabulka stránek**
- Obsahuje pro každý rámec fyz. paměti jednu řádku, ve které jsou uložené inf:
 - Číslo stránky, která je nahrána do tohoto rámce
 - Číslo procesu, do jehož VAS tato stránka patří
 - Kontrolní byty
 - Present bit – zda je stránka v hlavní paměti
 - Reference bit – zda se ke stránce přistupovalo
 - Modify bit – zda byl obsah modifikován
 - Přístup. Práva
 - Cache disabled/enabled – zda jsou registry periferií mapovány přímo do paměti
 - Index zřetězení (chain)
 - OS si musí udržovat pouze jednu tabulku pro celý systém
 - Číslo stránky se pomocí hašovací funkce přepočte na index do tabulky. Protože počet stránek je větší než počet rámců fyz. paměti, několik různých stránek je větší než počet rámců fyz. paměti, několik různých stránek se může namapovat na stejnou řádku v tabulce – proto se používá technika zřetězení
 - Zabírá méně místa než tabulka stránek, ale hledání v této tabulce je pomalé – pro urychlení překladu se používá společně s TLB
- **Translation lookaside buffer (TLB) – pro urychlení překladu v procesorech**
- Implementován jako n-way cache
 - Obsahuje informace o posledně překládaných virtuálních adresách (číslo stránky-číslo rámce)
 - Počet položek TLB je menší, než počet virt. Stránek/počet fyz. rámců
 - Položka TLB obsahuje:
 - Valid bit – zda je platná položka
 - Číslo stránky
 - Číslo rámce
 - ASID – address space ID
 - Control bits

- Pokud aplikace používá/alokuje velkou část ze svého VAS, pak může docházet k velké frekvenci TLB miss (časté přepisování položek v TLB)
- CPU i OS nám umožňují sledovat jak využití TLB, tak i ostatních parametrů „memory managementu“
 - Na základě těchto inf. můžeme změnit nastavení OS nebo chování aplikací
- Jednorozměrný VAS x segmentace
 - Jednorozměrný VAS procesu obsahuje datové struktury s různými vlastnostmi (velikost, typ přístupu), jejich mapování do jednorozměrného VAS je umělé, vnikají problémy při implementaci pomocí dynamických oblastí
 - Řešení
 - Stránkování – řeší problémy v rámci jednorozměrného VAS
 - Segmentace – rozdelení VAS do několika segmentů s různými vlastnostmi, v rámci segmentů se používá stránkování
- **Čistá segmentace**
 - VAS procesu je rozdelen do několika segmentů s různými vlastnostmi impl. pomocí dyn. oblastí
 - Překlad virtuálních adres – tabulka segmentů
 - Položka tabulky:
 - Kontrolní bity
 - Velikost segmentu – length
 - Počáteční adresa segmentu – base
 - Číslo segmentu funguje jako index do tabulky segmentů
 - OS si musí udržovat pro každý proces jednu tabulku segmentů
- **Segmentace se stránkováním**
 - Čistá segmentace – problém nalezení volné oblasti pro nový segment \Rightarrow segmentace se kombinuje se stránkováním a TLB
 - *Mechanismus segmentace*
 - Číslo segmentu = index do tabulky segmentů
 - Položka v tabulce segmentů obsahuje číslo rámce, kde je uložená tabulka stránek/víceúrovňová tabulka stránek/invertovaná tabulka stránek
 - *Mechanismus stránkování*
 - Číslo stránky = index do tabulky stránek, ve které najdeme číslo rámce, kde je uložená přísl. stránka ve fyz. paměti
 - Nejméně významné bity virtuální adresy reprezentují offset
 - *TLB*
 - Položka TLB obsahuje navíc číslo segmentu a jeho atributy
- **Algoritmy pro nahradu stránek**
 - Když je většina/všechny rámce fyz. paměti obsahené, OS musí najít vhodný rámec, jehož obsah se uvolní
 - Požadavky – minimalizovat počet výpadků stránek, rychlosť, iži implementace
 - Princip alg. – k instrukcím/datům ve VAS procesu se nepřistupuje náhodně
 - Instrukce se většinou vykonávají sekvenčně
 - Data uložena blízko sebe, přístup často sekvenčně
 - Platí princip prostorové a časové lokality
- **Optimální algoritmus**
 - Nahradí se stránka, která má čas příštího přístupu nejdelší
 - Generuje minimální počet výpadků stránek
 - Pro porovnání kvality, v praxi nepoužitelný
- **NRU algoritmus (not recently used)**
 - Většina systémů se stránkováním si pro každou stránku pamatuje R bit (reference) a M bit (modified)
 - Při načtení stránky do paměti jsou bity nastaveny na hodnotu 0
 - Tyto bity jsou nastaveny automaticky při každém přístupu ke stránce
 - R bit se nastaví, pokud se ke stránce přistupuje

- M bit se nastaví, pokud se změnil obsah stránky
 - Abychom získali informaci, kde se ke stránce přistupovalo, musí OS periodicky resetovat hodnotu R bitu na 0
 - Na základě hodnot R a M bitů můžeme stránky rozdělit do čtyř tříd (00, 01, 10, 11)
 - NRU nahradí stránku z neprázdné třídy s nejnižším číslem
- **FIFO algoritmus**
- OS udržuje seznam stránek, které se aktuálně nachází v hl. paměti
 - V okamžiku, kdy se stránka nahraje do paměti přidá se její záznam na konec seznamu
 - FIFO vybere první stránku ze seznamu jako vhodného kandidáta pro nahradu
 - Způsobuje relativně velký počet výpadků stránek
- **Clock algoritmus** – modifikovaný FIFO
- Seznam stránek = kruhová fronta
 - Na počátku ručička ukazuje na první položku fronty
 - Pro každou stránku si zapamatuje její R bit
 - Postup při hledání vhodné stránky pro nahradu
 - Pokud ručka ukazuje na stránku s R = 1, resetuje R bit na 0 a ručička se posune
 - Opakuje se, dokud ručička nebude ukazovat na stránku s R = 0 – stránka se nahradí a ručička se nastaví na následující stránku
- **Two-handed clock algoritmus**
- Kruhová fronta a R byty
 - Obě ručičky se otáčejí stejnou rychlosťí
 - U stránky, na kterou ukazuje první ručička, se resetuje R na nulu
 - Pokud stránka, na kterou ukazuje druhá ručička má stále R nulový, je vybrána na nahradu
- **LRU algoritmus**
- Pro nahradu stránka, ke které se nepřistupovalo nejdelší dobu
 - Dobrá aproximace optima, ale blbá implementace
 - Implementace pomocí speciálního hardwarového čítače
- **Aging algoritmus** – softwarová simulace LRU algoritmu
- Pro každou stránku si systém pamatuje R bit, n-bitový čítač C
 - Systém periodicky pro každou stránku
 - Posune obsah C doprava o jeden bit
 - Nastaví hodnotu nejvýznamnějšího bitu C na hodnotu R
 - Resetuje hodnotu R na 0
 - Pro nahradu stránka s nejmenším C

Datové typy v programovacích jazycích. Staticky a dynamicky alokované proměnné, spojové seznamy. Modulární programování, procedury a funkce, vstupní a výstupní parametry. Překladač, linker, debugger

BI-PA1

- **Datové typy** = množina možných hodnot a použitelných operací
 - o Informace uloženy v proměnných, které mohou obsahovat určité typy hodnot – typ je neměnný
 - o Podle typu je třeba také alokovat paměť
 - o U některých jazyků není třeba definovat typ, o přiřazení paměti se stará interní proces
 - o Do paměti se ukládají dle little/big endian (liho/holi)
 - o Primitivní, složené, ukazatele, jiné (např. funkce v C), abstraktní (fronta, pole, ...)
 - o **Primitivní typy**: např char, (un)signed int, long, float, short, double, bool
- o **Kontejnery**: vektor, mapa, list...
- o Programátor si může deklarovat vlastní datové typy (class, struct...)
- o **Proměnná** = pojmenovaný datový objekt obsahující nějakou hodnotu, která se může měnit
 - Jsou identifikovány jmény – jednoznačnými identifikátory
 - Lokální x globální
- o **Deklarace** – definuje význam identifikátoru (proměnná, funkce, konstanta, ...)
- o **Inicializace** – přiřazení konkrétní hodnoty
- o **Globální proměnné**
 - Vždy inicializovány nulovými bajty
 - Alokovány stále – použitelné pouze pro data, která program potřebuje uchovávat po celou dobu běhu
 - Alokovány v datovém segmentu komplátorem – alokace jedinou v době komplikace, stejná po celý běh programu
 - Pro konstanty a předpočítané tabulky, které se během výpočtu nemění
 - Fixní velikost při komplikaci – nevhodné pro data s velkým rozpětím velikosti
- o **Lokální proměnné**
 - Alokace až za běhu programu
 - Prostor pro proměnné je vytvořen v momentu, kdy je třeba
 - Proměnné alokovány až při volání funkce nebo vstupu do bloku
 - Po návratu paměťový prostor uvolněn a může být použit jinou funkcí – šetření paměťovým prostorem
 - **Alokace na zásobníku** (stack) – režim LIFO (last in, first out)
 - Když je funkce zavolána, její proměnné jsou alokovány na vrcholu zásobníku
 - Když funkce skončí (návratem do volající), vrchol zásobníku se vrátí do stavu před zavoláním funkce – prostor, který zabíralo volání funkce se tím uvolní pro nové využití

- **Ukazatel** = proměnná obsahující odkaz na datovou buňku v paměti
 - o Hodnota ukazatele na typ T reprezentuje adresu v paměti, kde je hodnota typu T uložena
 - o Deklarace ukazatele: T^* ptr
 - o Proměnná ptr reprezenuje adresu – stejně jako u jiných lokálních proměnných je počáteční hodnota nedefinována
 - o Inicializován adresou – $T^* p = \&x;$ nebo mimo deklaraci $p=\&x;$
 - Unární prefixový operátor & = **reference** (odkaz)
 - Výsledkem operace $\&x$ je ukazatel (adresa) proměnné x
 - o Ukazatel je datového typu T^* – ukazatel na hodnoty typu T
 - o **Vlastní adresa** = celé číslo udávající pozici proměnné v paměti (počítáno od počátku paměťového prostoru)
 - o **Dereference** – pro přístup k proměnné, jež je referencována ukazatelem, musí být použit operátor dereference *
 - Když ptr je ukazatel typu T^* , pak výraz $*ptr$ označuje hodnotu referencovanou tímto ukazatelem ptr
 - o Ukazatel **NULL** – speciální hodnota adresy, která nic nereferencuje
 - o **Aritmetika ukazatelů** – k ukazateli je možno přičítat a odečítat celočíselnou hodnotu, ukazatele je možno dále odečítat a porovnávat
 - Přehled povolených operací:

$T^* + int \rightarrow T^*$	$T^* - int \rightarrow T^*$
$T^* - T^* \rightarrow int$	$T^* \text{ rel op } T^* \rightarrow int$
 - Když je celé číslo n přičteno k ukazateli T^* , výsledná adresa je zvětšena o $n * \text{sizeof}(T)$ bajtů
 - Jestliže jsou dva ukazatele odečteny, je výsledkem je celé číslo, jehož hodnotou je počet prvků typu (T) mezi ukazateli
 - Výsledek porovnání ukazatelů je dán hodnotami adres
 - o Hlavní použití ukazatelů
 - Předání výstupních (referencovaných) parametrů funkci
 - Předání polí jako parametrů funkci
 - Manipulace s polí (zejména s řetězci)
 - Přístup k dynamicky alokované paměti
- **Reference** = odkaz na proměnnou nebo instanci objektu
 - o Abstraktnější varianta ukazatele
 - o Neobsahuje žádnou informaci o uložení objektu v paměti
- **Staticky alokované proměnné** – definuje předem daný počet bytů – neměnný
 - o **Alokace na zásobníku**, vzniká deklarací
 - **Zásobník** – vyhrazen v horní části paměti
 - Malá počáteční velikost, další přidána podle potřeby
 - Velmi omezená maximální hloubka
 - Překročení limitu vede k pádu programu
 - o Vhodné pro:
 - Alokaci jednotlivých proměnných primitivních typů
 - Malá pole s omezením velikosti známým v době komplikace, které není potřeba předávat volajícímu
 - Malé struktury, které není potřeba předávat volajícímu

- **Dynamicky alokované proměnné** – nevznikají deklarací, ale dynamicky v době běhu programu, pomocí speciálních funkcí (operátorů)
 - o Dynamicky alokovaná proměnná nemá jméno, pro přístup k proměnné slouží její adresa – **ukazatel**
 - o Dopředu nelze říci, kolik místa daná informace zabere (např. pole) – místo se přiřazuje v době běhu programu
 - o **Alokace na haldě** – části paměti, která je využívána pro alokaci dyn. paměti
 - o Programátor se musí o paměť starat (new, free) nebo čekat na garbage collector
 - o Nutno použít, když: v době komplikace neznáme velikost pole, nebo je pole příliš velké, pro alokaci velkých struktur, když potřebujeme pole vytvořit ve funkci a předat ji volajícímu
 - o Bloky paměti je potřeba před ztrátou reference **uvolnit** (a to právě jednou)
 - o Kdy použít dynamickou paměť
 - V době komplikace neznáme velikost pole
 - Známe velikost pole, ale pole je veliké, tedy se nemusí vejít na zásobník jako lokální proměnná
 - Pro alokaci velkých struktur
 - Potřebujeme pole (či strukturu) vytvořit ve funkci a předat ji volajícímu
- **Spojové seznamy** – dynamicky alokovaná struktura, kde každý prvek obsahuje hodnotu a ukazatel na další prvek (pokud oboustranný, tak i na předechozí)
 - o Cyklický nebo poslední prvek končí odkazem NULL
 - o Ukazatel aktualizován při každém přidání prvku na konec seznamu
 - o Přidání O(1), odebrání a find O(n) v případě neseřazeného
 - o **Zarázka** = prvek navíc na konci seznamu, neobsahuje užitečná data, zjednoduší některé operace
 - o Prázdný seznam – jediný prvek – zarázka
 - o Seřazený spojový seznam – vkládání prvků kamkoli v seznamu – O(n) – při vložení nutno modifikovat
 - o Ukazatele
- **Modulární programování** = jeden ze způsobů strukturování programů
 - o Program je rozdělen do několika zdrojových souborů (modulů), ty se dají vyměňovat
 - o **Modul** = část programu, která poskytuje určitou funkcionality pro zbytek programu, obvykle zpracovává několik souvisejících úkolů, black box s veřejným interfacem
 - o Modul = specifikace (seznam zdrojů poskytovaných modulem) + implementace (implementace zdrojů)
 - o Lepší udržovatelnost, přehlednost a logické oddělení
 - o Každý modul má vlastní funkce, třídy, metody... jiné využití
- **Funkce** = podprogram řešící dílčí problém a vracející hodnotu na základě vstupních proměnných
 - o Svůj výsledek předává ve formě výstupních parametrů
 - o Při volání je vytvořen aktivační záznam na zásobníku
 - o **Obecný tvar:** typ jméno (seznam parametrů)
 - Typ – návratový typ funkce
 - Jméno – identifikátor funkce
 - Seznam parametrů – seznam formálních parametrů, každý ve tvaru typ jméno, parametry jsou odděleny čárkou (funkce bez parametrů používají místo seznamu klíčové slovo void)
 - o **Parametry** = lokální proměnné funkce
 - o **Tělo funkce** = blok, jež je proveden, když je funkce zavolána
 - o Hodnota výrazu – výsledek předaný volající funkci
 - o **Parametry funkce** (formální parametry) – lokální proměnné této funkce
 - Před vlastním zavoláním funkce jsou výrazy reprezentující skutečné parametry vyhodnoceny a tyto hodnoty vytvoří počáteční hodnoty formálních parametrů
 - Při předání parametrů se mohou provádět typové konverze, jako při regulérním přiřazení
 - Formální parametr je při volání naplněn hodnotou skutečného parametru

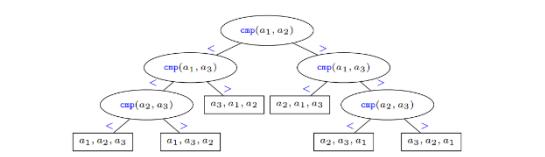
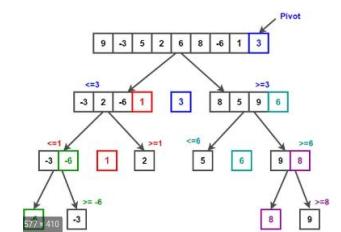
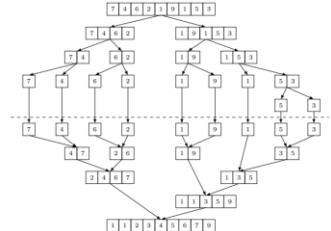
- Deklarace funkce – hlavička funkce následovaná středníkem
 - Deklarované funkce mohou být použity v programu
 - Definice funkce – tělo funkce – může být dodáno později – z jiného modulu/knihovny
- **Procedura** = podprogram řešící dílčí problém bez návratové hodnoty (vrací void) – jediný rozdíl od funkce
 - V C jsou procedury funkciemi s návratovou hodnotou typu void
- **Vstupní parametry** = proměnné, které vstupují do konkrétní funkce
 - Mají svůj název (většinou i typ) a mohou mít defaultní hodnotu
 - Kopírují se nebo se předávají ukazatelem (kopíruje se adresa na buňku)
 - Nemohou předat hodnotu z volané do volající funkce
 - Může se lišit typ skutečného a formálního parametru (např. skutečný parametr typu int je zkonzervován na formální parametr typu double)
- **Výstupní parametry**
 - Proměnné, které vystupují z funkce
 - Když má mít funkce výstupní parametr typu T, deklaruje (vstupní) parametr typu T*
 - Pro přístup k proměnné je třeba ukazatel dereferencovat pomocí *
 - Volající předává ukazatel výstupní proměnné místo její hodnoty, adresa prom. Je získána pomocí &
 - Datové typy ukazatelů se musí přesně shodovat
- **Překladač (kompilátor)**
 - Nástroj pro překlad vstupního (vyššího) jazyka do jazyka nižšího
 - Nejčastěji překlad do strojového jazyka
 - Optimalizuje kód
 - V případě existence reference na jiné soubory se předávají do linkeru
- **Linker**
 - Řeší reference mezi jednotlivými moduly, soubory a knihovnami
 - Propojuje zkompilované soubory do funkčního celku
 - Výstup je spustitelný binární soubor
- **Debugger**
 - Nástroj pro ladění kódu
 - umožňuje třeba krokování v programu
 - Slouží k lepšímu porozumění chování programu

Časová a paměťová složitost algoritmů. Algoritmy vyhledávání (sekvenční, půlením intervalu), slučování a řazení (BubbleSort, SelectSort, InsertSort, MergeSort, QuickSort). Dolní mez složitosti řazení v porovnávacím modelu. Řazení v lineárním čase

BI-PA1 + BI-AG1

- Složitost algoritmů
 - o Časová složitost (v nejhorším případě) pro vstup velikosti N je maximum z počtu vykonaných instrukcí přes všechny možné vstupy velikosti nejvýše N
 - Neměří se v sekundách, ale v počtu potřebných operací
 - Nezávislá na implementaci, jazyku, zařízení
 - 3 případy – nejlepší, průměrný, nejhorší
 - o Paměťová složitost programu pro vstup velikosti N je maximum z počtu použitých paměťových buněk přes všechny možné vstupy velikosti nejvýše N
 - Z hlediska paměti lze rozdělit algoritmy:
 - In-place – pracují s konstantní pamětí, která jsou nutná pro uchovávání dat
 - Out-of-place – potřebují místo navíc
- Asymptotický odhad – zobecněný odhad složitosti
 - o Uvažujeme pouze členy nejvyššího rádu
 - o Vynecháváme multiplikativní konstanty
 - o Kompletně vynecháváme členy nižších rádů
- Algoritmy vyhledávání
 - o Sekvenční – jednoduché procházení celé struktury a porovnání každé hodnoty s vyhledávanou
 - Složitost je $O(n)$, protože při nejhorším musí projít všechny prvky
 - o Půlením intervalu – binární vyhledávání – velikost problému půlí v každé iteraci
 - Vyžaduje seřazenou strukturu
 - Rychlosť je $O(\log n)$, protože vždycky dělí velikost intervalu na polovinu
- Algoritmy slučování a řazení
 - o Stabilní/nestabilní: podle toho, jestli zachovává pořadí prvků se stejným klíčem
 - o Datově citlivé algoritmy – časová složitost závisí na hodnotách prvků
 - o Bubble sort – porovnání všech sousedních prvků pole
 - Algoritmus:
 - Pokud jsou dva sous. prvky pole v nesprávném pořadí, zamění se hodnoty těchto pr.
 - Opakování porovnávání sous. prvků, až všichni sousedé budou ve správném pořadí
 - Časová složitost: $O(n^2)$
 - Paměťová složitost: in-place; stabilní, citlivý
 - o Select sort – opakovaný výběr nejmenšího prvku
 - Algoritmus:
 - Naleze nejmenší prvek ve zbytku pole
 - Zamění nejmenší prvek s prvním prvkem (z neseřazené části)
 - Opakuje s polem o jeden prvek kratším
 - Časová složitost: $O(n^2)$
 - Paměťová složitost: in-place; nestabilní, necitlivý
 - o Insert sort – třídění přímým vkládáním
 - Algoritmus:
 - Začátek je první prvek pole – pole délky 1 je seřazeno
 - Vezme se první prvek zbytku pole a zařadí se do již seřazeného pole
 - Časová složitost: $O(n^2)$
 - Paměťová složitost: in-place; nestabilní, necitlivý

- To se opakuje pro prvky s indexy 2, 3, 4, ..., dokud pole není seřazeno
 - Časová složitost: $O(n^2)$
 - Paměťová složitost: in-place; stabilní, citlivý
- **Merge sort** – efektivní řadící algoritmus založen na metodě Rozděl a panuj
 - Algoritmus
 - Rekurzivně rozděluje pole na poloviny, dokud nezůstane pole o velikosti jednoho prvku (tato pole jsou již seřazena)
 - Rekurzivní seřazování obou částí
 - Sloučí sousední dohromady (stylem vždycky vyber menší prvek ze dvou polí do nového zmergovaného pole)
 - Časová složitost: $O(n \log n)$
 - Paměťová složitost: $O(n)$; stabilní, necitlivý
- **Quick sort**
 - Oproti Merge sortu malá multiplikativní konstanta, nepotřebuje žádný prostor navíc, ale v nejhorším případě složitost $O(n^2)$
 - **Algoritmus:**
 - Zvolí se pivot (ideálně medián)
 - Většinou neznámý – zvolení např. mediánu z náhodně vybraných 3 prvků
 - Vstup se rozdělí na levou ($<$ pivot), střední (= pivot) a pravou ($>$ pivot) část
 - Pokud je dobrý pivot, pravá a levá část by měly být stejně velké
 - Obě části se řadí rekurzivně (střední je seřazena)
 - Výsledná posloupnost vznikne spojením částí za sebou
 - Časová složitost: záleží na vstupu, nejhorší $O(n^2)$, jinak $O(n \log n)$; nestabilní, citlivý
- **Dolní mez složitosti řazení v porovnávacím modelu**
 - **Předpoklady**
 - Čísla se smí pouze vzájemně porovnat a přesouvat v paměti
 - Porovnání $\text{cmp}(ai, aj)$ je binární operace, která dokáže porovnat prvky
 - Algoritmus je deterministický a nepoužívá zdroj náhody
 - **Dolní mez vyhledávání = $\Omega(\log n)$** operací porovnání
 - Žádný deterministický algoritmus v porovnávacím modelu nemůže v nejhorším případě nalézt dané číslo v n -prvkové seřazené posloupnosti použitím $O(\log n)$ operací porovnání
 - Binární vyhledávání s $O(\log n)$ je optimální
 - **Dolní mez řazení = $\Omega(n \log n)$** porovnání
 - Každý deterministický algoritmus v porovnávacím modelu, který seřadí n -prvkovou posloupnost, použije v nejhorším případě $\Omega(n \log n)$ porovnání.
 - Postupuje se sestavením rozhodovacího stromu
- **Řazení v lineárním čase** – speciální řadící algoritmy, které nepracují v porovnávacím modelu
 - **Couting sort** – řazení n celých čísel z množiny r
 - Omezení rozsahu vstupních hodnot je právě ta speciální vlastnost
 - **Algoritmus:**
 - Projde se vstupní pole a spočítá pro každé číslo z množiny r , kolikrát se ve vstupním poli vyskytuje (vypočte histogram)
 - Nad tímto polem počtů výskytů pak vypočte **prefixovým součtem** pozice, kde budou po seřazení začátky oblastí tvořených stejnými čísly z množiny r
 - Projde podruhé vstupní pole, pro každý prvek určí jeho pozici, na které se má nacházet po seřazení a na tuto pozici ho přesune



Příklad rozhodovacího stromu T_S pro $n = 3$ a nějaký algoritmus S .

<i>x</i> :	1 5 6 1 4 7 6 4 6 2	<i>i</i>	<i>pocet</i> :	1 1 1 0 2 1 3 1
<i>vystup</i> :	1 1 2 4 4 5 6 6 6 7		<i>zacatek</i> :	1 2 2 3 5 6 9 10

- Časová složitost: $\Theta(n + r)$
 - Paměťová složitost: $\Theta(n + r)$
 - Korektně řadí celá čísla z množiny r a je stabilní
 - Není in-place ani datově citlivý
- **LexCounting sort** – úkolem je seřadit k-tice slovníkově (lexikograficky)
 - Využívá se Counting sort
 - Algoritmus:
 - Použij CountingSort pro seřazení podle souřadnice (OD POSLEDNÍ)
 - Iteruj pro předposlední atd. až k první
 - Časová složitost: $\Theta(k \cdot (n + r))$
 - Paměťová složitost: $\Theta(kn + r)$
 - Na stejném principu je založen RadixSort – řazení vícečíferných čísel
 - Řadí vstupní k-tice správně lexikograficky a je stabilní

$n = 9, r = 3, k = 4$				
Vstup	$i = 4$	$i = 3$	$i = 2$	$i = 1$
(3,2,1,2)	(1,2,3, 1)	(3,2, 1 ,2)	(2,1,1,2)	(1,1,3,2)
(2,1,3,3)	(3,2,1,2)	(2,1,1,2)	(1,1,3,2)	(1,2,3,1)
(3,3,2,2)	(3,3,2,2)	(2,3,1,3)	(2,1,3,3)	(2,1,1,2)
(1,2,3,1)	(1,1,3,2)	(3,3,2,2)	(3,2,1,2)	(2,1,3,3)
(1,1,3,2)	(2,2,2,2)	(2,2,2,2)	(2,2,2,2)	(2,2,2,2)
(2,3,1,3)	(3,2,3,2)	(1,2,3,1)	(1,2,3,1)	(2,3,1,3)
(2,2,2,2)	(2,1,1,2)	(1,1,3,2)	(3,2,3,2)	(3,2,1,2)
(3,2,3,2)	(2,1,3,3)	(3,2,3,2)	(2,3,1,3)	(3,2,3,2)
(2,1,1,2)	(2,3,1,3)	(2,1,3,3)	(3,3,2,2)	(3,3,2,2)

Rekurzivní rozklad problému na podproblémy metodou Rozděl-a-panuj. Rekurze vs iterace. Dynamické programování

BI-PA1 + BI-AG1

- **Rozděl a panuj** – postup řešení problému nad vstupními daty při kterém:
 - o Se stejný postup aplikuje na část nebo na části vstupních dat
 - o Současně se poskytne přímé řešení triviální instance problému
 - o Řešení celého problému se sestaví z řešení podproblémů
- Řešení dané instance problému se tedy staví na stejném řešení jedné nebo více menších instancí téhož problému (tzn rozdělí se to na menší až triviální a lépe zpracovatelné podproblémy)
- **Výhody** – úspornost zápisu kódu, přirozenost, intuitivnost (explicitní pojmenování toho, co se opakuje v menším) a expresivnost (snadné vyjádření složitosti)
- např. klasické operace s binárními stromy (**BVS**, **AVL**), **MergeSort**, **QuickSort**...
- Algoritmy využívající Rozděl a panuj:
 - o **MergeSort** (viz. předchozí otázka) – řazení sléváním seřazených podposloupností
 - Posloupnost o 1 prvku už je seřazena
 - Mějme vstupní neseřazenou posloupnost n prvků pro $n \geq 2$
 - Rozdělíme ji na dvě části poloviční délky
 - Rekurzivním voláním téhož algoritmu na obě poloviny je seřadíme
 - Obě seřazené poloviny posléze slijeme dohromady do jedné seřazené posloupnosti (operace Merge) a máme to
 - o **Karacubův algoritmus** – rychlejší násobení dvou čísel
 - čísla jsou rozdělená na 2 části stejně délky

Platí tedy

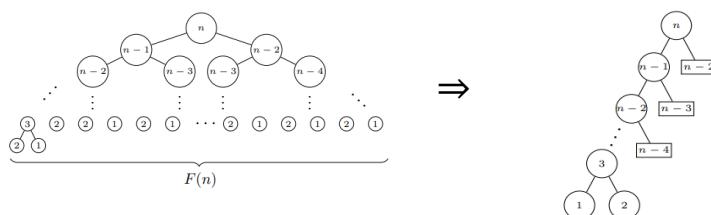
$$\begin{aligned}x &= a \cdot 10^{n/2} + b, \\y &= c \cdot 10^{n/2} + d.\end{aligned}$$

```
Algoritmus Merge(x1, ..., xm; y1, ..., yn)
(1) i := 1, j := 1, k := 1
(2) Dokud i ≤ m a j ≤ n, opakujeme:
    (3) Pokud xi ≤ yj: //přesuneme prvek z x
        zk := xi, i := i + 1
    (5) Jinak: //přesuneme prvek z y
        zk := yj, j := j + 1
    (7) k := k + 1
    (8) Pokud i ≤ m: //přesuneme zbytek x
        zk, ..., zm+n := xi, ..., xm
    (10) Pokud j ≤ n: //přesuneme zbytek y
        zk, ..., zm+n := yj, ..., yn
    (12) Vrat z1, ..., zm+n
```

- pro nějaká $(n/2)$ -ciferná čísla a, b, c, d .
- ▶ Pro výpočet $(ad + bc)$ jsme potřebovali 2 násobení čísel poloviční délky.
 - ▶ Triviálně platí $ad + bc = (a + b)(c + d) - ac - bd$.
 - ▶ Rekurzivní způsob násobení lze tedy přepsat na tvar
- $$xy = ac \cdot 10^n + ((a + b)(c + d) - ac - bd) \cdot 10^{n/2} + bd$$
- Není třeba 4x násobit, stačí $3x + 2x$ sčítat (což je lineární složitost)
 - Časová složitost: $\Theta(n^{\log 3})$ (oproti $O(n^2)$)
- o **QuickSelect** – hledání k -tého nejmenšího prvku
 - Postup jako u QuickSortu
 - Výběr pivota jako skoromediánu (číslo z prostředních dvou čtvrtin seřazené posl., šance 50%)
 - Rekurzivně se hledá vlevo/vpravo, pokud není ve střední části (porovnají se velikosti množin)
 - Časová složitost: $O(n)$
- **Rekurze** – rekurzivní funkce je funkce, která v sekvenci příkazů obsahuje svoje volání
 - o musí existovat konečná podmínka, kdy má rekurze končit
- **Iterace** – opakování přes cyklus, který se ukončí splněnou podmínkou, jinak je nekonečná
- **Typy rekurzí**
 - o Koncová – neprovádějí se již další “odložené” operace, snadno převoditelná na iteraci a naopak (např. GCD)
 - o Lineární – jedno nebo více disjunktních rekurzivních volání a provede se jen jedno (např. QuickSelect)
 - o Stromová/kaskádní – má v sobě aspoň dvě rekurzivní volání, které se za určitých okolností provedou, obvyklé pro Rozděl a panuj (např. Fibonacci)

- Převod mezi iterací a koncovou rekurzí
 - o Iterační/podmíněný cyklus se nahradí podmíněným příkazem
 - o Úprava iterátoru na konci těla cyklu se nahradí rekurzivním voláním na konci podmíněného příkazu

BubbleUpIter(p) (1) Dokud p není kořen: (2) $o := \text{otec}(p)$ (3) Pokud $k(o) \leq k(p)$: (4) return (5) prohoď $k(o)$ a $k(p)$ (6) $p := o$	BubbleUpRec(p) (1) Pokud p není kořen: (2) $o := \text{otec}(p)$ (3) Pokud $k(o) \leq k(p)$: (4) return (5) prohoď $k(o)$ a $k(p)$ (6) BubbleUpRec(o)
---	--
- Dynamické programování – další technika návrhu algoritmů, která je založená na rekurzivním rozkladu problému na podproblémy
 - o Na rozdíl od Rozděl a panuj, metoda DP využívá toho, že se podproblémy během rekurzivního rozkladu opakují
 - o Podmínkou je opakování výskyt stejných podproblémů
 - o Memoizace – zapamatování výsledků již spočtených podproblémů, které se mohou použít příště, až algoritmus narazí na stejný podproblém
 - výrazně redukuje počet operací k dokončení algoritmu
 - o Princip DP = postup při memoizaci řešení:
 1. Formulujeme řešení daného problému rekurzivně rozkladem na řešení podproblémů
 2. Popíšeme řešení pomocí rekurzivního algoritmu, který má exponenciální složitost
 3. Identifikujeme opakování výpočty stejných podproblémů
 4. Vytvoříme prázdnou tabulku hodnot řešení podproblémů
 5. Vložíme do ní hodnoty řešení triviálních instancí
 6. Před výpočtem řešení podproblému se podíváme do tabulky
 7. Pokud je políčko tohoto podproblému již vypočtené, vezmeme jeho hodnotu
 8. Jinak tento podproblém vyřešíme pomocí volání rekurse
 9. Po návratu z rekurse zapíšeme výsledek do příslušného políčka tabulky
 - o Příklady využití DP:
 - Fibonacciho čísla – výpočet $n-2, n-3$, atd. se neustále opakuje pro jednotlivé větve
 - V DP proto tento podproblém počítáme jen jednou a ukládáme např. do pole.



- Nejdelší rostoucí posloupnost – pro každou buňku si pamatujeme jeho nejdelší posloupnost
 - Pokud na ní navazujeme, tak už jen přičítáme a nemusíme kontrolovat nic jiného
 - Iterujeme od konce, aby seděla čísla
 - Přidáme na začátek $-\inf$, aby se nám tam uložil nejlepší výsledek

$x:$

0	1	2	3	4	5	6	7	8
$-\infty$	6	5	2	4	13	8	11	3

$T:$

0	1	2	3	4	5	6	7	8
5	3	3	4	3	1	2	1	1

Objektově orientované programování v C++, zapouzdření, dědičnost, atributy a metody, statické atributy a metody, virtuální metody, polymorfismus, abstraktní třídy, výjimky, šablony, přetěžování operátorů, mělká a hluboká kopie

BI-PA2

- **OOP v C++**
 - o Určení entit, analýza jejich interakcí a implementace entit v podobě tříd a jejich metod
 - o Zakládá se na představě reálného světa, kdy existují objekty s různými vlastnostmi
 - o **Objekt** – drží informaci – stav, který je dán obsahem jeho datových položek (atributů)
 - Instance třídy (takže proměnná)
 - V průběhu výpočtu vytvářeny a rušeny
 - Při vytvoření zavolán konstruktor, ty mohou být přetíženy
 - o **Interface objektu** – definuje chování objektu
 - Množina operací (metod), které má objekt k dispozici
 - o **Třída objektu** – definuje interface a datové položky (atributy) objektu'
 - Popis datového typu – jméno + data (členské proměnné) + interface (členské funkce)
- **Zapouzdření** – omezení přístupu ke členům třídy
 - o **Public** (dostupné komukoliv)
 - o **Private** (přístup jen ve třídě samé)
 - o **Protected** (přístup ve třídě samé a v jejích podtřídách – dědění)
- **Dědičnost** – schopnost tříd deklarovat jinou třídu jako nadřazenou (rodiče) a dědit její chování
 - o Potomci mohou rozšířit chování rodiče o další metody a atributy
 - o Mohou metody rodiče přepsat
 - o Lze vytvářet hierarchie tříd
- **Odvozené třídy** (podtřídy) – syntaxe `class T1 : public T {...}`
 - o Členské proměnné v odvozené třídě – existující zděděny a nemohou být odebrány, jejich datové typy nemohou být změněny, mohou být přidány nové proměnné
 - o Metody v odvozených třídách – existující zděděny, mohou být přepsány (stejná signatura, jiná implementace), mohou být přidány nové
 - o Instance odvozené třídy může být přiřazena předkovi, ale ne naopak
- **Atributy** – proměnné, které si objekt uchovává pro následnou práci
 - o Každá instance má vlastní instanční proměnné
 - o Deklarované bez klíčového slova
 - o **Statické** (static) – nejsou součástí objektu, ale třídy, tento atribut je sdílený napříč všemi instancemi třídy
- **Metody** – mohou být volány na existující objekt
 - o Mají přístup k instančním metodám a atributům
 - o **Statické** (static) – podobné funkcím, nejsou součástí konkrétního objektu, jsou přístupné bez jakékoliv instance, mají přístup k třídním proměnným i metodám
- **Virtuální metody** – základ polymorfismu
 - o Není závislá na typu ukazatele na objekt, rozpoznává deklarovanou instanci třídy a volá overridnutou metodu
 - o Musí mít dynamickou vazbu, ta určuje volanou metodu až v době výpočtu, na základě objektů, které jsou zpracovávány
 - o Např. je třída Person s metodou `print()`, která vypisuje "Jsem člověk". Od ní dědí Man a Woman, přepisují `print()`, aby vypisovala "Jsem muž" a "Jsem žena".

- Pokud deklaruju Person * m = new Man() a zavolám m->print(), tak se zavolá virtuální metoda třídy Man a vypíše "Jsem muž"
- **Polymorfismus** – schopnost objektů se stejnou specifikací mít rozdílnou implementaci
 - Musí mít dynamickou vazbu
 - Schopnost objektů se stejnou specifikací mít rozdílnou implementaci
 - Objekty poskytují stejné rozhraní, ale jiné chování na základě vyděděné třídy (viz příklad výš)
 - **Overriding** – předefinování již implementovaných parent metod na nové se zachováním rozhraní.
 - **Parametrický polymorfismus** – volání metody se stejným názvem na základě poskytovaného počtu a typech parametrů.
- **Abstraktní třídy** – třída, která obsahuje alespoň jednu abstraktní metodu (deklarovanou, ale neimplementovanou, = 0)
 - Využití při dědičnosti
 - Nelze takovou třídu deklarovat
 - Všechny podtřídy implementují tento interface
- **Šablony** – parametrisovaná deklarace a definice funkce/třídy
 - Mohou parametrisovat datové typy svých parametrů nebo návratové hodnoty
 - Generický parametr (T) je datový typ, který je komplátorem substituován, když vzniká nová instance generické třídy
 - Využití: pokud předem neznáme datový typ, případně chceme udělat funkce se stejným chováním pro více datových typů
 - Např. T max (T a, T b)
 - Implicitní deklarace max(1,2) → int max (int a, int b)
 - Explicitní deklarace max<char>(1, 2) → char max (char a, char b)
- **Výjimky** – neočekávaná situace, která přeruší příkazy
 - Speciální chybový signál procházející řetězem volání od volaného k volajícímu
 - Vyhazují se objekty typu Error, ty se musí zachytávat *try-catchem*, pokud se neodchytí, program spadne
 - V případě odchycení jsme schopni opravit program, aby pokračoval v běhu
 - Např. vyhození výjimky dělení 0
- **Přetěžování operátorů**
 - Přetěžovány mohou být jen existující operátory (+, +=, < ...)
 - Musí zůstat zachována arita (+= musí být binární)
 - Na základě přetěžování se dá definovat nové chování (např. můžu sečíst dvě instance třídy), nelze nahradit původní
 - Může se využívat **friend** funkcí. Ty jsou deklarovány ve třídě a samotná definice funkce je mimo třídu. Tato funkce má přístup k private a protected atributům a metodám. (např << pro ostream, aby se mohl výstup řetězit)
- **Mělká a hluboká kopie**
 - Nechť existuje instance objektu A, která má několik atributů staticky alokovaných na stacku (přímo data) a několik dynamicky alokovaných na heapu (ukazatelé).
 - Při **mělké kopii** instance A se kopíruje objekt a jeho přímo uložené hodnoty (hodnoty proměnných), tzn. pro dynamicky alokované proměnné se kopíruje pouze ukazatel. Ve výsledku dva objekty dynamicky ukazují na stejnou paměť.
 - Při **hluboké kopii** se např. v kopírovacím konstruktoru definuje i způsob kopírování dynamických proměnných. Dochází i ke kopii paměti na heapu, instance jsou pak nezávislé.
 - ve zkratce: mělká kopie = kopie ukazatelů, sdílená data, hluboká kopie = kopie všeho, nezávislé
- **Statická vazba** – rychlejší, ale neumožňuje polymorfismus, metoda vybrána v době komplikace
 - Volající musí rozlišit objekty a jejich typy, mít znalost všech možných odvozených tříd a starat se o implementační detaily objektů, které používá

- **Dynamická vazba** – určuje volanou metodu až v době výpočtu, na základě objektů, které jsou zpracovány
 - o Pomalejší volání, volající zpracovává objekty a provádí operace, nemusí při nich rozlišovat typy objektů
 - o VMT – virtual method table – k určení metody v konstantním čase – připravena pro každou třídu s virtuálními funkcemi – jsou v ní jen virtuální metody
 - Generována komplátorem
 - Obsahuje pole adres metod s dynamickou vazbou, každý objekt s dynamickou vazbou má ukazatel na svou VMT

Abstraktní datový typ, jeho specifikace a implementace. Zásobník, fronta, pole, seznam, tabulka, množina. Implementace pomocí pole, spojových struktur a stromů

BI-PA2

- **Abstraktní datový typ** – definuje množinu hodnot a množinu operací
 - o Může být formálně specifikován signaturou operací (definice syntaxe) a množinou axiomů (definice sémantiky)
 - Axiomy jsou ekvivalence mezi výrazy, každý výraz reprezentuje stav ADT
 - Výrazy jsou složeny z operací a proměnných
 - Axiomy mohou být použity pro zjednodušení komplexnějších operací
- **Implementace**
 - o Některé programovací jazyky (Clear) dovolují formální specifikaci ADT
 - o Imperativní a OOP vyžadují explicitní implementaci ADT
 - o V C++ je doporučeno ADT implementovat generickými třídami:
 - Typ prvku je generickým parametrem šablony třídy
 - Operace *init* implementována konstruktorem
 - Dynamicky alokovaná paměť (proto je třeba destruktor, copy-konstruktor a operátor =)
 - Pokud je signatura výstupu $\rightarrow elem$, doporučuje se const metoda vracející T
 - Pokud je signatura výstupu $\rightarrow ADT$, doporučuje se metoda modifikující objekt
- **Zásobník (stack)** – princip **LIFO** (Last In, First Out)
 - o Přistupovat se dá pouze k vrcholu zásobníku, odebírat a přidávat zrovna tak
 - časová složitost těchto operací je konstantní $O(1)$ - *push* amortizovaně kvůli dyn. alokaci $O^*(1)$
 - o Implementace:
 - Pole pevné délky, kapacita omezena už při komplikaci
 - Dynamicky alokované pole, velikost dána parametrem konstruktoru, opět omezená velikost
 - Dynamicky alokované pole, velikost se mění vždy, když je třeba
 - Jednosměrně zřetězený spojový seznam
- **Fronta (queue)** – sekvenční kontejner organizovaný **FIFO** (First In, First Out) způsobem
 - o Přistupovat se dá pouze k začátku, přidává se na konec
 - o Časová složitost je pro *push* i *pop* konstantní (opět amortizovaně pro dynamicky alokované pole)
 - o Implementace stejná jako u zásobníku
- **Pole (array)** – datový kontejner organizující prvky v n -dimenzionálním prostoru
 - o Prvek je identifikován n -icí indexů
 - o Náhodný přístup k prvkům je časově konstantní, zvětšování (kopírování) pole je $O(n)$
 - o Popis pole:
 - Datový typ prvků T
 - Počet dimenzí n
 - Dolní a horní meze jednotlivých dimenzí l_1 a h_1
 - o *Jednodimenzionální pole*:
 - $array[l_1..h_1]$ typu T
 - prvky jsou v kontinuálním paměťovém bloku, velikost: $(h_1 - l_1 + 1) * \text{sizeof}(T)$
 - mapovací funkce $\text{map}(i) = (i - l_1)$
 - o *Multidimenzionální pole*:
 - $array[l_1..h_1, l_2..h_2, \dots, l_n..h_n]$ typu T

Participující typy:

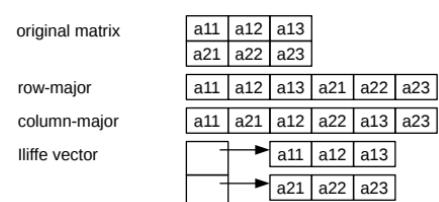
- zásobník – ADT stack,
- elem – prvky v ADT,
- bool – boolské hodnoty.

Signatury:

```
init:           -> stack
empty(_):     stack -> bool
push(_,_):    stack,elem -> stack
top(_):        stack -> elem
pop(_):        stack -> stack
```

Axiomy:

```
empty(init)    = true
empty(push(s,x)) = false
top(init)      = error
top(push(s,x)) = x
pop(init)      = init
pop(push(s,x)) = s
```



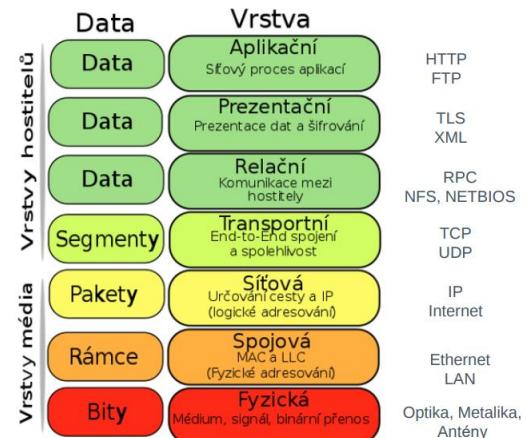
- **Seznam (list)** – datová struktura, která poskytuje operace *insert*, *remove* a *read*
 - o Operace jsou určeny pozicí v seznamu, ta se může měnit
 - o Vložení a rušení s konstantní časovou složitostí (nalezení je $O(n)$)
 - o Implementace:
 - Řetězec struktur (jednosměrný, obousměrný, cyklický)
 - Nelze implementovat polem, muselo by se kopírovat a posouvat v případě vkládání, to by nesplňovalo podmínky konstantního vkládání a odebrání
- **Tabulka (map)** – kontejner obsahující dvojici klíč-hodnota, kde klíč je unikátní
 - o Základní operace – init, insert, delete, isSet, read
 - o Implementace:
 - Pole, kde jsou klíče s indexy v poli (klíče jsou celočíselné) - na daném místě v poli se pak nachází hodnota, třeba odlišení validních a nevalidních prvků, vše časově konstantní
 - (ne)seřazené pole - vše $O(n)$, klíč musí být porovnatelný, seřazený isSet a read je $O(\log n)$
 - Obsahuje (ne)seřazené páry {key; val}
 - Jednosměrný (ne)seřazený spojový seznam
 - Binární vyhledávací strom
 - Hash tabulka
- **Množina (set)** – kontejner obsahující prvky typu T bez duplikátů
 - o Operace: insert, remove, test přítomnosti prvku a množinové operace
 - o Speciální případ mapy, kde hodnoty jsou typu bool
 - o Implementace:
 - Indikátorový (charakteristický) vektor – pokud je universum dostatečně malé a konečné, udělá se bool vektor nebo bitové pole (0 - není, 1 - je), vše je pak časově konstatní
 - (ne)seřazené pole - vše $O(n)$, kromě neseřazený add $O(1)$ a seřazený find $O(\log n)$
 - Jednosměrný (ne)seřazený spojový seznam – vše $O(n)$
 - Binární vyhledávací strom
 - Hash tabulka

ISO/OSI a TCP/IP model. Protokoly linkové vrstvy. Přepínání a směrování. Principy fungování propojovacích sítových prvků

BI-PSI

ISO/OSI

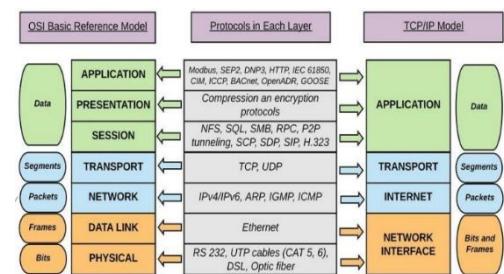
- Teoretický model pro popis sítí a protokolů pro komunikaci
- Nikdy nebyl plně implementován
- Zapouzdření: každá vrstva doplňuje svoje "metadata" k datům, aby se mohly potom znova úspěšně dekódovat
- **Fyzická vrstva – bity** – přenos bitů kanálem
 - o Definuje způsob přenosu 0 a 1
 - o Optika, kabely...
- **Spojová (linková) vrstva – rámce** – formátuje data do rámců
 - o Funkce spolehlivého spojení (detekuje a opravuje chyby)
 - o Formátuje data do rámců
 - o **Rámce** = hlavička, data vyšších vrstev a konec rámce
 - o Řízení přístupu k lince (medium access control – MAC) a toku na ní
 - o Jednoznačná adresa v segmentu sítě (MAC v Ethernetu)
 - o Bridge, switche
 - o MTU – maximum transfer unit – maximální možná velikost přenášených dat
 - o VLAN – virtual local area network – k tomu, aby uměly oddělovat toky v sítích
 - Protože přehlednost, bezpečnost, efektivita, jednoduchost
 - Lze použít jeden přepínač, který se nastaví tak, aby pracoval jako dva nezávisle oddělené přepínače
 - Dělení dle portů, adres, protokolu, značky
 - o MAC a LLC (fyzické adresy)
 - **LLC** = logické řízení toku, kódování, spolehlivost doručení, hardwarově nezávislá, kontr. Chyb
 - Úkol – řízení toku a kontrola chyb (detekční a samoopravné kódy)
 - Mechanismus existence různých protokolů nad společnou MAC adresou (IP, IPX, ...)
 - Rozdělení toku dat z vyšších vrstev do rámců – **framing**
 - Určování velikosti a konce rámce
 - **MAC** = přístup k médiu, multiplex, fyzická adresa, hardwarově závislá
 - Identifikace zařízení v počítačové síti, 48 bitů – pro komunikaci uvnitř 1 segmentu lokální sítě
 - Sdílený přístup k médiu – pomocí multiplexu a přístupových metod
 - Předávání dat fyzické vrstvě
 - Ukládání rámců do front a jejich odesílání
 - Vytváření virtuálních sítí (VLAN)
 - **Multiplex** = technika umožňující současně použití stejného přenosového média více účastníků (stanicemi, počítači) – mnohonásobný přístup
- **Síťová vrstva – pakety** – adresace a směrování dat přes mezilehlé prvky (určování adresy a IP – logické adresy)
 - o Jednoznačná adresa v rámci celé sítě (síťová adresa - např. IP adresa)
 - o Routery – přenosí dat po paketech (až 64 kb)
 - o Adresace je hierarchická – vhodná pro směrování
 - o Síťové adresy – použitelné jak v lokální síti tak i mezi různými sítěmi
- **Transportní vrstva – segmenty** – rozklad dat na pakety a uspořádání podle pořadí
 - o Multiplexuje a demultiplexuje data mezi transportními spoji
 - o Předává pakety pomocí TCP, UDP



- Transportní adresy (adresa, port)
- End-to-end spojení a spolehlivost
- **Relační vrstva – data** – vytváření logického rozhraní pro aplikace
 - Synchronizace spojení (transakce)
 - Přihlášení, udržení relace
 - Např. sdílení disků
- **Prezentační vrstva – data** – formátování a prezentace dat
 - Transformace dat (kompresie/dekomprese)
 - Kódování (pro různé jazyky) a šifrování
 - Např. ASCII, UNICODE, TLS, XML
- **Aplikační vrstva – data** – způsob komunikace aplikací (protokoly) – síťový proces aplikací
 - Podpůrné funkce aplikacím
 - Představuje interface pro uživatele
 - Např. HTTP, FTP

TCP/IP

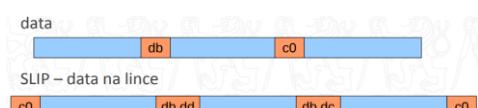
- Oproti ISO/OSI modelu jsou sloučené linková s fyzickou vrstvou
- Odstraněna prezentační a relační vrstva (jen aplikační)
- Vznikl v akademickém prostředí, globálně úspěšný díky internetu
- Vznikl dodatečně, nejprve byly protokoly
- **Protokoly linkové vrstvy**



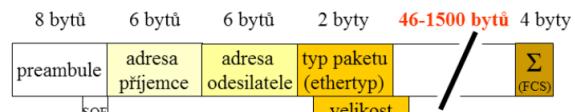
- Zodpovídají za přenos dat mezi propojenými systémy
- Linkování na základě MAC adres (unikátní pro každý síťový prvek – router, karta atd, přiděleno výrobcem, 6x8 bitů)
- Zajišťují spolehlivost přenosu
- **HDLC (High Level Data-Link Control)**
 - Bitově orientovaný protokol
 - Využit v sériových linkách
 - Synchronní i asynchronní přenos



- **SLIP (Serial Line IP)**
 - Definuje pouze zapouzdření paketů na sériové lince
 - Nedefinuje adresaci, typ paketů, detekci chyb, kompresi...
 - Speciální znaky END a ESC
 - Rámec ohraničen znaky END
- **PPP (Point to Point Protocol)**
 - Podmnožina HDLC
 - Asynchronní, bitově i znakově synchronní
 - Bitové linky – bit stuffing



- **Ethernet**
 - Dnes nejběžnější
 - Kroucené linky, optické a koaxiální kabely
 - Různé implementace dle typu rychlosti
 - Různé typy rámců (Ethernet II, IEEE 802.3+IEEE 802.2, SNAP)
 - MAC adresy – 48 bitů, první tři byty unikátní, přiděleny konkrétnímu výrobci
 - Zbylé tři byty – konkrétní sériové číslo daného zařízení
 - Formát rámce Ethernetu
 - Preamble – úvodní hlavička pro identifikaci rámce
 - SOF – start of frame



8 bytů 6 bytů 6 bytů 2 byty 46-1500 bytů 4 byty



Σ (FCS)

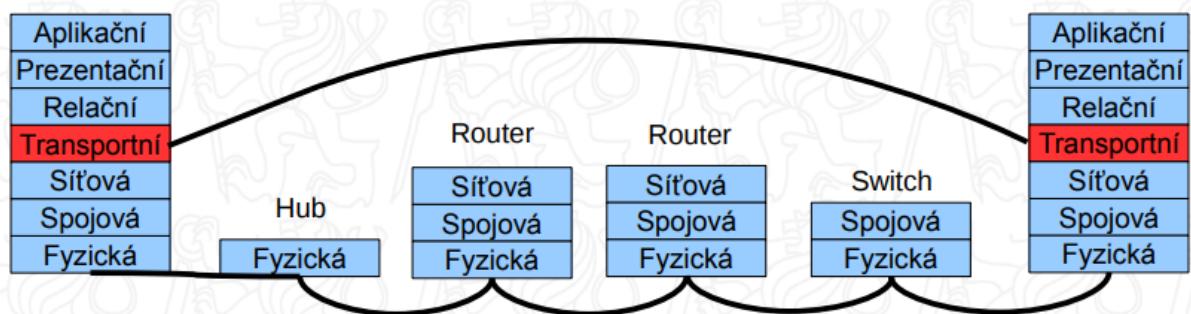
- EtherType – 16 bitů, které specifikují detailněji typ použitého rámce
- FCS – 32 bitů obsahující kontrolní součet rámce umožňující detekci poškozeného rámce
- Velikost přenášených dat – max. 1500 – nastavena v konfiguraci síťového rozhraní
- Umí ovlivnit příchozí tok a zabránit zahlcení sítě – pokud přepínač nestihá, může požádat sousedy o zpomalení vysílání – příkaz PAUSE
- Neobsahuje žádný mechanismus pro garanci spolehlivého doručení dat
- Autonegotiace = metoda, kterou si vysílač a přijímač dohodnou parametry přenosu
- Propojování zařízení – bez přepínačů kříženým kabelem, s přepínači přímým kabelem
- Implementuje VLAN pomocí speciálních rámců – značené (tagované)

Přepínání a směrování

- **Přepínání (switching)**
 - **Switch** = propojovací prvek stanic v jednom segmentu lokální sítě
 - Pracuje v linkové vrstvě
 - Přeposílá datové rámce mezi svými porty na základě MAC adresy – propojení dvou sítí
 - Tabulka dvojic [port, adresa], sám se učí podle MAC odesílatele, záznamy expirují
 - Rámcem se známou cílovou MAC odešle na daný port
 - Rámcem s neznámou cílovou MAC odešle na všechny porty, stejně jako broadcast
 - Aktualizace záznamů v tabulce u přepínačů – **učení** – poprvé se rámcem s neznámou adresou příjemce pošle na všechny porty s výjimkou odkud přišel a zaznamená se, z kterého portu přišla odpověď
 - Snižuje zátěž linek
 - Režimy práce:
 - **Store-and-forward** – rámcem je přijat, analyzován a odeslán, vyšší zpoždění pro delší rámce, poškozené se mohou hned zahodit
 - Přepínač přijme celý rámec
 - **Cut-through** – rámcem je průběžně analyzován, odeslán hned po přijetí cílové adresy, nižší zpoždění, poškozené rámce se odesírají
 - Přepínač načte jen cílovou adresu z hlavičky rámce
 - **Fragment free** – podobný Cut-through, ale z rámce je načteno a překontrolovanou větší množství dat (celá hlavička), schopnost odhalení většího množství chybných rámčů
- **Směrování (routing) =** přeposílání paketů mezi segmenty sítě na základě směrovací tabulky
 - **Router** – směrovač – síťový prvek, který odděluje jednotlivé segmenty sítě
 - Pracuje v síťové vrstvě
 - Zajišťuje komunikaci mezi segmenty
 - Směrovací protokol RIP – DVA (distance vector algorithm) přístup ke směrování
- **Směrovací tabulka** – „rozcestník“ – procházena směrovačem při zpracování každého paketu
 - Z adresy cílové sítě se v tabulce hledá daný záznam
 - Destinace – IP cílové sítě
 - Brána – IP adresa sousedního směrovače směrem do cíle
 - Maska – dle prefixu (cílová síť)
 - Metrika – kvalita cesty
 - Rozhraní – název
 - Přenáší data i mezi sítěmi s odlišnou technologií
 - Proaktivní směrování = používá směrovací tabulku
 - **Záplavové:**
 - Směrovač odešle paket na každou výstupní linku
 - Doručení v nejkratším možném čase
 - Paket se duplikuje exponenciálně

- Velmi neefektivní využití sítě
- **Náhodné:**
 - Paket se odešle do náhodně zvolené linky (nezaručuje konečnou dobu doručení)
 - Doplňek k jiným algoritmům (třeba při přetížení hlavní linky)
- **Statické:**
 - Fixní směrovací tabulka vytvořená podle konfigurace sítě
 - Nereaguje na stav sítě
 - Např. počítač v lokální síti
- **Dynamické:**
 - Směrovací tabulka se mění podle stavu sítě
 - Protokoly a jejich algoritmy:
 - RIP (routing information protocol) – DVA (distance-vector algoritmy) – periodicky vysílaný obsah tabulek sousedům
 - OSPF (open shortest path first) – LSA (link-state algoritmy) – vzájemná informace o stavu linek
 - BGP (border gateway protocol) – PVA (path-vector algoritmy) – směrovače si vyměňují kompletní cesty k cílům
- Způsoby aktualizace:
 - Izolovaně (bez ohledu na ostatní směrovače),
 - Centralizovaně (výpočet tabulek je prováděn centrálně)
 - Decentralizovaně (výpočet pro každý směrovač, přičemž podkladem jsou data ostatních směrovačů)

Principy fungování propojovacích síťových prvků



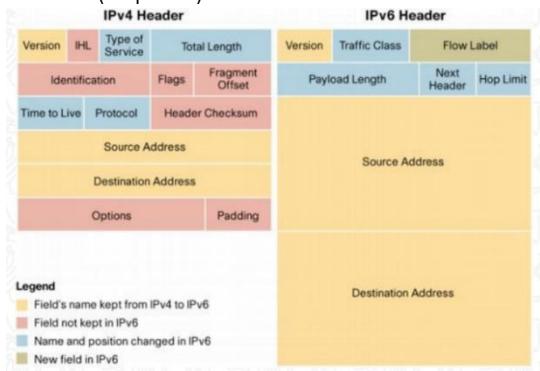
- **Hub** – fyzická vrstva, žádná paměť, posílá na všechny porty
- **Repeater** – fyzická vrstva, zesiluje signál
- **Switch** – linková vrstva, pamatuje si MAC (narozdíl od hubu), odděluje kolizní segmenty
 - Desítky až stovky portů
 - Hardwarová podpora zpracování rámců
 - Vyrovnávací paměť, dovedou regulovat výstupní tok
 - I možnost webového rozhraní, vzdáleného monitorování, ...
- **Bridge** – propojují/oddělují provoz více částí sítě (obsahují přepínací tabulky), softwarová nebo hardwarová realizace
 - Jednoduší, málo portů (2-4), propojují většinou jen dvě sítě, neobsahují vyrovnávací paměť (buffery)
 - Označení i pro softwarová řešení, která spojují několik sítí do jediné
- **Router (směrovač)** - síťová vrstva, směrování paketů po sítích, směruje pakety na základě IP, nešíří broadcast

Protokolová rodina TCP/IP (IPv4, IPv6, TCP, UDP, aplikační protokoly). Řízení datového toku. Princip a využití NAT. Systém DNS

BI-PSI

Protokolová rodina TCP/IP

- **IPv4** – 32-bitová adresa, kvůli nedostatku adres zavedenaIpv6
 - o priv. rozs. adres (neroutují se do internetu): 10.0.0.0/8, 172.16.0.0/12 a 192.168.0.0/16, využití pro NAT
 - o 127.0.0.1 - loopback (smyčka)
 - o **Maska adresy** – určuje adresy používané v síti odvozené od IP, schéma CIDR
 - Rozděluje IP adresu na adresu sítě a adresu stanice (nepoužívá se samostatně)
 - Binární sekvence 1 následovaní sekvencí 0
 - Její velikost určuje rozsah adres v síti – (nejnižší adr. = adr. sítě, nejvyšší adr. = adr. broadcastu) -> celkem $2^n - 2$ přidělitelných adres
 - o **TTL** – *Time to live* – délka životnosti záznamu – každý router sníží hodnotu o 1, ochrana proti zacyklení
 - o **MTU** (maximální délka rámce): definováno linkovou vrstvou, většinou 1500 B
 - o unicast (individuální), broadcast, anycast (výběrové, prostřednictvím BGP) a multicast (skupinové)
 - o **ARP protokol** – zjišťuje MAC adresu podle známé IP adresy
 - Komunikace v rámci jednoho segmentu používá výhradně MAC adresy -> ARP zjišťuje aktuální vazbu mezi IP adresou a MAC adresou
 - Odešle zprávu s dotazem, komu patří daná IP adresa – cílová MAC adresa je broadcast, aby to dostaly všechny rozhraní v segmentu
 - zjišťuje MAC adresy sousedů, mapování mezi síťovou adresou (např. IP) a adresou hardware (linková vrstva, např. MAC)
- **IPv6** – 128-bitová adresa – 2^{96} větší adresní prostor
 - o 8 skupin po 4 hexadecimálních číslicích oddělených dvojtečkou (2001:0db8::1428:57ab)
 - o Hop limit: obdoba TTL
 - o minimální MTU je 1280 B
 - o síťová rozhraní mají více adres
 - o ::1/128 - loopback
 - o NDP (Neighbor Discovery Protocol): rozšířená náhrada ARP
 - o Kompatibilita s IPv4 – NAT64 – překlad IP adres, statický x dynamický
 - o Druhy adres:
 - **Unicastové** – určené jednoznačnému příjemci
 - **Multicastové** – pro určitou skupinu příjemců
 - **Anycastové** – novinka – doručí se jednomu členu určité skupiny (alespoň někomu)
 - Zrušen **broadcast**, nahrazen specifickým multicastem
 - o ICMPv6 – Internet Control Message Protocol – test. chybových stavů, dosažitelnost, výměna provoz. inf.
 - Jednoduchý protokol sloužící k ověření funkčnosti vzdáleného zařízení – utility ping a traceroute
- **TCP (Transmission Control Protocol)** – zajišťuje spojově orientovanou komunikaci na transportní vrstvě
 - o Garance doručení dat ve správném pořadí, **řízení datového toku**
 - o Rozlišení spojení a služeb pomocí zdrojových a cílových portů
 - o Kanál typu point-to-point, který doručuje zprávy v pořadí, v jakém byly odeslány a **bez chyb**
 - o Na začátku spojení dvou stran – **3way handshake** – zpráva, odpověď, potvrzení odpovědi
 - Spojení je definováno:
 - IP adresou klienta
 - Číslem portu klienta
 - IP adresou serveru
 - Číslem portu serveru



- Různé implementace: Reno, Tahoe, Vegas, New Reno, CUBIC, ...
- Duplexní – odesílání z obou stran, každá má své okénko
- Šířka **Congestion Window** – udává maximální objem dat, které může vysílač odeslat najednou, aniž by došlo k zahlcení linky
 - Pokud je šířka klouzavého okna vyšší než CW, posílá se maximálně tolik dat, kolik je CW
 - Hodnotu určuje vysílač
- není šifrovaný, služba spojově orientovaná, zabezpečená, duplexní, kontrola doručení paketů
- v 1 relaci lze přenášet neomezený počet dat
- *zabezpečení*: kontrolní součty, detekce duplicit, opakované odeslání, správné seřazení a timeout
- nezbytné pro nutně zabezpečený datový kanál
- nevhodné pro realtime aplikace (streaming atd.)
- **UDP (User Datagram Protocol)** – cíl rozlišit jednotlivá spojení a služby pomocí číselných portů
 - adresace na transportní vrstvě, streamované datové toky
 - nezajišťuje spojení, nepotvrzuje data, nereguluje datový tok, malé pakety
 - vhodné tam, kde vadí režie TCP (např. stream), kde nevadí ztráta (DNS)
- číslo **portu** v TCP a UDP – u klienta k rozlišení jednotlivých spojení, 0–65535
 - na serveru běží aplikace, která předpokládá, že všechna komunikace s ní bude probíhat na definovaném portu
- **Aplikační protokoly** – server nabízí službu, klient se připojí a službu využívá (alternativa P2P)
 - kooperují s prezenční vrstvou, zejména kvůli šifrování, nebo využívají služeb transportní vrstvy
 - Telnet, **SSH**, RDP = protokoly vzdálené správy – často používány pro „nabourání“ systému
 - **FTP** = protokol pro přenos souborů
 - Režim klient-server, běžně nešifrované (alternativa Secure FTP – využití SSH)
 - Protokoly pro posílání mailů
 - **MTA** = mail transfer agent – program, který běží na poštovním serveru, který doručuje poštu jinému poštovnímu serveru – MSExchange
 - **UA** = user agent – běží v zařízení klienta a načítá poštu z mailboxu na poštovním serveru
 - **SMPT** (doručování mailů mezi poštovními servery), **POP3**, **IMAP**, SMPT relay
 - **HTTP(S)** = hyper-text transfer protocol – pro přenos webových stránek
 - *Cookies* – obsahuje nastavení konkrétního klienta pro daný web, web server upraví obsah odpovědi na míru konkrétnímu uživateli
 - *Cachování, proxy* – dotaz směrován na proxy server, ten přesměrovává
 - NTP = network time protocol – přenos času v síti – Christianův algoritmus
 - BitTorrent protokol – peer-to-peer model – opak modelu klient-server, komunikace přímo mezi klienty, všechny uzly sítě jsou rovnocenný, při nárůstu uživatelů roste přenosová kapacita
 - SIP = session initialization protocol – pro hlasové aplikace, pouze signalizační, pro doručení hlasových dat protokol RTP (real time protocol)
 - **DHCP** – automatická konfigurace síťových rozhraní v 1 segmentu lokální sítě
 - Na žádost klienta mu přidělí IP adresu, výchozí bránu, DNS
 - Pomocí DHCP lze obdržet informace: přidělená IP adresa a maska, IP adresa defaultní brány, IP adresa DNS serveru)
 - Konfigurace – výměna 4 zpráv – Discovery (K) – Offer (S) – Request (R) – Acknowledgement (A)

Řízení datového toku

- Pro řízení toku k přijímači – určuje, jak velké množství dat se posílá najednou bez potvrzení, velikost nastavuje přijímač, 0 = data se neposírají
- Obdobný jako v linkové vrstvě: detekování chybných paketů (hash), samoopravné kódy, plovoucí okénko (odesílání několika paketů najednou a detekce chyb)
- Snaha o nalezení optimálního řešení (nejvyšší rychlosť, nejmenší ztrátovost)

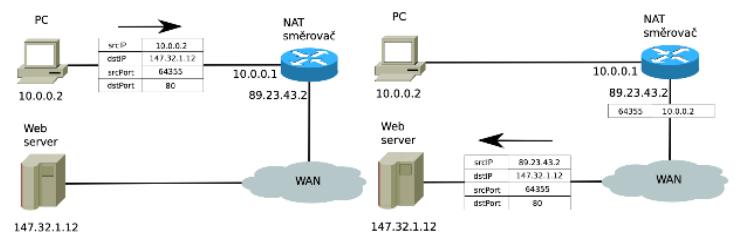
- Algoritmy pro řízení toku
 - o Stop&Wait – jednotlivé potvrzování – doručování dat v pořadí, jak jdou za sebou
 - o Stop&Go – dokáže ovlivnit množství paketů, které přichází od vysílače
 - o Klouzavé okno – sliding window – dovoluje řízení toku úpravou velikosti okna
 - Úprava velikosti okna se optimalizuje podle parametrů přenosu
 - Ztrátovost – packet loss – množství paketů ztracených při přenosu
 - Delay – průměrná doba na doručení paketů mezi zdrojem a cílem
 - RTT – obousměrné zpoždění – průměrná doba doručení paketu od zdroje k cíli a pět
 - Jitter – nestabilita – kolísání zpoždění

- Kontrola zahlcení sítě – kvantita požadavků stoupá, detekce pomocí packet loss nebo zvětšení zpoždění, Token/Leaky Bucket nebo rezervace pásma

- záložní komunikační cesty – pokud dochází k zahlcení jedné cesty, je možné využít jiné

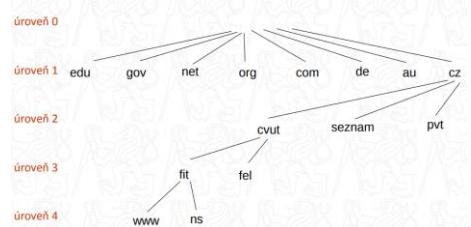
NAT – překlad síťových adres

- odděluje LAN od WAN
- obchází nedostatek IPv4 adres
- NAT směrovač má vlastní veřejnou IP adresu, která vystupuje jako zástupce všech počítačů v lokální síti
- Požadavky ze sítě/do sítě se párují s pomocí portů NAT zařízení
- funguje jako router
- **Masquerade**: všechny zařízení vystupují jako jedno do WAN sítě.
- **Port-Forwarding**: pokud by se chtělo přistoupit do konkrétního počítače z WAN do LAN, tak by zařízení muselo forwardovat jistý port na jednu konkrétní IP adresu v LAN
- **Statický NAT** – překlad jedné IP adresy na jednu IP adresu, pro zpřístupnění služeb schovaných v lokální síti
- **Dynamický NAT** – překlad rozsahu IP adres na jednu IP adresu, když potřebuje více rozhraní z lokální sítě přistupovat na internet – přístup pod adresou, se kterou je směrovač připojen k internetu



Systém DNS – překlad jmen (webů) na IP adresy

- servery, které udržují překladové tabulky
 - o záznam [jméno, typ, hodnota, třída, TTL]
- komunikace přes UDP i TCP
- při vstupu "example.com" se zařízení nejdřív obrací k lokální cache, pak k DNS serveru, aby získal požadovanou IP adresu příjemce, kterou následně může využít.
- **Dynamické DNS** = DynDNS když dojde ke změně adresy, zařízení informuje dydns.com, ta ve svém DNS změní příslušnou IP
- **DNSSEC** = zabezpečené rozšíření DNS tak, aby bylo možné zabránit zfalšování odpovědi – založeno na asymetrické kryptografii – každý DNS server podepisuje odpověď svým soukromým klíčem. Pomocí veřejného klíče, který je uložený v RRSIG záznamu na DNS serveru nadřazené domény je možné provést ověřit
- **Typy serverů**
 - o **Kořenové (root) DNS servery** – top-level domény jako org, cz, com – nad nimi je jediná, kořenová doména
 - o **Autoritativní server** – server, na němž jsou trvale uloženy záznamy k dané doméně/zóně
 - Primární server – hlavní nositel informace – hlavní úl. dat a místo, kde admin provádí úpravy DNS
 - Sekundární server – kopíruje data z primárního serveru, okamžitá synchronizace
 - o **Rekurzivní/caching only DNS server** – server, na který se se svými dotazy obrací klientská zařízení
 - Rekurzivní dotaz – name server, pokud informaci neví, se musí rekurzivně doptat
 - forwarding: předává rekurzivní dotaz (odlehčení linky)
 - o **Iterativní DNS server** – name server vrátí resolveru odkaz na nejbližší autoritativní DNS server, který zná odpověď a resolver musí zbytek zjistit sám



- **Name server** = server/počítač, který se stará o jména a jejich překlad
- **Zóna** = prostor jmen konkrétní části
- **Resolver** = program, který komunikuje s name serverem a žádá o překlad jména na adresu
- FQDN – Fully Qualified Domain Name – spojením názvů všech domén v jeden řetězec, oddělení '.'
 - Např. www.fit.cvut.cz
- **Top-level domény** – Generic Top Level Domains (.com, .net) New Generic Top Level Domains (.dev, .corp), Country Code Top Level Domains (.cz), Reverzní domény (com.stranka místo stranka.com)

Pravidla pro výpočty pravděpodobností, Bayesův vzorec. Náhodné veličiny, příklady rozdělení, distribuční funkce, hustota, momenty. Nezávislost náhodných jevů a veličin.

Centrální limitní věta, zákony velkých čísel

BI-PST

Pravidla pro výpočty pravděpodobností

- **Klasická definice** pravděpodobnosti – počitatelné objekty
 - o Konečný počet n vzájemně různých výsledků nějakého pokusu
 - o Předpoklad, že všechny výsledky jsou stejně pravděpodobné
 - o Jestliže právě m z těchto výsledků odpovídá realizaci jevu A, potom definujeme pravděpodobnost jevu A jako:

$$P(A) = \frac{m}{n} = \frac{\text{počet příznivých výsledků}}{\text{počet všech možných výsledků}}$$

- **Geometrická definice** pravděpodobnosti – plocha, časový úsek
 - o Výsledky nastávají v nějakém geometrickém objektu či množině konečné velikosti
 - o Každý výsledek je stejně pravděpodobný
 - o Pravděpodobnost jevu A = relativní velikost množiny A:

$$P(A) = \frac{\text{velikost množiny příznivých výsledků}}{\text{velikost množiny všech výsledků}} = \frac{\text{velikost } A}{\text{velikost } \Omega}$$

- **Pravděpodobnostní prostor** – experiment: $\varepsilon = (\Omega, \mathcal{F}, P)$
 - o Ω zahrnuje všechny výsledky experimentu
 - o \mathcal{F} = kolekce všech jevů, kterým umíme přiřadit či spočítat pravděpodobnost P
 - o Pravděpodobnost je vždy nezáporná
- **Prostor elementárních jevů Ω** – výběrový prostor – množina všech možných výsledků daného experimentu
 - o **Elementární jev** – libovolný možný výsledek $\omega \in \Omega$
 - o Elementární jevy v Ω (výsledku experimentu) musí být:
 - **Vzájemně exkluzivní**
 - **V souhrnu vyčerpávající** – každý výsledek experimentu je možno interpretovat jako některý elementární jev
 - o Součet pravděpodobností všech elementárních jevů je roven 1
- **Náhodný jev A** – množina elementárních jevů $A \subset \Omega$, které potřebujeme přiřadit pravděpodobnost
 - o Např. „Při dvěma hody mincí padne alespoň jednou hlava“
- **Operace s náhodnými jevy** – všechny klasické množinové operace s fancy názvy
 - o A^c doplněk jevu A – **opačný jev**
 - o $A \cap B$ průnik jevů A a B – **nastává současně** A a B
 - o $A \cup B$ sjednocení A a B – **bud A nebo B** (nebo oba)
 - o $A \setminus B$ rozdíl A a B – A, ale ne B
 - o $A \subset B$ podmnožina – **když A, tak B**
 - o \emptyset prázdná množina – **nemožný jev**
 - o Ω výběrový prostor – **jistý jev**
 - o ω prvek Ω – **elementární jev**
- **σ -algebra** – systém \mathcal{F} podmnožin prostoru Ω , jestliže jsou splněny podmínky:
 - o $\emptyset \in \mathcal{F}$ - obsahuje nemožný jev
 - o Když $A \in \mathcal{F}$, tak $A^c \in \mathcal{F}$ - obsahuje opačný jev (doplněk)
 - o Když $A_1, A_2, \dots \in \mathcal{F}$, tak $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$ - obsahuje spočetné sjednocení
- **Pravděpodobnostní míra** na (Ω, \mathcal{F}) – funkce $P: \mathcal{F} \rightarrow \mathbb{R}$ splňující:
 - o Nezápornost - $\forall A \in \mathcal{F}: P(A) \geq 0$
 - o Normalizace - $P(\Omega) = 1$

- σ -aditivita – když jsou $A_1, A_2, \dots \in \mathcal{F}$ vzájemně disjunktní jevy ($A_i \cap A_j = \emptyset \forall i, j: i \neq j$, tak:

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$$

- Trojice $\varepsilon = (\Omega, \mathcal{F}, P)$ = experiment / pravděpodobnostní prostor
- Z definice: $\forall A \in \mathcal{F}: 0 \leq P(A) \leq 1$

- **Vlastnosti pravděpodobnosti** – nechť A a B jsou náhodné jevy na pravděpodobnostním prostoru s mírou P.

Potom platí:

- $P(\emptyset) = 0$
- Jestliže A a B jsou vzájemně disjunktní, pak $P(A \cup B) = P(A) + P(B)$
- $P(A^c) = 1 - P(A)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Monotonie – pokud $A \subset B$, tak $P(A) \leq P(B)$
- Nechť A_1, A_2, \dots jsou náhodné jevy na pravděpodobnostním prostoru s mírou P. Potom platí:
 - **σ -subaditivita:** $P(\bigcup_{i=1}^{\infty} A_i) \leq \sum_{i=1}^{\infty} P(A_i)$
 - **Princip inkluze-exkluze:** $P(\bigcup_{i=1}^n A_i) = \sum_{\substack{J \subset \{1, 2, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} P(\bigcap_{i \in J} A_i)$
 - Pro 3 jevy: $P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$

Podmíněná pravděpodobnost a nezávislost

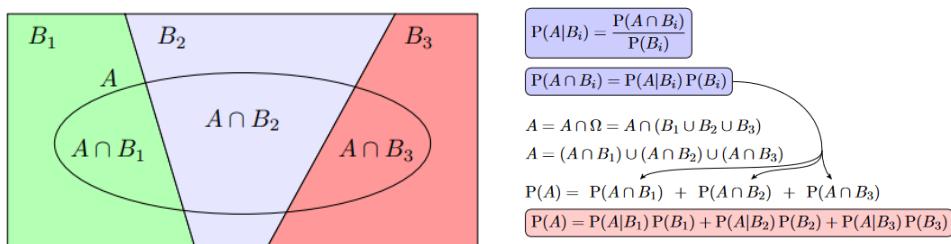
- Podmíněná pravděpodobnost jevu A za podmínky, že nastal jev B, kde A a B jsou náhodné jevy a $P(B) > 0$:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

- **Vlastnosti podmíněné pravděpodobnosti**

- Nechť pro jev B platí $P(B) > 0$. Podmíněná pravděpodobnost $P(\cdot | B)$ je také pravděpodobnostní mírou, tj. $P(\cdot | B) \in [0, 1]$ a splňuje axiomy pravděpodobnosti.
- Podmíněná pravděpodobnost splňuje všechny další vlastnosti pravděpodobnosti:
 - Jestliže A_1 a A_2 jsou vzájemně disjunktní, pak $P(A_1 \cup A_2 | B) = P(A_1 | B) + P(A_2 | B)$
 - $P(A_1 \cup A_2 | B) = P(A_1 | B) + P(A_2 | B) - P(A_1 \cap A_2 | B)$
 - $P(A^c | B) = 1 - P(A | B)$

- **Úplný rozklad pravděpodobnosti:** $\Omega = B_1 \cup B_2 \cup B_3$ – disjunktní rozklad



- Nechť B_1, B_2, \dots, B_n je rozklad Ω takový, že $\forall i: P(B_i) > 0$. Potom pro každý jev A platí:

$$P(A) = \sum_{i=1}^n P(A|B_i)P(B_i)$$

- **Bayesova věta** (prohození jevu a podmínky) – Pozorujeme jev A a ptáme se, jaká je pravděpodobnost, že nastala možnost B_j

- Nechť B_1, B_2, \dots, B_n je rozklad Ω takový, že $\forall i: P(B_i) > 0$. Potom pro každý jev A platí:

$$P(B_j | A) = \frac{P(A|B_j)P(B_j)}{\sum_{i=1}^n P(A|B_i)P(B_i)}$$

- **Multiplikativní zákon** – Pro jevy A_1, \dots, A_n splňující $P(A_1 \cap \dots \cap A_n) > 0$ platí:

$$P(A_1 \cap \dots \cap A_n) = P(A_1)P(A_2 | A_1)P(A_3 | A_1 \cap A_2)P(A_n | A_1 \cap \dots \cap A_{n-1})$$

- **Nezávislost náhodných jevů** – Náhodné jevy A a B se nazývají nezávislé, pokud $P(A \cap B) = P(A)P(B)$. Obecně, soubor jevů $\{A_i : i \in I\}$ se nazývá nezávislý, jestliže $P(\bigcap_{i \in I} A_i) = \prod_{i \in I} P(A_i)$ pro všechny neprázdné konečné podmnožiny J indexové množiny I.
- **Nezávislost náhodných veličin** – Náhodné veličiny X a Y nazýváme nezávislé, pokud pro všechna $x, y \in \mathbb{R}$ jsou jevy $\{X \leq x\}$ a $\{Y \leq y\}$ nezávislé. Tedy pokud pro všechna $x, y \in \mathbb{R}$ platí rovnost $P(X \leq x \cap Y \leq y) = P(X \leq x) \cdot P(Y \leq y)$. Náhodné veličiny X_1, \dots, X_n nazýváme nezávislé, jestliže pro každé $x \in \mathbb{R}^n$ platí rovnost $P(X \leq x) = \prod_{i=1}^n P(X_i \leq x_i)$
- **Podmíněná nezávislost** – Necht (Ω, \mathcal{F}, P) je pravděpodobnostní prostor a C je náhodný jev s $P(C) > 0$. Náhodné jevy A a B se nazývají podmíněně nezávislé za podmínky C, pokud $P(A \cap B|C) = P(A|C)P(B|C)$.

Náhodné veličiny

- **Náhodná veličina** X na pravděpodobnostním prostoru (Ω, \mathcal{F}, P) – **funkce**, která každému výsledku experimentu $\omega \in \Omega$ přiřadí hodnotu $X(\omega) \in \mathbb{R}$ a pro kterou platí podmínka měřitelnosti:

$$\{X \leq x\} \in \mathcal{F}, \forall x \in \mathbb{R}$$
- **Distribuční funkce** náhodné veličiny X – určuje **pravděpodobnostní rozdělení** náhodné veličiny, určena:

$$F_X(x) = P(X \leq x)$$
- **Vlastnosti distribuční funkce** F náhodné veličiny X
 - F je neklesající: když $x < y$, pak $F(x) \leq F(y)$
 - F „začíná v 0 a končí v 1“: $\lim_{x \rightarrow -\infty} F(x) = 0$ a $\lim_{x \rightarrow \infty} F(x) = 1$
 - F je spojitá zprava: $\lim_{y \rightarrow x^+} F(y) = F(x)$
- **Použití distribuční funkce** – pomocí distribuční funkce F náhodné veličiny X můžeme vyjádřit:
 - $P(X > x) = 1 - F(x)$
 - $P(X \in (x, y]) = P(x < X \leq y) = F(y) - F(x)$
 - $P(X < x) = \lim_{y \rightarrow x^-} F(y)$
 - $P(X = x) = F(x) - \lim_{y \rightarrow x^-} F(y)$
- **Diskrétní náhodná veličina** – náhodná veličina X se nazývá diskrétní, jestliže nabývá pouze hodnot z nějaké nejvýše spočetné množiny $\{x_1, x_2, \dots\}$.
 - Pravděpodobnosti možných hodnot náhodné veličiny X jsou $P(X = x_k), k = 1, 2, \dots$
 - Nenulovou pravděpodobnost mají pouze hodnoty x_k , kterých náhodná veličina X nabývá
 - Pravděpodobnost $P(X = x)$ lze chápat jako funkci x, kterou pak nazýváme pravděpodobnostní funkcí nebo **diskrétní hustotou** náhodné veličiny X
- **Distribuční funkce** diskrétní náhodné veličiny X:

$$F_X(x) = P(X \leq x) = \sum_{k: x_k \leq x} P(X = x_k)$$
 - „stoupající“ skoky v bodech x_k , mezi nimi je konstantní, spojitost zprava
 - Velikost skoku v bodě x_k je $P(X = x_k)$
- **Normalizační podmínka** – musí vždy platit, ověřujeme při přiřazování pravděpodobností hodnotám x_k :

$$\sum_{all x_k} P(X = x_k) = 1$$

- Příklady rozdělení – diskrétních náhodných veličin

Bernoulliho (Alternativní) rozdělení s parametrem p , $0 \leq p \leq 1$, $X \sim \text{Be}(p)$:
(jiná běžná značení $X \sim \text{Bernoulli}(p)$, $X \sim \text{Alt}(p)$)
(Jeden hod nevyvážnou minci s pravděpodobností hlavy p)

$$P(1) = p, \quad P(0) = q = 1 - p, \quad E X = p, \quad \text{var } X = p(1 - p).$$

Binomické rozdělení s parametrem p , $0 \leq p \leq 1$, $X \sim \text{Binom}(n, p)$, $\text{Bin}(n, p)$:
(Počet hlav v n hodech nevyvážnou minci.)

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad E X = np, \quad \text{var } X = np(1-p).$$

Geometrické rozdělení s parametrem p , $0 < p < 1$, $X \sim \text{Geom}(p)$:

(Počet hodů nevyvážnou minci, než padne první hlava.)

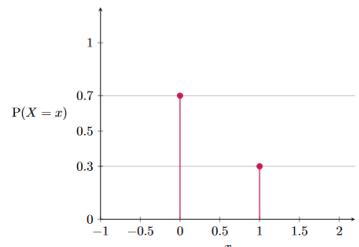
$$P(X = k) = (1-p)^{k-1} p, \quad k = 1, 2, \dots, \quad E X = \frac{1}{p}, \quad \text{var } X = \frac{1-p}{p^2}.$$

Poissonovo rozdělení s parametrem $\lambda > 0$, $X \sim \text{Poisson}(\lambda)$:

(V jistém smyslu limity binomického pro $n \rightarrow \infty$.)

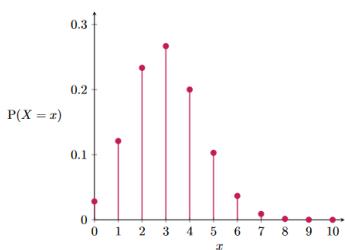
$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad k = 0, 1, 2, \dots, \quad E X = \text{var } X = \lambda.$$

Bernoulliho rozdělení – graf pravděpodobnosti
Pravděpodobnosti hodnot Bernoulliho rozdělení pro $p = 0.3$



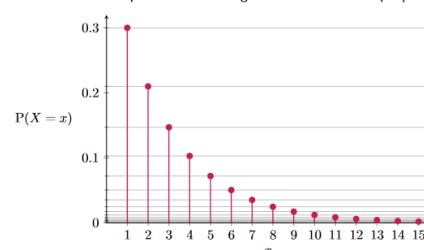
Binomické rozdělení – graf pravděpodobnosti

Pravděpodobnosti hodnot binomického rozdělení pro $n = 10$ a $p = 0.3$



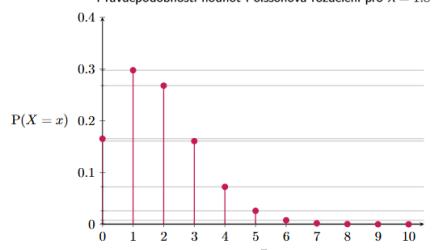
Geometrické rozdělení – graf pravděpodobnosti

Pravděpodobnosti hodnot geometrického rozdělení pro $p = 0.3$



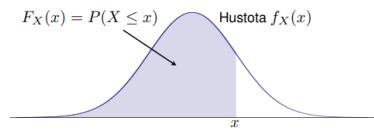
Poissonovo rozdělení – graf pravděpodobnosti

Pravděpodobnosti hodnot Poissonova rozdělení pro $\lambda = 1.8$



- **Spojité náhodné veličiny** – náhodná veličina X se nazývá (absolutně) spojitá, jestliže existuje nezáporná funkce f_X taková, že pro každé $x \in \mathbb{R}$ můžeme distribuční funkci F_X vyjádřit jako

$$F_X(x) = \int_{-\infty}^x f_X(t) dt$$



- f_X = hustota pravděpodobnosti náhodné veličiny X
 - Hustota = derivace distribuční funkce
- Distribuční funkce spojité náhodné veličiny je spojitá

- **Vlastnosti spojitých náhodných veličin** – pro hustotu f_X spojité náhodné veličiny X platí:

- $\int_{-\infty}^{+\infty} f_X(x) dx = 1$ – normalizační podmínka
- $P(X = x) = 0$ pro všechna $x \in \mathbb{R}$
- $f_X(x) = \frac{dF_X}{dx}(x)$ v bodech, kde má F_X derivaci
- $P(a < X \leq b) = \int_a^b f_X(x) dx = F_X(b) - F_X(a)$
- $P(X \in B) = \int_B f_X(x) dx$ pro všechny B v Borelovské σ -algebře na \mathbb{R} (tzn. množiny $\{X \in B\}$ jsou měřitelné, tj. pro všechny „běžné“ množiny

- **Příklady rozdělení – spojitých náhodných veličin**

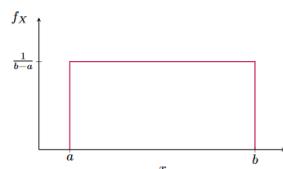
Rovnoměrné rozdělení na intervalu $[a, b]$, $X \sim \text{Unif}(a, b)$:

$$f_X(x) = \frac{1}{b-a}, \quad x \in [a, b] \quad E X = \frac{a+b}{2}, \quad \text{var } X = \frac{(b-a)^2}{12}.$$

Exponenciální rozdělení s parametrem $\lambda > 0$, $X \sim \text{Exp}(\lambda)$:

$$f_X(x) = \lambda e^{-\lambda x}, \quad x \in [0, \infty) \quad E X = \frac{1}{\lambda}, \quad \text{var } X = \frac{1}{\lambda^2}.$$

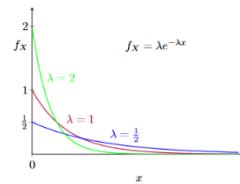
Rovnoměrné rozdělení – graf hustoty



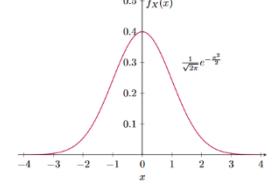
Normální (Gaussovo) rozdělení s parametry $\mu \in \mathbb{R}$ a $\sigma^2 > 0$, $X \sim \mathcal{N}(\mu, \sigma^2)$:

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in (-\infty, \infty) \quad E X = \mu, \quad \text{var } X = \sigma^2.$$

Exponenciální rozdělení – graf hustoty



Standardní normální rozdělení $\mathcal{N}(0, 1)$ – graf hustoty



- Střední hodnota μ
 - Diskrétní náhodné veličiny X , nabývající hodnoty x_1, x_2, \dots , je definována vztahem:
$$EX = \sum_k x_k P(X = x_k)$$
 - Spojité náhodné veličiny X s hustotou f je určena vztahem:
$$EX = \int_{-\infty}^{\infty} xf(x) dx$$
 - Vážený průměr všech možných hodnot, rovnovážný bod pravděpodobností či hustoty, popisuje „střed“ rozdělení, hodnotu z příštího pokusu očekáváme poblíž EX
- Vlastnosti střední hodnoty:
 - Je-li $X \geq 0$, pak $E(X) \geq 0$
 - Je-li $a, b \in \mathbb{R}$, pak $E(aX + b) = aE(X) + b$
 - Konstantní náhodná veličina $X = c \in \mathbb{R}$ má střední hodnotu rovnou příslušné konstantě $E(X) = c$
 - Linearita střední hodnoty - $E(X + Y) = EX + EY$
- Rozptyl σ^2 (variance) náhodné veličiny X:
 - $$\text{var}X = E[(X - EX)^2]$$
 - $$\text{var}X = E(X - EX)^2 = EX^2 - (EX)^2$$
- Směrodatná odchylka σ náhodné veličiny X:
 - $$sdX = \sqrt{\text{var}X}$$
- Vlastnosti rozptylu:
 - $\forall a, b \in \mathbb{R}$ a náhodnou veličinu X platí: $\text{var}(aX + b) = a^2 \text{var}X$
 - Náhodná veličina konstantně rovná $c \in \mathbb{R}$ má rozptyl $\text{var } c = 0$

- Obecné momenty náhodné veličiny
 - Pro $k \in \mathbb{N}$, pírozené číslo, definujeme **k-tý moment** náhodné veličiny X jako:
$$\mu_k = E(X^k) = \begin{cases} \sum_{\text{all } x_i} x_i^k P(X = x_i) & \text{pro } X \text{ diskrétní,} \\ \int_{-\infty}^{\infty} x^k f_X(x) dx & \text{pro } X \text{ spojitolou.} \end{cases}$$
 - Podobně, k-tý centrální moment je:
$$\sigma_k = E(X - EX)^k = \begin{cases} \sum_{\text{all } x_i} (x_i - \mu_1)^k P(X = x_i) & \text{diskrétní} \\ \int_{-\infty}^{\infty} (x - \mu_1)^k f_X(x) dx & \text{spojitá.} \end{cases}$$

Zákony velkých čísel

- Aritmetický průměr: $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$,
- Součet: $S_n = \sum_{i=1}^n X_i$,
- **Markovova nerovnost** – Nechť X je náhodná veličina s konečnou střední hodnotou. Potom platí:
$$P(|X| \geq a) \leq \frac{E|X|}{a} \quad \forall a > 0$$
 - Horní odhad toho, že pravděpodobnost, že jev bude větší než a je menší roven podílu střední hodnoty ku a
- **Čebyševova nerovnost** – Nechť X je náhodná veličina s konečnou střední hodnotou a konečným rozptylem. Potom platí:

$$P(|X - EX| \geq \epsilon) \leq \frac{var X}{\epsilon^2} \quad \forall \epsilon > 0$$

- Přesnější, než Markovova
- Charakteristiky průměru náhodných veličin

- **Střední hodnota** průměru \bar{X} :

$$E\bar{X}_n = E \frac{1}{n} \sum_{i=1}^n X_i = \frac{1}{n} E \sum_{i=1}^n X_i = \frac{1}{n} \sum_{i=1}^n EX_i = \frac{n\mu}{n} = \mu$$

- **Rozptyl** průměru \bar{X} :

$$var \bar{X}_n = var \frac{1}{n} \sum_{i=1}^n X_i = \frac{1}{n^2} var \sum_{i=1}^n X_i = \frac{1}{n^2} \sum_{i=1}^n var X_i = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

- **Slabý zákon velkých čísel** – Nechť X_1, X_2, \dots jsou i.i.d. náhodné veličiny s konečnou střední hodnotou $EX_i = \mu$ a konečným rozptylem σ^2 . Potom \bar{X}_n konverguje k μ v pravděpodobnosti:

$$\bar{X}_n \xrightarrow{P} \mu \quad pro n \rightarrow \infty$$

To znamená, že $\forall \epsilon > 0$ je $\lim_{n \rightarrow \infty} P(|\bar{X}_n - \mu| \geq \epsilon) = 0$.

- *intuitivně*: máme náhodné veličiny, které jsme měřili každý den. Jejich střední hodnota každý den je vždy $EX_i = \mu$ a konečná, rozptyl je konečný. Pak aritmetický průměr konverguje v pravděpodobnosti ke střední hodnotě v nekonečnu $n \rightarrow \infty$. Tzn přidáváme další dny, kde hodnota se blíží ke středu.

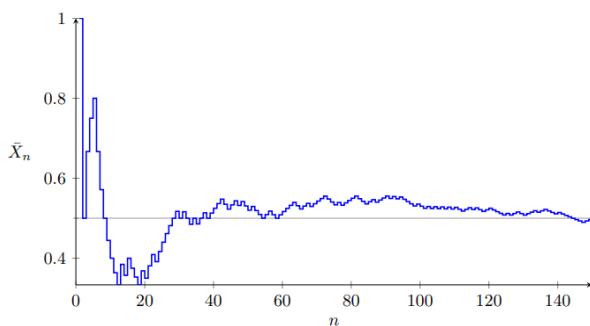
- **Silný zákon velkých čísel (SZVČ, SLLN)** – Nechť X_1, X_2, \dots jsou i.i.d. náhodné veličiny se střední hodnotou $EX_i = \mu$ (ne nutně konečnou). Potom \bar{X}_n konverguje k μ skoro jistě:

$$\bar{X}_n \xrightarrow{a.s.} \mu \quad pro n \rightarrow \infty$$

To znamená, že množina $\omega \in \Omega$, kde $X_n(\omega)$ konverguje jako číselná posloupnost, má pravděpodobnost 1:

$$P(\{\omega \in \Omega : \bar{X}_n(\omega) \rightarrow \mu \text{ pro } n \rightarrow \infty\}) = 1$$

- Silnější v tom, že rozptyl, ani střední hodnota nemusí být konečný
- Konvergence skoro jistě implikuje konvergenci v pravděpodobnosti



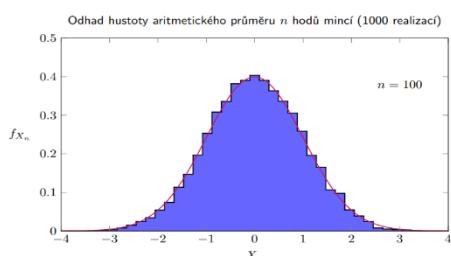
- Ilustrace – čím více hodů mincí, tím více konverguje ke střední hodnotě

Centrální limitní věta – CLV

- Průměr pro velká n představuje dobrý odhad střední hodnoty.
Tzn. střední hodnota lze chápat jako ideální průměr nekonečného množství pokusů. CLV říká, že za jistých podmínek lze dané rozdělení approximovat normálním rozdělením, nejvíc hodnot bude v okolí střední hodnoty.
- CLV – Nechť X_1, X_2, \dots je posloupnost i.i.d. náhodných veličin s konečnou střední hodnotou $EX_i = \mu$ a konečným rozptylem $var X_i = \sigma^2 > 0$. Potom

$$\frac{\bar{X}_n - \mu}{\sqrt{n}} \xrightarrow{D} N(0,1) \quad pro n \rightarrow \infty$$

Podobně



$$\frac{S_n - n\mu}{\sigma\sqrt{n}} \xrightarrow{D} N(0,1) \text{ pro } n \rightarrow \infty$$

- Pro součet a průměr náhodných veličin:

$$\begin{aligned}\mathbb{E} S_n &= n \cdot \mu, & \mathbb{E} \bar{X}_n &= \mu, \\ \text{var } S_n &= n \cdot \sigma^2, & \text{var } \bar{X}_n &= \sigma^2/n.\end{aligned}$$

- Standardizovaný průměr/součet:

$$\begin{aligned}Z_n &= \frac{S_n - \mathbb{E} S_n}{\sqrt{\text{var } S_n}} = \frac{\bar{X}_n - \mathbb{E} \bar{X}_n}{\sqrt{\text{var } \bar{X}_n}} \\ &= \frac{S_n - n\mu}{\sqrt{n\sigma^2}} = \frac{\bar{X}_n - \mu}{\sqrt{\sigma^2/n}},\end{aligned}$$

- CLV říká, že když vezmeme standardizovaný průměr/součet, výsledná náhodná veličina konverguje ke standardnímu normálnímu rozdělení.
- pravděpodobnost lze určit pomocí distribuční funkce Φ standardního normálního rozdělení (v tabulkách):

$$\mathbb{P}(Z_n \leq z) \xrightarrow{n \rightarrow \infty} \mathbb{P}(Z \leq z) = \Phi(z).$$

Základy statistické indukce, náhodný výběr, bodové odhady pro střední hodnotu a rozptyl, intervalové odhady pro střední hodnotu, testování statistických hypotéz o střední hodnotě

BI-PST

- **Statistická indukce** – metoda, která dovoluje induktivně odhadnou chování celku na základě jeho vybraných částí (náhodný výběr)
- Kroky statistického uvažování:
 - Vybereme naměřené hodnoty
 - Odhadneme tvar rozdělení – vychází ze znalosti, intuitivní odhad
 - zúžení úvah na třídu rozdělení F_θ určenou neznámým parametrem θ (např. binomické rozdělení)
 - Odhadneme parametry rozdělení (normální má dva – střední hodnotu a rozptyl)
 - **bodový odhad**: určení nejpravděpodobnější hodnoty θ
 - **intervalový odhad**: určení intervalu, ve kterém θ s danou vysokou pravděpodobností leží
 - Ověříme správnost modelu – testování hypotéz
 - **testy dobré shody**: ověřujeme hypotézy o tvaru pravděpodobnostního rozdělení (např. jestli má normální rozdělení)
 - **parametrické testy**: vytvoříme hypotézu o parametru θ (např. $\theta = 0$) a na základě dat se snažíme rozhodnout, jestli je možné tuto hypotézu zamítнуть nebo ne
- **Náhodný výběr** z rozdělení $F = n$ -tice stejně rozdělených nezávislých náhodných veličin (i.i.d.) X_1, \dots, X_n s distribuční funkcí F
 - **realizace náhodného výběru**: n -tice konkrétních pozorovaných čísel x_1, \dots, x_n (např. měření výšky n lidí)
- **Bodový odhad** parametru $\theta =$ funkce $\hat{\theta}_n = (X_1, \dots, X_n)$ náhodného výběru, která nezávisí na hodnotách θ
 - Aproximace hodnot parametru θ z naměřených dat, měl by být konzistentní a nestranný
 - **Nestrannost** – odhad $\hat{\theta}_n$ parametru θ se nazývá nestranný (nevychýlený), jestliže $\forall \theta \in \Theta: E\hat{\theta}_n = \theta$
 - Odhad není zatížen systematickou chybou, odhad. hodnotu systematicky nepřestřeluje, ani nepodhodnocuje
 - **Konzistence** – odhad $\hat{\theta}_n$ parametru θ se nazývá konzistentní, jestliže $\hat{\theta}_n \xrightarrow{P} \theta$ pro $n \rightarrow \infty$
 - Volbou velkého n lze učinit chybu dostatečně malou
 - Pokud $E\hat{\theta}_n \rightarrow \theta$ a $var\hat{\theta}_n \rightarrow 0$, pak je $\hat{\theta}_n$ konzistentní
 - Nejvyužívanější bodové odhady
 - **Výběrový průměr** – bodový odhad střední hodnoty EX – aritmetický průměr nasbíraných dat
 - **Výběrový rozptyl** – bodový odhad rozptylu $varX$ – součet rozptylů dělený počtem $n-1$ (aby byla zajištěna nestrannost)
 - Výběrová směrodatná odchylka, k -tý výběrový moment, výběrová kovariance, výběrový korelační koeficient
 - **Momentová metoda** – odhadujeme $\theta \in \mathbb{R}^d$
 1. Spočítáme teoretické momenty EX, EX^2, \dots, EX^d
 2. Vyjádříme parametry jako funkce momentů
 3. Odhadneme momenty EX^n výběrovým průměrem
 4. Dosadíme a získáme odhady parametrů
 - **Metoda maximální věrohodnosti**
 - X_1, \dots, X_n – náhodný výběr z rozdělení s hustotou $f(x)$, resp pravděpodobnostmi $P(X = x)$
 - Věrohodnostní funkce: $L(\theta) = \begin{cases} \prod_{i=1}^n f_\theta(x_i) & \text{pro spojité} \\ \prod_{i=1}^n P_\theta(X_i = x_i) & \text{pro diskrétní} \end{cases}$
 - Hledáme θ , pro které je $L(\theta)$ max. (vyplatí se hledat $\max l(\theta) = \ln L(\theta) - \log. věrohodnost$)

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i.$$

$$s_n^2 = s_X^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2.$$

1. Spočítáme hustotu/pravděpodobnost
2. Spočítáme věrohodnostní funkci $L(\theta)$
3. Spočítáme logaritmickou věrohodnost $l(\theta) = \ln L(\theta)$
4. Maximalizace - $l'(\theta) = 0$ (derivujeme podle θ → získáme $\hat{\theta}$)

- Interval spolehlivosti

Definice

Nechť X_1, \dots, X_n je náhodný výběr z rozdělení určeného parametrem θ . Interval (L, U) s krajními body určenými statistikami $L \equiv L(\mathbf{X}) \equiv L(X_1, \dots, X_n)$ a $U \equiv U(\mathbf{X}) \equiv U(X_1, \dots, X_n)$ splňující

$$P(L < \theta < U) = 1 - \alpha$$

se nazývá $100 \cdot (1 - \alpha)\%$ interval spolehlivosti, případně konfidenční interval.

Statistika L resp. U se nazývá dolní resp. hornímez intervalu spolehlivosti.

Číslo $(1 - \alpha)$ se nazývá hladina spolehlivosti.

- jinak řečeno: máme jistotu $100 \cdot (1 - \alpha)\%$, že parametr θ se v tomto intervalu skutečně nachází (lze sestavit i pouze dolní/horní interval spolehlivosti)

- Intervalové odhady pro střední hodnotu

- potřebujeme intervalový odhad (L, U) , ve kterém leží střední hodnota. Takový interval se nazývá interval spolehlivosti
- Známý rozptyl

Věta

Uvažujme náhodný výběr X_1, \dots, X_n z normálního rozdělení $N(\mu, \sigma^2)$ a předpokládejme, že známe rozptyl σ^2 . Oboustranný $100 \cdot (1 - \alpha)\%$ interval spolehlivosti pro μ je

$$\left(\bar{X}_n - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}, \bar{X}_n + z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \right),$$

kde $z_{\alpha/2} = \Phi^{-1}(1 - \alpha/2)$ je kritická hodnota standardního normálního rozdělení, tj. číslo, pro které platí $P(Z > z_{\alpha/2}) = \alpha/2$ pro $Z \sim N(0, 1)$.

Jednostranné $100 \cdot (1 - \alpha)\%$ intervaly spolehlivosti pro μ jsou pak

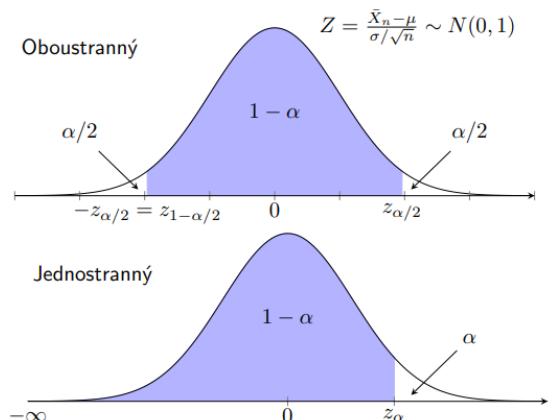
$$\left(\bar{X}_n - z_{\alpha} \frac{\sigma}{\sqrt{n}}, +\infty \right) \text{ a } \left(-\infty, \bar{X}_n + z_{\alpha} \frac{\sigma}{\sqrt{n}} \right)$$

při stejném značení.

- gaussovo rozdělení, standardizovali jsme výběrový průměr
- díky CLV jsme schopni intervaly spolehlivosti použít na libovolné rozdělení
- kritická hodnota standardního normálního rozdělení je v tabulkách
- α určuje konfidenční interval, čím vyšší α , tím užší interval spolehlivosti, protože je méně spolehlivý (např. $\alpha = 0,2 \Rightarrow 80\%$ interval spolehlivosti, může být užší)

○ Neznámý rozptyl

- Rozptyl nahradíme výběrovým rozptylem, směrodatná odchylka je jeho odmocninou
- Studentovo rozdělení
- Opět lze použít na libovolné rozdělení díky CLV
- Přibližná spolehlivost intervalu je pak $1 - \alpha$



- intervaly jsou širší než pro známý rozptyl (protože kalkulujeme s jeho odhadem)

Věta

Uvažujme náhodný výběr X_1, \dots, X_n z normálního rozdělení $N(\mu, \sigma^2)$. Oboustranný $100 \cdot (1 - \alpha)\%$ interval spolehlivosti pro μ je

$$\left(\bar{X}_n - t_{\alpha/2, n-1} \frac{s_n}{\sqrt{n}}, \bar{X}_n + t_{\alpha/2, n-1} \frac{s_n}{\sqrt{n}} \right),$$

kde $t_{\alpha/2, n-1}$ je **kritická hodnota Studentova t-rozdělení** s $n - 1$ stupni volnosti.

Jednostranné $100 \cdot (1 - \alpha)\%$ intervaly spolehlivosti pro μ jsou pak

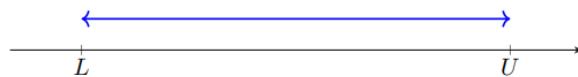
$$\left(\bar{X}_n - t_{\alpha, n-1} \frac{s_n}{\sqrt{n}}, +\infty \right) \quad a \quad \left(-\infty, \bar{X}_n + t_{\alpha, n-1} \frac{s_n}{\sqrt{n}} \right)$$

při stejném značení.

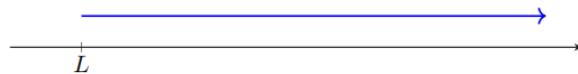
- Testování statistických hypotéz o střední hodnotě

- často potřebujeme ověřit tvrzení o rozdělení dat, ale máme k dispozici jen náhodný výběr (např. vadné produkty při přijetí)
- nulová hypotéza** H_0 : tvrzení, které chceme potvrdit nebo zamítнуть
- alternativní hypotéza** H_A : opačné tvrzení, které stavíme proti H_0
 - testujeme nulovou hypotézu proti alternativní hypotéze
 - na základě zoho H_0 nezamítáme, nebo H_0 zamítáme ve prospěch H_A
- Typy hypotéz:
 - neparametrické**: výběr z obecného rozdělení, tvrzení se týkají různých vlastností rozdělení (mediánu) nebo tvaru celého rozdělení
 - parametrické**: náhodný výběr z rozdělení s parametrem $\theta \in \mathbb{R}^d$, tvrzení se týkají hodnoty θ .
- Postup:
 - zvolíme hladinu významnosti α
 - napozorujeme realizace náhodného výběru
 - sestaváme jednostranný (horní/dolní) nebo oboustranný interval spolehlivosti (L, U) pro θ odpovídající alternativní hypotéze H_A
 - zamítáme hypotézu H_0 ve prospěch H_A na hladině významnosti α , pokud H_0 v tomto intervalu neleží

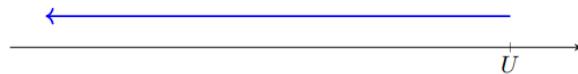
Zamítneme $H_0 : \theta = \theta_0$ ve prospěch oboustranné alternativy $H_A : \theta \neq \theta_0$, pokud testovaná hodnota θ_0 neleží v **oboustranném** konfidenčním intervalu.



Zamítneme $H_0 : \theta = \theta_0$ ve prospěch jednostranné alternativy $H_A : \theta > \theta_0$, pokud testovaná hodnota θ_0 neleží v **horním jednostranném** konfidenčním intervalu.



Zamítneme $H_0 : \theta = \theta_0$ ve prospěch jednostranné alternativy $H_A : \theta < \theta_0$, pokud testovaná hodnota θ_0 neleží v **dolním jednostranném** konfidenčním intervalu.



- Pokud má náhodný výběr jiné než normální rozdělení, můžeme postupovat přibližně (opět díky CLV, testy mají potom **asymptotickou** hladinu významnosti α)

Kombinační a sekvenční logické obvody (Mealy, Moore), popis a možnosti implementace na úrovni hradel. Minimalizace vyjádření logické funkce (s využitím map)

BI-SAP

Kombinační a sekvenční logické obvody

- **Kombinační logický obvod** – takový obvod, kde každý z výstupů out je určen kombinací vstupů in
 - o k vyjádření funkce výstupu používáme Booleovu algebru
 - o hodnota výstupu je určena pouze vstupními hodnotami, nezáleží na stavu
 - o je popsán **kombinační funkcí**:
$$y_1 = f_1(x_1, x_2, x_3, \dots, x_p),$$

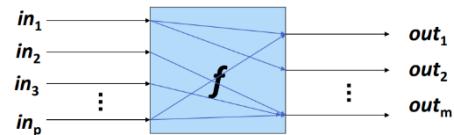
$$y_2 = f_2(x_1, x_2, x_3, \dots, x_p),$$

$$\vdots$$

$$y_m = f_m(x_1, x_2, x_3, \dots, x_p)$$
 - o Realizace všech funkcí f_i najednou



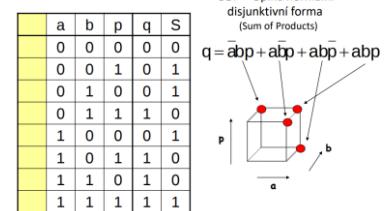
$$out_k = f_i(in_1, in_2, in_3, \dots, in_p), \quad k=1,2,\dots,m, \quad i=1,2,\dots,p$$



- **Reprezentace kombinačních logických funkcí**

- o **Booleovská n-krychle**
 - Krychle = logický součin (konjunkce) literálů nebo samotný literál
- o **Logická funkce**

- Literál = proměnná v přímé nebo negované formě
- Booleovské formule (výrazy) = zřetězení závorek, literálů, Booleovských operátorů a komplementace (negace)
- Každá formule vyjadřuje nějakou Booleovskou funkci $f: B^n \rightarrow B$ n proměnných



- **Mapa** = plošný útvar (čtverec/obdélník) rozdelený do 2^n polí, kde n je počet nezávisle proměnných dané funkce. Podle vzájemného přiřazení políčka mapy a stavového indexu typy:

- o **Svobodova mapa** – stavové indexy přiřazeny do políček, která leží v mapě vedle sebe
- o **Karnaughova mapa** – kombinace hodnot nezávisle proměnných zobrazených do políček vedle sebe liší vždy hodnotou jedné nezávislé proměnné – v políčkách, která leží v mapě vedle sebe jsou vždy sousední stavy
 - Pro blížecky: proměnné jsou buď 1 nebo 0, když je mám 4, tak mám tabulku 4×4 a vyjadřuju je od 0–15 podle jejich binárního zápisu, a chci, aby se v binárním zápisu čísla lišila max. v jednom místě binárního zápisu

- **Pravdivostní tabulka** – sloupce stavový index / proměnné / funkční hodnota / minterm / maxterm

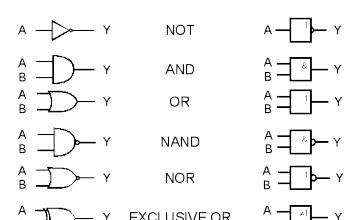
- o Term = výraz tvořený pouze literály a jednou operací
- o P-term (součinový term) = term tvořen pouze literály a operacemi logického součinu
- o S-term (součetový term) = term tvořen pouze literály a operacemi logického součtu
- o **Minterm** = takový P-term, který obsahuje všechny nezávisle proměnné
- o **Maxterm** = takový S-term, který obsahuje všechny nezávisle proměnné
- o **Stavový index** = desítkový zápis kombinace hodnot nezávisle proměnných

- Výčet stavových indexů

- Grafem přechodových funkcí

- **Schéma** – realizace logických funkcí pomocí hradel

- o Hradla graficky představují operace AND, OR, NOT, NAND, NOR, XOR, XNOR



- **Sekvenční obvod** – obvod, ve kterém hodnoty výstupních proměnných závisí nejen na kombinaci hodnot vstupních proměnných, ale i na posloupnosti těchto hodnot

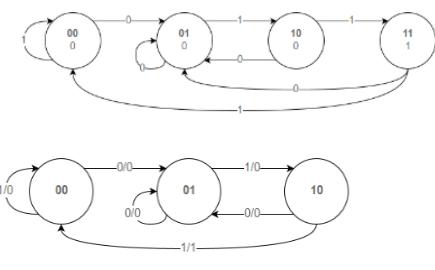
- o Zpětná vazba – realizace „pamatování“ historie
- o Matematický model SO = **konečný automat**

X - Množina vstupů
Y - Množina výstupů
Q - Množina stavů

$FSM = (X, Y, Q, \delta, \lambda)$
 δ – Přechodová funkce
 $\delta : X \times Q \rightarrow Q$

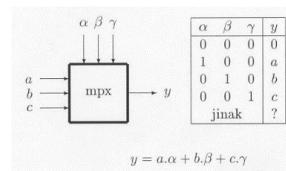
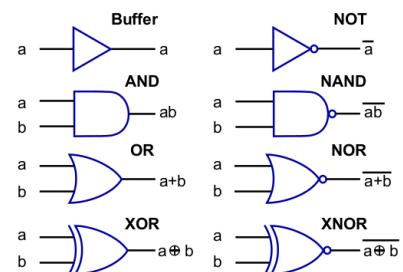
λ – Výstupní funkce
Moore: $\lambda : Q \rightarrow Y$
Mealy: $\lambda : X \times Q \rightarrow Y$

- o Asynchronní (bez hodinového vstupu) a synchronní (s hodinovým vstupem)
- o Na rozdíl od kombinačních obvodů sekvenční mají Hodinový pulz (Clock), jenž s každým tikem (vznik náběžné hrany – z 0 do 1) přechází do dalšího stavu v automatu na základě vstupu
- o **Typy sekvenčních obvodů** (automatů):
 - **Moore** – reaguje na vstup až při přechodu do dalšího stavu
 - závisí pouze na stavu, kde se nachází – *state-based*
 - výstup je v uzlech
 - **Mealy** – reaguje na vstup ihned – *input-based*
 - výstup záleží na aktuálním stavu
 - výstup je v přechodech (vpravo)
 - vstup je v přechodech vlevo



Popis a možnosti implementace na úrovni hradel

- Implementace na úrovni hradel lze pro kombinační i sekvenční obvody reprezentovat s použitím hradel
- Pro optimalizaci kombinačních obvodů se používají **minimální normální disjunktní formy**
- Pro sekvenční obvody se využívají **Moore** a **Mealy** s tabulkami přechodů
- Z hradel se vytváří diagramy
- XOR hradlo je velmi používané, je úspornější oproti ostatním
- **Dekodér 1 z N** – převodník – přijde zakódovaný vstup (např. číslo v binárním zápisu) a výstup 1 je pouze pro jednu žárovku, protože odpovídá jen ona (tzn. přijde 0101 a rozsvítí se 5. žárovka z 16)
- **Multiplexor** – mám několik vstupů a vybírám jen jeden, aby šel na výstup
 - o realizace např. pomocí hradel NAND
 - o nejjednodušší multiplexor – přivedení jednoho ze dvou vstupních signálů na výstup podle signálu V
- **Demultiplexor** – opak multiplexoru, přesměruje 1 vstup na několik výstupů
- **Poloviční sčítáčka** – sčítá 2 vstupy (bity) podle pravdivostní tabulky
 - o součet pro úplnou sčítáčku zřetězením dvou polovičních sčítáček a přenos pomocí hradla OR
- **Úplná sčítáčka** – sčítá 2 bity + přechod (přenos z předchozího řádu), dá se nakombinovat pro sčítání binárních čísel



Minimalizace vyjádření logické funkce (s využitím map)

- Hledání minimálního (optimálního) pokrytí jedničkových stavů logické funkce založeno na hledání maximálních skupin sousedních stavů
- **Karnaughovy mapy** – sousední stavy topologicky vedle sebe – sousední jsou i oba krajní sloupce a oba krajní řádky
- **MNDF** – minimální normální disjunktní forma – logický součet minimálního počtu minimálních P-termů
- **MKNF** – minimální konjunktivní normální forma – logický součin minimálního počtu minimálních S-termů

- Postup pro vytvoření MNDF

1. napíšu pravdivostní tabulku podle toho, co chci za vstupy
2. do Karnaughovy mapy zapíšu 1 tam, kde chci true, a x (don't care)
3. najdu co největší skupiny (velikosti skupin = mocniny dvojkdy) a zakroužkuju je
4. skupinu přepíšu do funkce

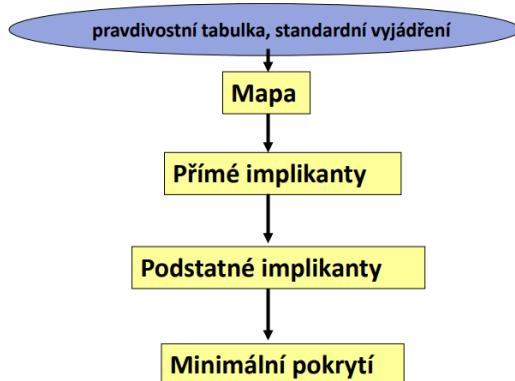
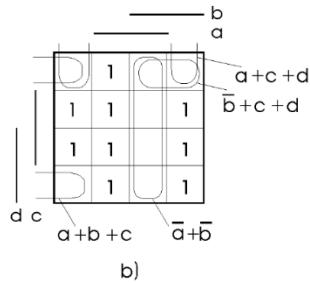
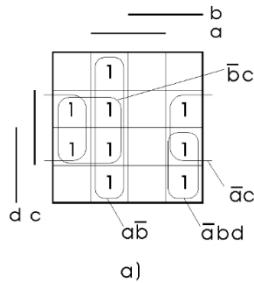
	b	a
d	1	
c		
0	1	2
4	5	6
8	9	10
12	13	14
15		

Svobodova mapa

	b	a
d	1	
c		
0	1	3
4	5	7
8	9	11
12	13	15
14		
15		

Karnaughova mapa

- Neurčený stav – don't care – stav logické funkce, pro který daná funkce není definovaná – lze zároveň zahrnout do skupin jedniček i nul – pokrýváme, pokud je to výhodné

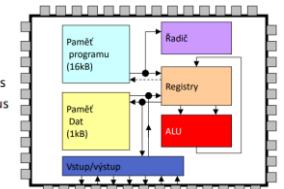
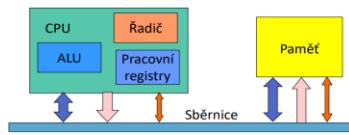


Architektura číslicového počítače, instrukční cyklus počítače, základní třídy souborů instrukcí (ISA). Paměťový substitut počítače, paměťová hierarchie, cache

BI-SAP

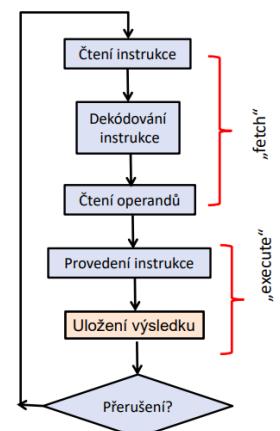
- Architektura číslicového počítače

- **Hlavní paměť** (main memory) – často mimo procesor
- **Procesor**
 - **Datová část** – pracovní a adresové registry – dočas. uchování dat, instrukcí a pomocných proměnných
 - **Aritmeticko-logická jednotka – ALU**
 - **Řadič** (controller) – řídí všechny jednotky, vysílá řídící signály po CB
- Vstupní a výstupní zařízení
- **Von Neumannova architektura** – paměť společná pro instrukce a data
- **Harvardská architektura** – oddělená paměť instrukcí a paměť dat, je využívanější



- Instrukční cyklus počítače

- **Instrukce** – příkaz zakódovaný jako číslo
 - Vyjadřuje, co se má provést, jak se to má provést (jaká operace), s čím se to má provést (operandy), kam se má uložit výsledek a kde se má pokračovat
 - Operační znak + operand
 - Nejnižší úroveň, se kterou může programátor pracovat
 - Aritmetické (ADD, SUB), řídící (JMP), uložení do paměti, ...
- Činnost procesoru (řadiče) – sekvenční zpracování instrukcí v nekonečné smyčce podle instrukčního cyklu
- **Dělení instrukcí** (podle počtu adres instrukcí):
 - 1 adresová (ADD R1)
 - 2 adresová (ADD R1, R2)
 - 3 adresová (ADD R1, R2, R3)



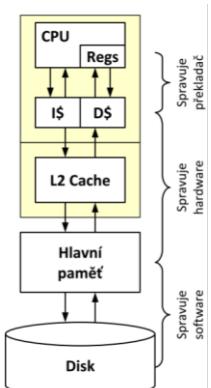
- Základní třídy souborů instrukcí (ISA)

- **ISA = Instruction Set Architecture**
 - Zahrnuje:
 - Typy a formáty instrukcí, instrukční soubor
 - Datové typy, kódování a reprezentace, způsob uložení dat v paměti
 - Módy adresování paměti, přístup do paměti, mimořádné stavy
 - Umožnuje definici rozhraní mezi HW a SW
 - Standardizuje instrukce, bitové vzory strojového jazyka
 - Abstrakce (výhoda – různé implementace stejné architektury)
 - Obecný popis organizačních, funkčních a provozních principů procesoru
- Jazyk symbolických instrukcí (JSI/JSA) – mezi strojovým kódem a vyššími programovacími jazyky
- **Adresace operandů** – operandy určeny adresou do paměti – v instrukci není uvedena hodnota operandu
 - Přímá – práce přímo s registrem – operand je implicitní, přímá konstanta nebo přímá adresa
 - Nepřímá – v registru/paměti je adresa, má data, se kterými se pracuje
 - Relativní – offset od určité adresy (v registru nebo přímo)
 - Indexová – báze + offset

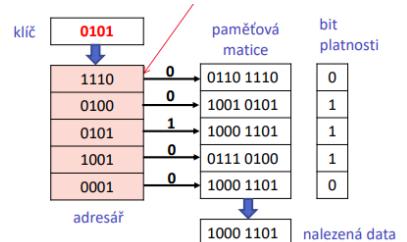
- autoinkrementace + autodekrementace
- **Střadačově (akumulátorově) orientovaná ISA**
 - Jeden pracovní registr = **střadač** – implicitní operand ALU (druhý operand v paměti)
 - Z kalkulaček (nejstarší ISA)
 - Jednoduchý HW, rychlé přepínání kontextu
 - Omezený paralelismus, častá komunikace s pamětí
- **Zásobníkově orientovaná ISA**
 - Pracovní registry uspořádány do **hardware zásobníku** – ALU pracuje pouze s operandy uloženými na jeho vrcholu
 - Krátké instrukce (operandy určeny implicitně), krátké prog., jednoduché dekódování instrukcí
 - Nelze náhodně přistupovat k lokálním datům – zásobník je sekvenční (omezený paralelismus)

0 operandů ADD stack(top-1) \leftarrow stack(top) + stack(top-1)
top--
- **Registrová ISA (GPR – general purpose registers)**
 - Registry rychlejší než paměť (včetně cache)
 - Lze přistupovat náhodně k datům, mohou obsahovat mezivýsledky a lokální proměnné – méně častý přístup do paměti
 - Omezený počet těchto registrů, složitější překladač, registry bez složitějších datových struktur
 - Dnes převládá

2 operandy ADD A B EA(A) \leftarrow EA(A) + EA(B)
3 operandy ADD A B C EA(A) \leftarrow EA(B) + EA(C)
- Registry s konkrétní funkcí
 - **Programový čítač** (PC – program counter) – obsahuje adresu instrukce
 - **Ukazatel zásobníku** (SP – stack pointer) – nutný při volání podprogramů a zpracování přerušení
 - Ukládání a výběr dat – instrukce POP a PUSH
 - **Stavový registr** (SREG/FLAG – flag register) – uchov. příznaků – podm. pro větvení programů
 - Registr instrukce (RI) – uložení právě přečtené instrukce
- **Paměťový subsystém**
 - **Paměť** = zařízení pro uchování dat a programů
 - Zařízení obsahují různé druhy pamětí, liší se rychlosťí, přístupem, uchováváním informací atd.
 - Nejčastěji jsou využívány data v M1 (primárním cache)
 - Kritéria pro dělení typů paměti:
 - **Použití v počítači** – hlavní paměť, skrytá paměť, vnější paměť, ...
 - **Fyzikální princip** – polovodičová, magnetická, optická, ...
 - **Způsob výběru položek** – adresový výběr, postupný výběr, asociativní, zásobník, fronta, ...
 - **Způsob a možnost změny** uložené informace – RWM, RAM, ROM, PROM, EPROM, EEPROM, FLASH, ...
 - **Vztah k napájení**
 - Volatilní – uložená informace zanikne po vypnutí napájení – RAM
 - Non-volatile – obsah zůstane uchován i po vypnutí napájení – HDD
 - **Paměťová buňka** – základní stavební blok paměti – záznam jednoho bitu
 - **Paměťové místo** – skupina paměťových buněk, do kterých lze najednou zapisovat nebo je číst
 - **Položka** – obsah paměťového místa
 - **Adresa** – číselné označení (index) paměťového místa, jímž lze vybírat jednotlivé položky
 - **Kapacita paměti** – maximální počet položek, které lze do paměti uložit
 - **Paměťová matice** – skupina paměťových míst uspořádaná tak, že je lze vybírat adresou

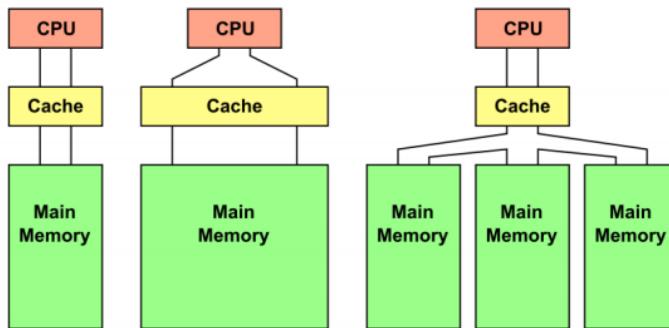


- **Paměťová hierarchie**
 - Registr: rychlý, drahý, nejblíž ALU, protože ho přímo využívá (klopné obvody)
 - Cache (skrytá paměť): rychlé, dražší, kapacitně menší (~kB), blíž k procesoru
 - primární: oddělená, instrukční a datová (I\$, D\$)
 - sekundární (L2): nejlépe na čipu, určitě v pouzdře (SRAM)
 - Hlavní paměť: pomalejší, levnější, větší, magnetický disk „hard“ disk (DRAM, ~ GB)
 - Vnější paměť: ještě pomalejší, ale velká kapacita, soubory swap, magnetické disky, flash paměti, CD
 - **Cache – skrytá paměť**
 - Využití asociativního přístupu k položkám (CAM) – přístup pomocí obsahu, ne adresy
 - většinou více vrstev (L1, L2...)
 - "malá" rychlá paměť zařazená mezi procesor a hlavní paměť
 - obsahuje kopie nejčastěji používaných položek hlavní paměti – princip lokality
 - časová lokalita – když procesor používá nějakou položku v paměti, je vysoká pravděpodobnost, že ji bude používat znova
 - prostorová lokalita – procesor bude používat položky v paměti umístěné poblíž právě používané
 - možné realizovat RAM paměť
 - **Plně asociativní paměť**
 - adresuje se částí datové položky, která se má vyhledat, tzv. klíčem
 - na rozdíl od adresovatelné paměti (např. SRAM) neobsahuje adresový dekodér, ale adresář
 - Nevýhoda – velikost
 - **Čtení z cache**
 - cache hit: data jsou nalezena
 - hit rate: poměr cache hit a počet všech dotazů
 - hit time: doba nalezení údajů v cache a předání procesoru
 - cache miss: výpadek cache (je třeba načtení z nižší úrovně)
 - miss rate: četnost výpadků cache (Miss rate = 1 – Hit rate)
 - miss penalty: doba potřebná k získání dat z pomalejší paměti
 - **Zápis do cache**
 - pokud položka v cache není, zapíše se jen do hlavní paměti
 - průběžný zápis: nová hodnota se zapíše se zároveň do cache i HP
 - odložený zápis: zapíše se do cache a při uvolnění z cache se musí zapsat do HP
 - **Asociativní paměť s omezeným stupněm asociativity**
 - Každé položce je určeno místo (nebo několik míst podle stupně associativity), kde se může nacházet – místo je určené částí adresy položky
 - Klíč rozdělen na dvě části
 - Část adresy (adr), podle které se určí místo, kde se hledaná položka může vyskytovat
 - Podklíč (tagr),
 - Adresář možné realizovat běžnou pamětí RAM
 - Přítomnost položky se zjistí porovnáním s podklíči uloženými v adresáři
 - Stupeň asociativity – počet míst, na kterých se položka může nacházet
 - Stupeň asociativity 1 – přímo mapovaná cache (direct mapped) – je možné data se stejnou částí adresy uložit jen na jedno místo – řádek v paměťové matici
 - Zbylá část adresy – klíče (tags), uložená do adresáře, se může měnit
 - Celá adresa rozdělena na 3 části – podklíč (tags), adresu řádku (adr) a adresu slabiky v bloku



- Vezmu kus adresy – kouknu na to místo – přečtu – srovnám – přiřadím bit platnosti – řekne mi, jestli je nalezeno – vypadnou buď data, nebo musím do hlavní paměti – přemazání tím, co v ní je
- Organizace paměťového subsystému

Stejná šířka sběrnice mezi CPU a Cache, Cache a Hlavní paměti
Sběrnice mezi CPU a Cache je rozšířena o multiplexor (zúžení k CPU). Cache a Hlavní paměti propojena N širokou sběrnicí
Paměť organizována do samostatných banků. Datová sběrnice má stejnou šířku na všech úrovních.



- 1. Jednoduchá organizace hlavní paměti**
 - Šířka datového slova je zachována
 - Jednoduchost modelu, snadná implementace, velká latence
- 2. Paměť s širokým datovým slovem**
 - Datová sběrnice mezi HP a cache je rozšířena na N násobek velikosti datového slova procesoru.
 - Dodatečný HW k realizaci výběru konkrétního slova z N slov dlouhého řádku v cache.
 - Významné snížení latence paměti při sekvenčním čtení v rámci N slov
- 3. Prokládaná hlavní paměť**
 - Hlavní paměť je rozdělena na nezávislé moduly. Na každý z modulů jsou požadavky zasílány samostatně. Dochází ke zřetězení přístupů do HP.
 - Snížení latence, odpadá potřeba široké sběrnice, komplexnější řadič paměti

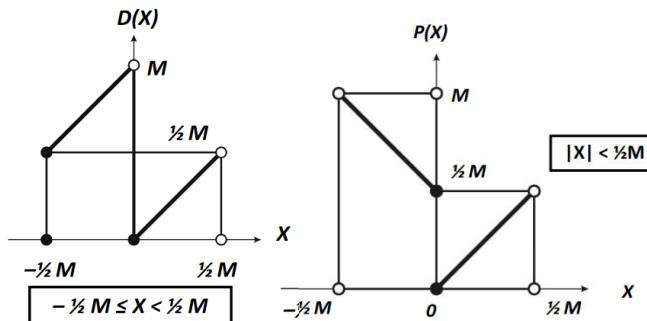
Kódy pro zobrazení čísel se znaménkem a realizace aritmetických operací (paralelní sčítáčka/odčítáčka, realizace aritmetických posuvů, dekodér, multiplexor, čítač).

Reprezentace čísel v pohyblivé řádové čárce.

BI-SAP

Kódy pro zobrazení čísel se znaménkem

- Počítání ve dvojkové soustavě
 - o Převod z dvojkové do desítkové soustavy – podle mocnin 2
 - o Převod z desítkové soustavy do dvojkové – rozdělení čísla před řádovou čárkou
 - Část před ř. č. postupně dělíme dvěma a zapisujeme od nultého řádu
 - Část za ř. č. násobíme dvěma a hodnotu, která vyjde před řádovou čárkou (0 nebo 1), zapisujeme od řádové čárky vpravo
 - o **Řádová mřížka** – určuje formát zobrazitelných čísel v počítači (definuje nejvyšší řadu n a nejnižší řadu -n)
 - Délka řádové mřížky (l) – počet řad obsažených v řádové mřížce
 - Jednotka řádové mřížky (e) – nejmenší číslo zobrazitelné v řádové mřížce
 - **Modul řádové mřížky (M)** – nejmenší číslo, které již v řádové mřížce zobrazitelné není
 - o **Přenos (carry – C)** – bit, který vypadává z řádové mřížky
 - x Přeplnění (overflow) – signalizace nesprávného výsledku
- **Přímý kód** – znaménko a absolutní hodnota – sign-magnitude
 - o Znaménko reprezentováno číslicí 0 pro plus a 1 pro mínus
 - o 0 má dva obrazy – kladná a záporná 0 (000 a 100)
 - o Při realizaci operací musíme pracovat zvlášť se znaménkem a zvlášť s absolutní hodnotou (nezáporným číslem)
 - o Znázornění zobrazení:



X	P(X)
+0	0 0 0
+1	0 0 1
+2	0 1 0
+3	0 1 1
-0	1 0 0
-1	1 0 1
-2	1 1 0
-3	1 1 1

- o Příklad: $A - B: M = 1000, B = 101, \bar{B} = 010$
 $B + \bar{B} = 111 = 1000 - 1 = M - 1$
 $-B = \bar{B} + 1 - M$

Hledaný rozdíl:

$$A - B = A + \bar{B} + 1 - M$$

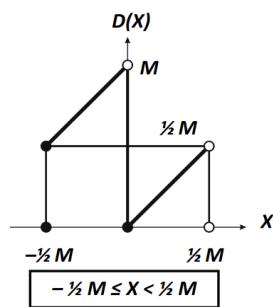
→ abychom dostali správný výsledek, musíme mít možnost odečíst modul → musí vyjít přenos

- **Doplňkový kód** – pro dvojkovou sestavu – 2's complement

$$\mathcal{D}(X) = \begin{cases} X, & \text{je-li } X \geq 0 \\ M + X, & \text{je-li } X < 0 \end{cases}$$

- o Pracujeme s celým obrazem čísla včetně znaménka
- o Jeden obraz nuly
- o Minus se pořád značí jako 1 na začátku
- o Přičtení modulu M nemá žádný vliv – přenos se ignoruje
- o Určení **obrazu záporného čísla**

X	D(X)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
-4	1 0 0
-3	1 0 1
-2	1 1 0
-1	1 1 1



- Zapíšeme číslo ve dvojkové soustavě do řádové mřížky
- Invertujeme všechny bity
- Přičteme jedničku

nejvyšší bit představuje znaménko

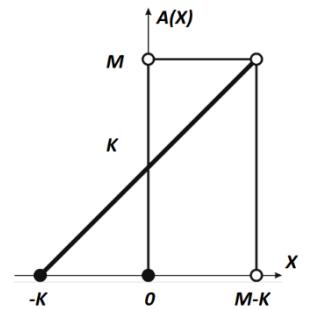
$$\mathcal{D}(5) = 5_{10} = 101_2 \quad +101_2 \xrightarrow{\mathcal{D}} \begin{array}{|c|c|c|c|}\hline 0 & 1 & 0 & 1 \\ \hline \end{array}$$

$$\mathcal{D}(-5) = 16_{10} + (-5_{10}) = 11_{10} = 1011_2 \quad -101_2 \xrightarrow{\mathcal{D}} \begin{array}{|c|c|c|c|}\hline 1 & 0 & 1 & 1 \\ \hline \end{array}$$

- Obraz záporného čísla X je doplňkem jeho hodnoty do modulu M řádové mřížky

- Aditivní kód – s posunutou nulou – biased

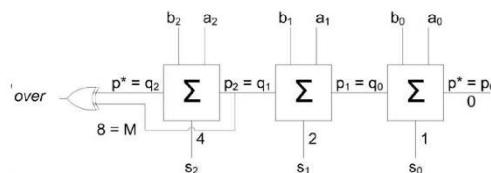
- Posunutí intervalu zobrazovaných čísel – ke všem číslům se přičítá vhodná konstanta $A(X) = X + K$ pro $-K \leq X < M - K$
- Nula se nezobrazuje jako nula
- Konstanta K – polovina modulu
- Obraz čísla X: $A(X) = X + K$



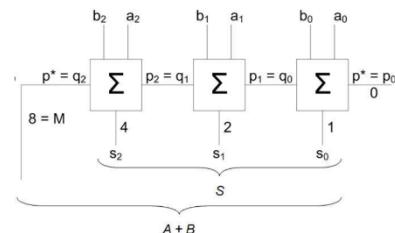
Realizace aritmetických operací

- Paralelní sčítáčka/odčítáčka

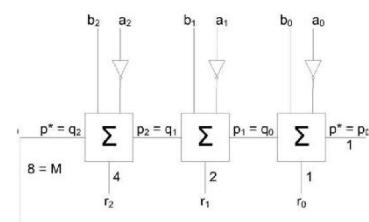
- Tříbitová sčítáčka – základní blok



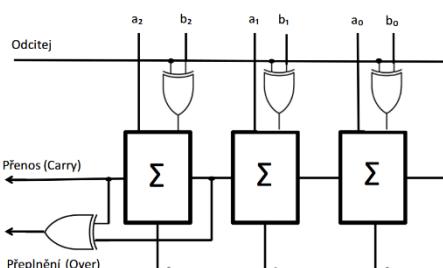
- Paralelní sčítáčka



Paralelní odčítáčka



- Paralelní sčítáčka/odčítáčka

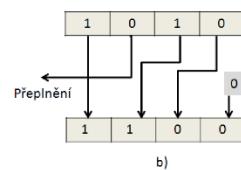
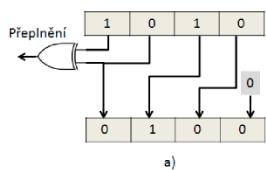


- Sčítání a odčítání je v podstatě stejná operace, jen někdy musíme vstupy B negovat, přivést horkou jedničku a jinak detekovat, zda je výsledek správný
- U odčítání místo přenosu jeho negace – **výpůjčka**
- **Horká jednička** – přičtená jednotka řádové mřížky při počítání obrazu opačného čísla v doplňkovém kódru

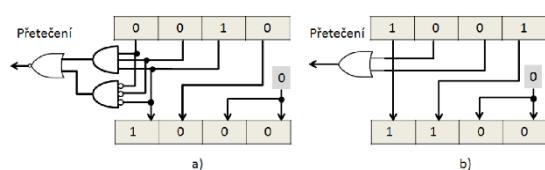
- Realizace aritmetických posuvů

- **Aritmetický posuv vlevo**: výsledek musí odpovídat násobení mocninou 2

- Posun o jedno místo vlevo

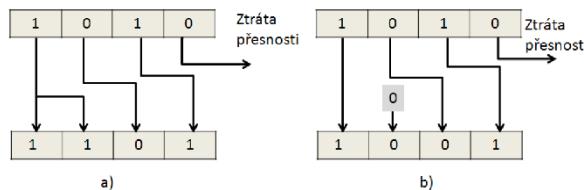


- Posun o dvě místa vlevo – násobení 4



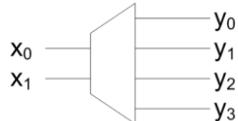
- **Aritmetický posuv vpravo:** výsledek musí odpovídat dělení mocninou 2

- Posun o jedno místo vpravo



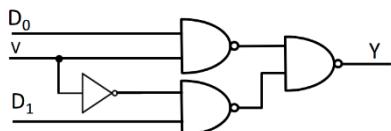
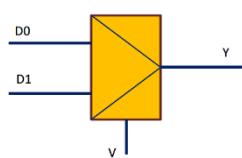
○ a = doplňkový kód, b = přímý kód

- Dekodér

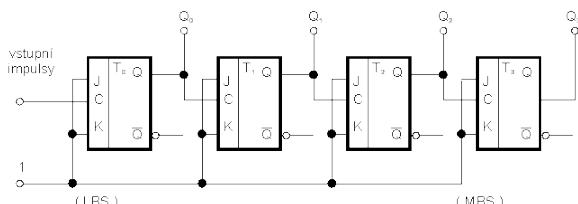


- Multiplexor

$$Y = \bar{V} \cdot D1 + V \cdot D0$$



- Čítač



Reprezentace čísel v pohyblivé řádové čárce

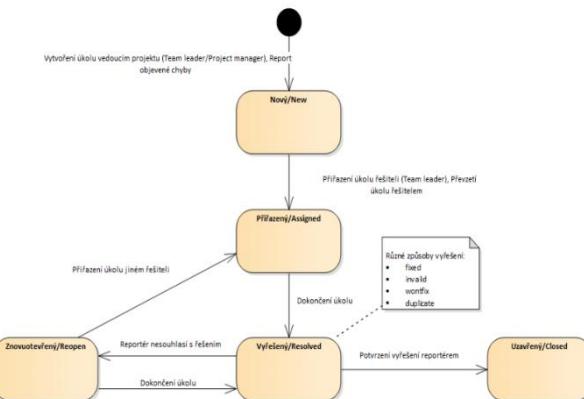
- Rozsah zobrazitelných čísel je vždy hardwarově omezen, ale je nutné pracovat s velmi velkými i velmi malými číslami → pohyblivá řádová čárka – **floating point**
- Uspořádaná dvojice čísel:
 - M – **mantisa** – informace o „hodnotě“ čísla
 - e – **exponent** – informace o pozici řádové čárky
 - zobrazení čísla A ve tvaru: $A = M \cdot z^e$
 - z = základ soustavy (obvykle z = 2)
 - e obvykle celé číslo
- **Normalizovaný tvar** – zápis čísla, kdy už nelze mantisu posunout více doleva – po každé operaci je nutno znova provést normalizaci
 - Normalizovaný tvar mantisy bude mít vždy v nejvyšším řádu jedničku
 - Jednička lze „skrýt“ – vynechat ze zápisu → skrytá jednička
- **Aritmetické operace**
 - Sčítání/odčítání – srovnáním exponentů a sečtením/odečtením mantis
 - Násobení – sečtením exponentů a vynásobením mantis
 - Dělení – odečtením exponentů a vyděleném mantis
 - Porovnání – srovnáním exponentů a porovnáním mantis
 - Posuv – posunem mantisy nebo zvětšením/zmenšením exponentu

Nástroje pro podporu tvorby softwarových produktů: Sledování chyb a správa úkolů (používané nástroje, typický životní cyklus úkolu/chyby), správa a sdílení zdrojových kódů (principy řešení spolupráce, hlavní přínosy, používané nástroje)

BI-SI1.2

- Sledování chyb a správa úkolů

- Ve větších týmech během tvorby projektu je nutné mít nástroj pro management jednotlivých úkolů (např. od projektového manažera, zákazníka).
- Pomáhá vyřešit organizaci v týmu:
 - Evidence úkolů: co je nutné udělat?
 - Přidělování úkolů: kdo to bude dělat?
 - Plánování: do kdy je nutné úkol udělat?
 - Kontrola plnění úkolů
 - Vyhodnocování odvedené práce
- existující nástroje: Trac Tickets, JIRA, GitLab, YouTrack, Mantis, Bugzilla, Redmine
- Milníky (milestones)
- **Severita** – dopad na uživatele
- **Priorita** – blocker, critical, major, minor, trivial
- **Životní cyklus** – new, assigned, resolved, closed, reopened



- **Způsob vyřešení** – fixed, invalid, wontfix, duplicate, worksforme
- **Typy** – defect/bug, enhancement, task)
- Úkol má životní cykly a různou úroveň priority
- Z hlediska úkolů projekt začíná *User Stories* (abstraktní popisy od zákazníka).
- Dělí se na *Epics* (velké úkoly), ty se rozdělují na *Tasks* (atomické úkoly), příp. na *Subtasks*

- Sdílení a správa souborů

- Důležité uchovávání historie projektu, to poskytují verzovací nástroje (VCS)
- **Větev** – oddělení vývoje (např. pro opravu chyby)
- **Tag** – označení jedné revize/verze
- Řeší více dílčích problémů:
 - **Verzování** – udržuje kompletní historii každého souboru pod správou verzí
 - Lze se k jednotlivým verzím v minulosti kdykoliv vrátit
 - **Zálohování** – v případě poškození/ztráty souborů je možné je obnovit

- Nástroje pro správu verzí (SCM)

- **Centralizované:**
 - Např. SVN, CVS
 - Veškeré revize/verze souborů jsou uloženy pouze v centrálním repozitáři
 - Na lokálním počítači je pouze pracovní kopie (aktuální revize/verze) souborů

- **Distribuované:**
 - Např. GIT, Mercurial
 - Na lokálním počítači jsou uloženy všechny revize/verze
 - Díky tomu může být velká část operací prováděna lokálně
- **Principy řešení spolupráce**
 - **Zamykání** (lock-modify-unlock):
 - Pouze centralizované systémy
 - Může způsobovat organizační problémy (zbytečné blokování práce)
 - Vynucuje serializovaný přístup i při nekonfliktních úpravách
 - Využití pro soubory, které nelze po částech slučovat (grafické podklady, modely, apod) = binární soubory
 - **Kopie** (copy-modify-merge)
 - Centralizované i distribuované systémy
 - Častější, podporuje to většina nástrojů
 - Odstraňuje problémy zamykání
 - V případě konfliktních změn je nutné provést ručně sloučení
 - Git add, status, commit, clone, checkout, push, pull...
- Ignorování souborů – umožňuje identifikovat soubory, které nemají být verzovány (.gitignore)
 - Záložní soubory, dočasné soubory, lokální nastavení, ...

Analytický doménový model tříd a popis životního cyklu identifikovaných tříd (cíle, UML diagram tříd, UML stavový diagram)

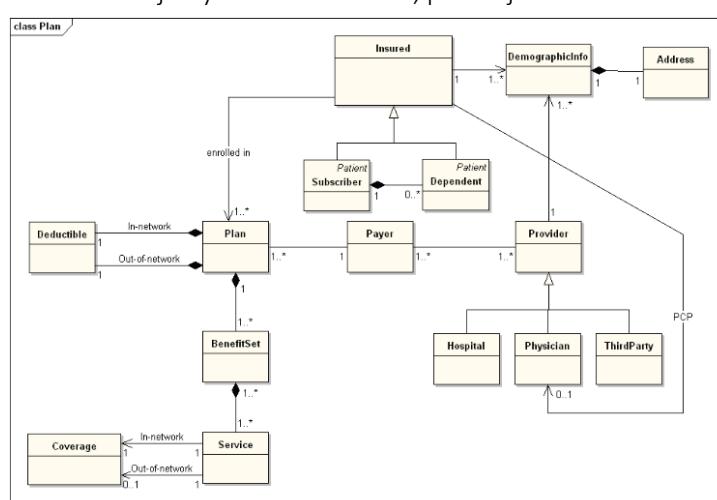
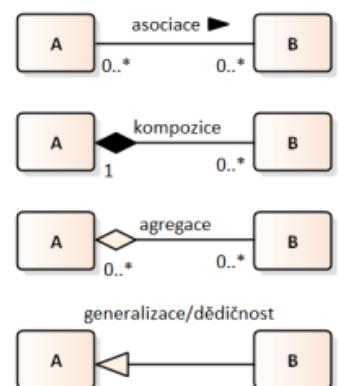
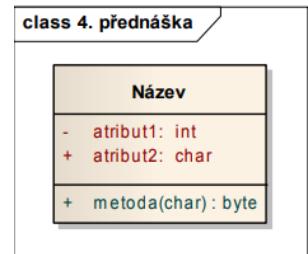
BI-SI1.2

- Analytický doménový model tříd

- Model pomáhající definovat entity v problémové doméně, jejich vlastností a vztahů mezi nimi
- Modeluje se diagram tříd, ale vynechávají se implementační detaily
- Třídy reprezentují entity z problémové domény, nikoliv softwarové třídy
- V praxi slouží nejdřív pro analýzu projektu, pochopení cílů
- Později z toho vychází návrh pro architekturu
- *Cíle:*
 - Popis dat
 - Popis významu termínů
 - Popis vazeb mezi entitami
 - Identifikace stavu entit
 - Základ pro design (DB model, model tříd)
 - Zachycení atributů

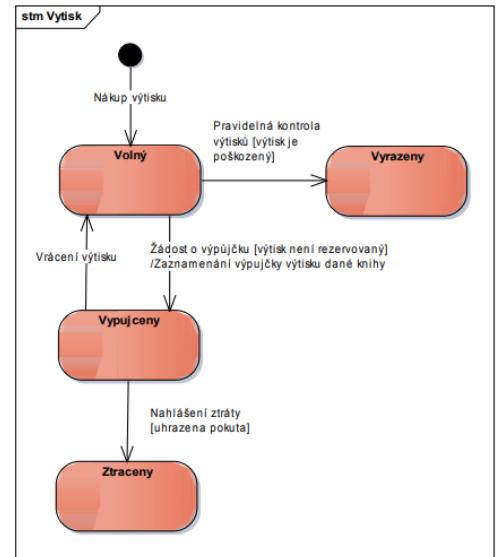
- UML diagram tříd

- Patří do skupiny diagramů struktur
- Ro zšiřuje vizualizaci doménového modelu
- Upřesňuje data a slouží k vývoji aplikací
- Doplňuje se o potřebné entity
- Skládá se z jednotlivých tříd a vazeb mezi nimi, které mohou být různých typů
- *Využití:* business/analytický doménový model, databázový model, návrhový model tříd
- Složení třídy: atributy, metody, viditelnost
- Typy vztahů:
 - **Asociace** – vztah mezi objekty A a B (objekt někomu patří)
 - **Kompozice** – A je složena z B, B bez A není potřeba (např. člověk a orgány)
 - **Agregace** – A obsahuje seznam B, ale nemusí (členové týmu)
 - **Dědičnost** – B dědí od A
- Násobnosti a popis vztahů
 - Atribut vs vazba: význam shodný, vazba je názornější
 - Asociační třída (asociouje dvě třídy mezi sebou)
 - Hledání tříd – objekty z reálného světa, podst. jména



- UML stavový diagram

- Patří do skupiny diagramu chování
- Není diagram aktivit
- Konečný stavový automat
- Cíle:
 - Porozumět životnímu cyklu entit,
 - Zachycení událostí vyvolávajících přechod a podmínek, za kterých může změna nastat
 - Vyjasnění stavů, ve kterých se může entita nacházet
- Stav a přechody (popis mezi nimi pro jednu entitu)
- *Událost[Podmínka] / Akce*



Metody řešení rekurentních rovnic, sestavování a řešení rekurentních rovnic při analýze časové složitosti algoritmů

BI-ZDM

- Lineární rekurentní rovnice řádu $k \in \mathbb{N}$ = libovolná rovnice ve tvaru

$$a_{n+k} + c_{k-1}(n) \cdot a_{n+k-1} + \cdots + c_1(n) \cdot a_{n+1} + c_0(n) \cdot a_n = b_n$$

pro všechna $n \geq n_0$, kde $n_0 \in \mathbb{Z}$

- $c_i(n)$ pro $i = 0, \dots, k-1$ = **koeficienty rovnice** – nějaké funkce $\mathbb{Z} \mapsto \mathbb{R}$, přičemž $c_0(n)$ není identicky nulová funkce
- $(b_n)_{n=n_0}^{\infty}$ = **pravá strana rovnice** – pevně zvolená posloupnost reálných čísel
- Jestliže $b_n = 0$ pro všechna $n \geq n_0$, pak se příslušná rovnice nazývá **homogenní**
- $c(n)$ jsou funkce, mohou to být ale jen konstanty, pak je to LRR s konstantními koeficienty (pak $c \in \mathbb{R}$ a $c_0(n) \neq 0$)
- **Řešení** = libovolná posloupnost taková, že po dosazení odpovídajících členů dostaváme pravdivý výrok pro všechna $n \geq n_0$
- **Počáteční podmínky** = libovolná soustava rovnic $a_{n_0} = A_0, a_{n_0+1} = A_1, \dots, a_{n_0+k-1} = A_{k-1}$, kde $A_i \in \mathbb{R}$ jsou pevně zvolená čísla
- Každá LRR má nějaké řešení. Je-li dána LRR řádku k a příslušné počáteční podmínky, pak existuje jediné řešení dané rovnice, které splňuje dané počáteční podmínky.

- Asymptotická složitost – chování funkce pro $n \rightarrow \infty$, zanedbáváme vše, kromě nejvyššího řádu funkce

$f(n) = O(g(n))$	$\exists c \in \mathbb{R}^+$	$\exists n_0 \in \mathbb{N} \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
$f(n) = o(g(n))$	$\forall c \in \mathbb{R}^+$	$\exists n_0 \in \mathbb{N} \forall n \geq n_0 : f(n) < c \cdot g(n)$
$f(n) = \Omega(g(n))$	$\exists c \in \mathbb{R}^+$	$\exists n_0 \in \mathbb{N} \forall n \geq n_0 : c \cdot g(n) \leq f(n)$
$f(n) = \omega(g(n))$	$\forall c \in \mathbb{R}^+$	$\exists n_0 \in \mathbb{N} \forall n \geq n_0 : c \cdot g(n) < f(n)$
$f(n) = \Theta(g(n))$	$\exists c_1, c_2 \in \mathbb{R}^+$	$\exists n_0 \in \mathbb{N} \forall n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Metody řešení rekurentních rovnic a časová složitost

- Problém analýzy složitosti se přesouvá na hledání asymptotického řešení RR. U rovnic se omezíme na tvat $t(n) = a \cdot t\left(\frac{n}{b}\right) + f(n)$

- Cílem je nalézt asymptoticky těsnou funkci pro $t(n)$

- Iterační metoda – postupný rozklad rekurence, telescoping

- K odhadu asymptotické horní a dolní meze
- Předpokládáme lineární rekurenci 1. Řádu se zadanou poč. podmínkou $a_n = K a_{n-1} + f(n)$
- **Dosazování** rekurentního předpisu **do sebe sama** „shora dolů“, tzv. iterováním
- Po konečně mnoha krocích se na konci **součtu** zbavíme rekurentních členů a_k
- Pro vizualizaci iterační metody může být užitečný strom rekurzivních volání, kde se určuje jeho hloubka, šířka v hloubce i a složitost podúlohy v hloubce i
- Příklad: $x_n = x_{n-1} + 1, n > 1, x_1 = 3$:

$$\begin{aligned} x_n &= 1 + x_{n-1} \\ &= 1 + (1 + x_{n-2}) = 2 + x_{n-2} \\ &= 2 + (1 + x_{n-3}) = 3 + x_{n-3} \\ &= \dots \\ &= (n-1) + \underbrace{x_1}_{=3} \\ &= n + 2 \end{aligned}$$

nakonec důkaz indukcí:

Základní krok: $n = 1$. Platí, neboť $x_1 = 3$ a současně $n + 2 = 1 + 2 = 3$

Indukční krok: $n \rightarrow n + 1$. Jestliže pro nějaké $n \geq 1$ platí $x_n = n + 2$, pak také platí $x_{n+1} = n + 3$. Jelikož $x_{n+1} = x_n + 1 = n + 2 + 1 = n + 3$, tvrzení platí

- **Mistrovská metoda** – Master theorem – Nechť $a \geq 1, b > 1$ jsou reálné konstanty, $f(n)$ kladná funkce jedné proměnné. Uvažujme rekurentní rovnici

$$t(n) = a \cdot t\left(\frac{n}{b}\right) + f(n)$$

kde $\frac{n}{b}$ může znamenat buď $\left\lfloor \frac{n}{b} \right\rfloor$ nebo $\left\lceil \frac{n}{b} \right\rceil$ a tyto rozdíly zanedbáváme. Pak $t(n)$ má následující asymptotické řešení:

- MM1 – pokud $f(n) = O(n^{\log_b a - \varepsilon})$ pro nějakou konstantu $\varepsilon > 0$, pak $t(n) = \Theta(n^{\log_b a})$
 - $f(n)$ je polynomiálně menší, než $n^{\log_b a}$, která se nakonec prosadí
- MM2 – pokud $f(n) = \Theta(n^{\log_b a})$, pak $t(n) = \Theta(n^{\log_b a} \log n)$
 - Obě funkce jsou stejného rádu, proto výsledkem je $\Theta(f(n) \log n) = \Theta(n^{\log_b a} \log n)$
- MM3 – pokud $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pro nějakou konstantu $\varepsilon > 0$ a pokud $af\left(\frac{n}{b}\right) \leq d \cdot f(n)$ pro nějakou konstantu $d \in (0,1)$ a všechna $n \geq n_0$, pak $t(n) = \Theta(f(n))$
 - Opak prvního případu, $f(n)$ je polynomiálně větší, než $n^{\log_b a}$
- Odpovídá rekurzivnímu algoritmu, který dělí problém velikosti n na a částí velikosti n/b , na zpětné kombinování potřebuje $f(n)$ času
- **Nepokrývá** všechny možnosti (např. rovnice $t(n) = 2t(n/2) + n \log n$ není touto metodou řešitelná)
- **Příklady:**

i. $t(n) = 6t\left(\frac{n}{4}\right) + n$

$$a = 6, b = 4, n^{\log_4 6} \doteq n^{1.3} = \Omega(n) \Rightarrow f(n) = O(n^{\log_4 6 - 0.1})$$

$$\Rightarrow \text{MM1} - t(n) = \Theta(n^{\log_4 6})$$

ii. $t(n) = 2t\left(\frac{n}{2}\right) + n$

$$a = 2, b = 2, n^{\log_2 2} = n = \Theta(n)$$

$$\Rightarrow \text{MM2} - t(n) = \Theta(n \log n)$$

iii. $t(n) = 3t\left(\frac{n}{4}\right) + n^2$

$$a = 3, b = 4, n^{\log_4 3} \doteq n^{0.7} = o(n^2) \text{ a platí, že } 3 \cdot \left(\frac{n}{4}\right)^2 \leq cn^2 \text{ pro nějakou } c < 1$$

$$\Rightarrow \text{MM3} - t(n) = \Theta(n^2)$$

- **Substituční metoda** – nedá se použít k přímému nalezení asymptotického odhadu, ale lze k potvrzení odhadu již získaného

1. Formulujeme indukční hypotézu $t(n) = \Theta(g(n))$ pro nějakou fci $g(n)$ – odhad tvaru řešení
2. Pomocí matematické indukce nalezneme konstanty a ověříme správnost odhadnutého řešení

- Důkaz bude správný, pouze pokud jsme schopni dokázat přesný tvar induk. hypotézy

- Na určení horní i dolní meze

- Příklad: $t(n) = 2t\left(\frac{n}{2}\right) + n$

1. Horní odhad řešení – $t(n) \leq cn \log n$ – indukcí dokazujeme správnost, tedy že platí $t(n) = O(n \log n)$
2. Indukční krok (ověření, že nerovnost vyhovuje rekurenci):

- Předpokládejme, že nerovnost platí pro $\frac{n}{2}$ a dosaďme $t\left(\frac{n}{2}\right) \leq c \frac{n}{2} \log \frac{n}{2}$ do počáteční rovnice:

$$\begin{aligned} t(n) &\leq 2\left(c \frac{n}{2} \log \frac{n}{2}\right) + n = cn \log \frac{n}{2} + n = cn \log n - cn \log 2 + n \\ &= cn \log n - cn + n \leq cn \log n, \text{ pokud } c \geq 1 \end{aligned}$$

3. Základní krok – je třeba dokázat, že existuje konstanta $c \geq 1$ taková, že nerovnost přímo platí pro počáteční hodnoty n (uvažujeme ve tvaru $n = 2^k$)

- Pro jednoduchost předpokládejme, že platí $t(1) = 1$. Pak z počáteční rovnice plyne $t(2) = 4$ a $t(4) = 12$

- Pak ale nerovnost neplatí pro $n = 1$, neboť neexistuje konstanta c taková, že $t(1) \leq c \cdot 1 \cdot \log 1 = 0$
 - Řešíme rci asymptoticky, takže stačí ukázat, že nerovnost platí od nějakého $n_0 \geq 1$
 - Stačí proto ukázat, že lze zvolit dostatečně velkou konstantu c , aby platilo $t(2) = 4 \leq c \cdot 2 \cdot \log 2$ nebo $t(4) = 12 \leq c \cdot 4 \cdot \log 4$
 - Triviálně lze spočítat, že stačí zvolit $c \geq 2$
 - Naprosto analogickým způsobem lze ukázat, že pro řešení počáteční rovnice platí $t(n) \geq c \cdot n \log n$, kde $c > 0$ je vhodně zvolená konstanta
- $$t(n) \geq 2 \left(c \frac{n}{2} \log \frac{n}{2} \right) + n = cn \log \frac{n}{2} + n = cn \log n - cn \log 2 + n \\ = cn \log n - cn + n \geq cn \log n, \text{ pokud } c \leq 1$$
- Z indukčního základu odvodíme např. $c = \frac{1}{2}$

4. Závěr – rovnice má řešení $\Theta(n \log n)$

- Lineární rekurentní rovnice řádu k s konstantními koeficienty – libovolná rekurentní rovnice ve tvaru

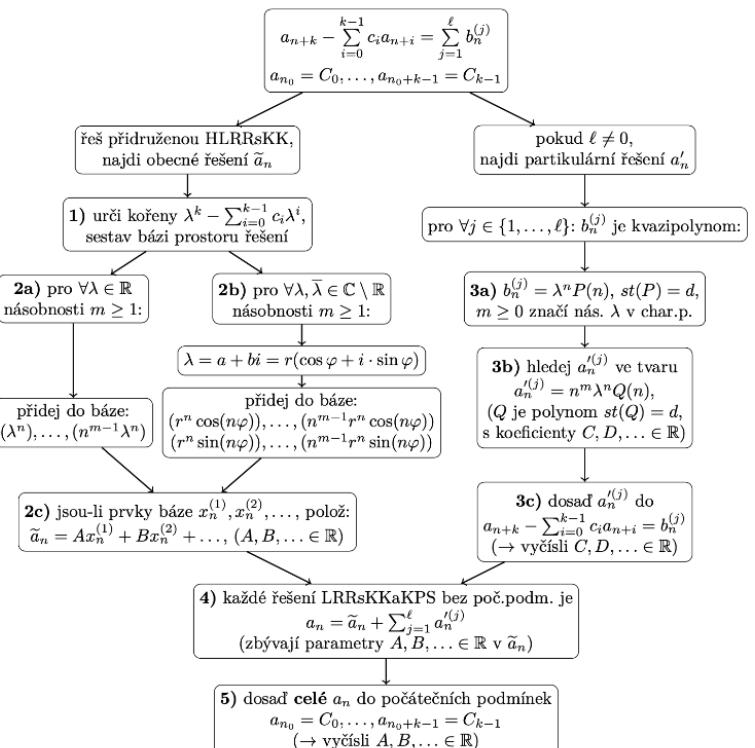
$$a_{n+k} + c_{k-1} \cdot a_{n+k-1} + \cdots + c_1 \cdot a_{n+1} + c_0 \cdot a_n = b_n$$

pro všechna $n \geq n_0$, $n_0 \in \mathbb{Z}$, $c_i \in \mathbb{R}$ pro $i = 0, \dots, k-1$ jsou pevně zvolená čísla, $c_0 \neq 0$ a $(b_n)_{n=n_0}^{\infty}$ je pevně zvolená reálná posloupnost

- Charakteristický polynom** této rovnice = polynom $p(\lambda) = \lambda^k + c_{k-1} \cdot \lambda^{k-1} + \cdots + c_1 \cdot \lambda + c_0$
 - Kořeny charakteristického polynomu se nazývají charakteristická čísla, popř. **vlastní čísla** dané rovnice
- Řešení** - jestliže je λ charakteristickým číslem homogenní linární rekurentní rovnice s konstantními koeficienty $a_{n+k} + c_{k-1} \cdot a_{n+k-1} + \cdots + c_1 \cdot a_{n+1} + c_0 \cdot a_n = 0$ pro všechna $n \geq n_0$, pak posloup. $(\lambda)_{n=n_0}^{\infty}$ je jejím řešením
- Příklad – HLRR 2. řádu $a_{n+2} - 3a_{n+1} + 2a_n = 0$ pro $n \geq 1$ s počátečními podm. $a_1 = -1, a_2 = 2$
 - Najdeme obecné řešení pomocí kořenů charakteristického polynomu
$$p(\lambda) = \lambda^2 + 3\lambda + 2 = (\lambda - 2)(\lambda - 1)$$
 - Kořeny jsou $\lambda_1 = 2, \lambda_2 = 1$, takže obecné řešení má tvar
$$(u \cdot 2^n + v \cdot 1^n)_{n=1}^{\infty} = (u \cdot 2^n + v)_{n=1}^{\infty} \text{ pro lib. } u, v \in \mathbb{R}$$
 - Zbývá nalézt řešení splňující zadané počáteční podmínky – tak dostaneme $u = \frac{3}{2}$ a $v = -4$

Řešení HLRRsKK – obecný případ

- Určíme charakteristický polynom $p(\lambda)$ a jeho kořeny.
- Pro každé reálné charakteristické číslo λ přidáme do báze řešení B posloupnost (λ^n) .
- Je-li toto λ násobnosti $m > 1$, pak přidáme také posloupnosti $(n \cdot \lambda^n), (n^2 \cdot \lambda^n), \dots, (n^{m-1} \cdot \lambda^n)$
- Pro každé charakteristické číslo $\lambda = r(\cos(\varphi) + i \cdot \sin(\varphi))$, které není reálné, přidáme do báze řešení B posloupnosti $(r^n \cos(n\varphi)), (r^n \sin(n\varphi))$ a $(r^{n-1} \cos(n\varphi)), (r^{n-1} \sin(n\varphi))$.
- Je-li toto λ násobnosti $m > 1$, pak přidáme také posloupnosti $((nr^n \cos(n\varphi)), (nr^n \sin(n\varphi))), \dots, ((n^{m-1}r^n \cos(n\varphi)), (n^{m-1}r^n \sin(n\varphi)))$.
- Je-li báze B tvořena posloupnostmi $(^1a_n), (^2a_n), \dots, (^ka_n)$, pak je obecné řešení dané rovnice určeno vzorcem $(\sum_{i=1}^k u_i \cdot ^i a_n)_{n=n_0}^{\infty}$ pro $u_1, u_2, \dots, u_k \in \mathbb{R}$.
- Jsou-li dány počáteční podmínky $a_j = A_j$ pro $j = n_0, \dots, n_0 + k - 1$, pak do nich dosadíme $a_j = \sum_{i=1}^k u_i \cdot ^i a_j$ a vyřešíme vzniklých k rovnic.



Modulární aritmetika, základy teorie čísel, Malá Fermatova věta, diofantické rovnice, lineární kongruence, Čínská věta o zbytcích

BI-ZDM

- Modulární aritmetika

- Množina $(\mathbb{Z}_m, \oplus, \odot)$, kde \oplus , resp. \odot označují sčítání, resp. násobení modulo m , tvoří konečný komutativní okruh s jednotkou
 - Platí uzavřenosť, komutativita, asociativita, neutrální prvek, inverzní prvek, distributivita
- Jedná se prakticky o normální aritmetiku s tím, že všechny operace jsou modulo m (tzn. $[a+b]_m$ atd.)
- Odčítání je sčítání s aditivní inverzí modulo m : $a \ominus b = a \oplus \bar{b}$.
- Pokud m není prvočíslo, nenulový prvek a nemusí mít multiplikativní inverzi ($b = a^{-1}(\text{mod } m)$):
 - Nechť $a \in \mathbb{Z}_m$. Potom existuje jediné celé číslo $b \in \mathbb{Z}_m$, které vyhovuje rovnici
$$a \odot b = b \odot a = 1 \Leftrightarrow \gcd(a, m) = 1$$
 - Číslo b je **multiplikativní inverze** a modulo m - $b = a^{-1}(\text{mod } m)$

- Kongruence – nechť $a, b, m \in \mathbb{Z}$, přičemž $m > 1$

- $m|(a - b) \rightarrow a$ je **kongruentní** s b modulo m : $a \equiv b \pmod{m}$
- $m \nmid (a - b) \rightarrow a$ **není kongruentní** s b modulo m : $a \not\equiv b \pmod{m}$
- Relace kongruence $\equiv \pmod{m}$ představuje ekvivalence na množině celých čísel \mathbb{Z} .
 - Reflexivita – $a \equiv a \pmod{m}$
 - Symetrie – $a \equiv b \pmod{m} \Leftrightarrow b \equiv a \pmod{m}$
 - Tranzitivita – $a \equiv b \pmod{m} \wedge b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$

- Zbytkové třídy – Ekvivalence $\equiv \pmod{m}$ definuje na množině \mathbb{Z} rozklad do tříd ekvivalence $\mathcal{R}_0, \mathcal{R}_1, \dots, \mathcal{R}_{m-1}$, kde $\mathcal{R}_k = \{b \in \mathbb{Z} : b \equiv k \pmod{m}\}$

- **Každou třídu ekvivalence \equiv** lze reprezentovat nejmenším nezáporným číslem, které obsahuje, tedy $\mathcal{R}_k = [k]_m$. Dostáváme tak rozklad množiny celých čísel do tzv. zbytkových tříd modulo m :

$$\mathbb{Z}_m := \mathbb{Z}_{/\equiv} = \{[0]_m, [1]_m, \dots, [m-1]_m\}$$

nebo zjednodušeně $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$.

- Teorie čísel

- **Dělitelnost** – nechť $a, b \in \mathbb{Z}$. Řekneme, že a dělí b ($a|b$), jestliže existuje $k \in \mathbb{Z}$ takové, že $b = k \cdot a$.
 - V takovém případě říkáme, že a je dělitel b , b je násobek a , také říkáme, že b je dělitelné a . Pokud a nedělí b , píšeme $a \nmid b$
 - Relace částečného uspořádání – reflexivní, tranzitivní, antisimetrické
 - Věta o **dělení se zbytkem** – nechť $a \in \mathbb{Z}$ a $d \in \mathbb{N}^+$. Pak existují jednoznačně určená čísla $q \in \mathbb{Z}$ a $r \in \mathbb{N}$ taková, že $a = qd + r$ a $0 \leq r < d$.
 - q = celočíselný **podíl**
 - r = **zbytek po dělení** a číslem d
- **GCD a LCM** – nechť $a, b \in \mathbb{Z}$
 - Číslo $d \in \mathbb{N}^+$ je **společný dělitel** čísel a, b , jestliže $d|a$ a $d|b$
 - Číslo $d \in \mathbb{N}^+$ je **největší společný dělitel** čísel a, b :
 - Pokud je aspoň jedno z čísel a, b nenulové, a d je největší z jejich společných dělitelů:
$$d = \gcd(a, b)$$
 - $d = 0$ pokud a, b jsou nulová: $\gcd(0, 0) = 0$
 - Čísla a, b nazýváme **nesoudělná**, jestliže $\gcd(a, b) = 1$
 - Číslo $n \in \mathbb{N}^+$ je **společný násobek** čísel a, b , jestliže $a|n$ a $b|n$
 - Číslo $n \in \mathbb{N}^+$ je **nejmenší společný násobek** čísel a, b :
 - Pokud jsou obě čísla a, b nenulová a n je nejmenší z jejich společných násobků:
$$n = \text{lcm}(a, b)$$
 - Jinak $n = \text{lcm}(0, 0) = \text{lcm}(a, 0) = \text{lcm}(0, b) = 0$

- Největší společný dělitel celých nenulových čísel a a b je nejmenší kladné celé číslo vyjádřené jako $m \cdot a + n \cdot b$, kde $m, n \in \mathbb{Z}$
 - **Bezoutova rovnost** – $\gcd(a, b) = m \cdot a + n \cdot b$ pro jistá $m, n \in \mathbb{Z}$
- **Eukleidův algoritmus** pro nalezení gcd dvou kladných celých čísel – nechť a, b jsou celá čísla, pro která platí $a \geq b > 0$. Nechť $(r_n)_{n=0}^{k+1}$ je klesající posloupnost zbytků definovaná rekurentním vztahem

$$r_{n+2} = r_n \text{ mod } r_{n+1} \text{ s počátečními podmínkami } r_0 = a, r_1 = b$$

kde $r_{k+1} = 0$ pro $k > 0$ je její první nulový člen.

Potom její poslední nenulový člen (poslední nenulový zbytek) je největším společným dělitelem a a b , tedy $\gcd(a, b) = r_k$.

- **Rozšířený Euklidův algoritmus:**

Nechť $a \geq b > 0$, hledáme $\alpha, \beta \in \mathbb{Z}$ takové, že $\gcd(a, b) = \alpha \cdot a + \beta \cdot b$.

r_j	α_j	β_j	q_j
a	1	0	–
b	0	1	$q_1 = \lfloor \frac{a}{b} \rfloor$
$r_2 = a - q_1 \cdot b$	$\alpha_2 = 1 - q_1 \cdot 0$	$\beta_2 = 0 - q_1 \cdot 1$	$q_2 = \lfloor \frac{b}{r_2} \rfloor$
...
$r_{i-2} - q_{i-1} \cdot r_{i-1}$	$\alpha_{i-2} - q_{i-1} \cdot \alpha_{i-1}$	$\beta_{i-2} - q_{i-1} \cdot \beta_{i-1}$	$\lfloor r_{i-1}/r_i \rfloor$
...
r_k	α_k	β_k	q_k
0	–	–	–

Potom $\gcd(a, b) = r_k$, $\alpha = \alpha_k$ a $\beta = \beta_k$, čili

$$\gcd(a, b) = \alpha_k \cdot a + \beta_k \cdot b.$$

- **Prvočíslo** – číslo dělitelné pouze samo sebou a 1 – ostatní čísla jsou **složená**

- Existuje nekonečně mnoho prvočísel
- Nechť $a, b, c \in \mathbb{Z}$. Jestliže $a|bc$ a a a b jsou nesoudělná, pak $a|c$
- Každé složené číslo můžeme psát ve tvaru součinu prvočísel
- **Kanonický rozklad** – každé číslo n se dá vyjádřit ve tvaru $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \dots \cdot p_k^{\alpha_k}$, kde p jsou prvočísla a α jsou přirozená čísla
- Funkce $\pi(n): \mathbb{N}^+ \mapsto \mathbb{N}$ určuje počet prvočísel, která jsou menší než n .
- Věta o počtu prvočísel – poměr $\pi(n)$ k výrazu $\frac{n}{\log(n)}$ se s rostoucím n přibližuje hodnotě 1, tj.

$$\lim_{n \rightarrow \infty} \pi(n) \cdot \frac{\log(n)}{n} = 1$$

- Nechť $a, b, c \in \mathbb{Z}$ a $m \in \mathbb{N}^+$ a nechť platí $ac \equiv bc \pmod{m}$. Potom platí $a \equiv b \pmod{\frac{m}{d}}$, kde d je největší společný dělitel čísel m a c .
- **Malá Fermatova věta** – nechť p je prvočíslo a $a \in \mathbb{N}^+$ takové přirozené číslo, které není násobkem p (tedy $\gcd(a, p) = 1$). Potom platí

$$a^{p-1} \equiv 1 \pmod{p}$$

- Důkaz – sporem z předpokladu, že čísla $1, \dots, p-1$ jsou nesoudělná
- Když p je prvočíslo a $a \in \mathbb{N}^+$, potom $a^p \equiv a \pmod{p}$
- Pokud je p prvočíslo a číslo $a \in \mathbb{N}^+$ není jeho násobkem, pak a^{p-2} je multiplik. inverzí č. $a \pmod{p}$
- Příklad: $5^{203} \pmod{101}$
 - Ověříme, že $\gcd(101, 5) = 1$
 - Z MFV víme, že platí $5^{100} \equiv 1 \pmod{101}$
 - $5^{203} = (5^{100})^2 \cdot 5^3 \equiv 1^2 \cdot 5^3 \equiv 1 \cdot 125 \equiv 24 \pmod{101}$

- **Eulerova věta** – zobecněná MFV

- Nechť $m \in \mathbb{N}^+$ a $a \in \mathbb{Z}$ je číslo nesoudělné s m . Potom platí

$$a^{\Phi(m)} \equiv 1 \pmod{m}$$

kde $\Phi(m)$ je Eulerova funkce

- Eulerova funkce $\Phi(n)$: $\mathbb{N}^+ \mapsto \mathbb{N}^+$ udává počet kladných celých čísel menších nebo rovných n , která jsou nesoudělná s n
- Přirozené číslo p je prvočíslem, právě když platí $\Phi(p) = p - 1$
- Nechť p je prvočíslo a $a \in \mathbb{N}$. Potom $\Phi(p^a) = p^a - p^{a-1}$
- Nechť $m, n \in \mathbb{N}$ a $\gcd(m, n) = 1$. Potom $\Phi(m \cdot n) = \Phi(m) \cdot \Phi(n)$
- Nechť $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ je kanonický rozklad složeného čísla $n \in \mathbb{N}^+$. Potom

$$\Phi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

- Lineární diofantická rovnice – libovolná rovnice typu

$$ax + by = c$$

s neznámými x, y , kde $a, b, c \in \mathbb{Z}$, pro jejíž řešení rovněž platí $x, y \in \mathbb{Z}$.

- Má alespoň jedno řešení právě tehdy, když $\gcd(a, b) | c$
- Věta o konstrukci všech řešení – Nechť a, b jsou nenulová celá čísla a dvojice (x_0, y_0) je řešením rovnice $ax + by = c$. Potom množina všech celočíselných řešení této rovnice je

$$\left\{ \left(x_0 + \frac{b}{\gcd(a, b)} \cdot k, y_0 - \frac{a}{\gcd(a, b)} \cdot k \right) : k \in \mathbb{Z} \right\}$$

- Lineární kongruence

- Úloha lineární kongruence – pro daná celá čísla a, b a $m > 1$ hledáme celé číslo x takové, že platí

$$ax \equiv b \pmod{m}$$

- Lineární kongruence má řešení právě tehdy, když $\gcd(a, m) | b$. Všechna řešení jsou tvaru

$$x = x_0 + k \cdot \frac{m}{\gcd(a, m)}$$

kde k je libovolné celé číslo a pro x_0 existuje y_0 takové, že dvojice (x_0, y_0) je řešením rovnice $a \cdot x + m \cdot y = b$

- Věta o **existenci a počtu řešení lineární kongruence** – jestliže $\gcd(a, m) | b$, potom kongruence $ax \equiv b \pmod{m}$ má konečně mnoho řešení modulo m . Těchto řešení je **přesně** $\gcd(a, m)$ a jsou dána výrazem

$$\left[x_0 + k \cdot \frac{m}{\gcd(a, m)} \right]_m$$

Pro $k = 1, 2, 3, \dots, \gcd(a, m)$, kde x_0 je z dvojice (x_0, y_0) , které je nějakým řešením rovnice

$$a \cdot x + m \cdot y = b$$

- Příklad: Řešíme $6x \equiv 12 \pmod{15}$, máme tedy $a = 6$, $b = 12$ a $m = 15$.

- ① Určíme $\gcd(a, m) = \gcd(6, 15) = 3$.
- ② $3|12$, tedy lineární kongruence má řešení.
- ③ Hledáme u a v takové, že $6u + 15v = 3$.
- ④ Vezmeme $u = -2$ a $v = 1$. Dosadíme do předpisu pro x_0 :

$$x_0 = \frac{ub}{\gcd(a, m)} = \frac{-2 \cdot 12}{3} = -8 \equiv 7 \pmod{15}.$$

- ⑤ Ověříme $6 \cdot 7 = 42 = 30 + 12 \equiv 12 \pmod{15}$,
- ⑥ Všechna řešení $\pmod{15}$ jsou (platí pro ně $x \equiv 7 \equiv 2 \pmod{15/3}$ tedy $x \equiv 2 \pmod{5}$)

$$[\{7 + 1.5, 7 + 2.5, 7 + 3.5\}]_{15} = [\{12, 17, 22\}]_{15} = \{12, 2, 7\}$$
- ⑦ Celkem všechna řešení v \mathbb{Z}_{15} tedy jsou 2, 7 a 12.

- Čínská věta o zbytcích – řešíme systém lineárních kongruencí:

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

...

$$x \equiv a_N \pmod{m_N}$$

kde čísla m_i jsou po dvou nesoudělná, tedy $\gcd(m_i, m_j) = 1$ pro všechna i, j , $i \neq j$

- Řešení tohoto systému existuje a všechna řešení jsou kongruentní modulo M (tedy v \mathbb{Z}_M je řešení určeno jednoznačně), kde

$$M = \prod_{i=1}^N m_i$$

- M je součin všech modulo

- Definujme $M_i = \frac{M}{m_i}$. Jelikož $\gcd(m_i, M_i) = 1$, pak existují řešení X_i lineárních kongruencí

$$M_i X_i \equiv 1 \pmod{m_i} \quad \forall i \in \{1, \dots, N\},$$

navíc platí pro všechna $j \neq i$: $M_i X_i \equiv 0 \pmod{m_j}$. Je tedy jasné, že

$$x \equiv a_1 X_1 M_1 + \dots + a_N X_N M_N \pmod{M}$$

je řešením našeho systému kongruencí.

- Příklad:

Řešíme

$$x \equiv 1 \pmod{2}, \quad x \equiv 2 \pmod{3}, \quad x \equiv 3 \pmod{5}.$$

$$M = 2 \cdot 3 \cdot 5 = 30, \quad M_1 = 3 \cdot 5 = 15, \quad M_2 = 2 \cdot 5 = 10, \quad M_3 = 2 \cdot 3 = 6.$$

Řešíme $M_1 X_1 = 15 X_1 \equiv 1 \pmod{2}$. Tedy $X_1 = 1$.

Obdobně $M_2 X_2 = 10 X_2 \equiv 1 \pmod{3}$ a $M_3 X_3 = 6 X_3 \equiv 1 \pmod{5}$. Obě řešení lze uhodnout: $X_2 = X_3 = 1$. Konečně

$$x = \underbrace{1 \cdot 1 \cdot 15}_{a_1 X_1 M_1} + \underbrace{2 \cdot 1 \cdot 10}_{a_2 X_2 M_2} + \underbrace{3 \cdot 1 \cdot 6}_{a_3 X_3 M_3} = 53 \equiv 23 \pmod{30}.$$

Ověříme

$$23 \equiv 1 \pmod{2}, \quad 23 \equiv 2 \pmod{3}, \quad 23 \equiv 3 \pmod{5}.$$

- Zobecněná Čínská věta o zbytcích – systém lineárních kongruencí:

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

...

$$x \equiv a_N \pmod{m_N}$$

má řešení právě tehdy, když $\gcd(m_i, m_j)$ dělí $a_i - a_j$ pro všechna $i, j: 1 \leq i < j \leq N$. Pokud řešení existuje, je určeno jednoznačně modulo $\text{lcm}(m_1, m_2, \dots, m_N)$

- Pro všechny páry kongruence musíme zkontrolovat, že \gcd jejich modulů dělí rozdíl zbytků. (tzn. $\gcd(m_1, m_2)|a_1 - a_2 \rightarrow$ pokud ano, pak existuje řešení).
- Příklad:

Řešíme systém lineárních kongruencí

$$x \equiv 5 \pmod{6}, \quad x \equiv 3 \pmod{10}, \quad x \equiv 8 \pmod{15}$$

Podmínky řešitelnosti jsou splněny, takže začneme řešením kongruence $x \equiv 5 \pmod{6}$ ve tvaru $x = 5 + 6t$.

Dosazením do druhé kongruence dostaneme $5 + 6t \equiv 3 \pmod{10}$ neboli $6t \equiv 8 \pmod{10}$, takže

$$t \equiv 3 \pmod{10} \quad (\text{dokonce } t \equiv 3 \pmod{5}, \text{ ale to není podstatné}).$$

To znamená, že platí $t = 3 + 10u$, takže dostáváme

$$x = 5 + 6t = 5 + 6(3 + 10u) = 23 + 60u$$

a můžeme dosadit do třetí kongruence ...

$$23 + 60u \equiv 8 \pmod{15}.$$

Po úpravě dostaneme $8 + 0 \cdot u \equiv 8 \pmod{15}$, což splňuje libovolné u . Dostali jsme výsledek $x = 5 + 6t = 23 + 60u$. Víme, že řešení je jednoznačné modulo $\text{lcm}(6, 10, 15) = 30$, takže

$$x \equiv 23 \pmod{30}.$$

Ověříme výsledek:

$$23 \equiv 5 \pmod{6}, \quad 23 \equiv 3 \pmod{10}, \quad 23 \equiv 8 \pmod{15}$$

Limita a derivace funkce (definice a vlastnosti, geometrický význam), využití při vyšetřování průběhu funkce

BI-ZMA

- **Okolí bodu** – Nechť $a \in \mathbb{R}$ a $\varepsilon > 0$. Otevřený interval $(a - \varepsilon, a + \varepsilon)$ nazýváme okolím bodu a o poloměru ε a značíme $H_a(\varepsilon)$. Někdy též o této množině mluvíme jako o ε -okolí bodu a .
 - o **Jednostranné okolí** – Nechť $a \in \mathbb{R}$ a $\varepsilon > 0$. Polouzavřený interval $(a, a + \varepsilon]$, resp. $(a - \varepsilon, a)$, nazýváme **pravým**, resp. **levým**, ε okolím bodu a a značíme ho $H_a^+(\varepsilon)$ resp. $H_a^-(\varepsilon)$
- **Limita funkce** – Buďte f reálná funkce reálné proměnné a $a \in \mathbb{R}$. Nechť f je definovaná na okolí bodu a , s možnou výjimkou bodu a samotného. Řekneme, že $c \in \mathbb{R}$ je **limitou funkce f v bodě a** , právě když pro každé okolí H_c bodu c existuje okolí H_a bodu a takové, že z podmínky

$$x \in H_a \setminus \{a\}$$

plyne

$$f(x) \in H_c$$

V symbolech

$$(\forall H_c)(\exists H_a)(\forall x \in D_f)(x \in H_a \setminus \{a\} \Rightarrow f(x) \in H_c)$$

Tuto skutečnost zapisujeme

$$\lim_{x \rightarrow a} f(x) = c, \text{ případně } \lim_{a} f = c$$

- o Hodnota limity funkce f v bodě a závisí pouze na chování funkce f na okolí bodu a mimo bod a (ten nemusí být ani v D_f , např. $\operatorname{sgn}(\frac{1}{x^2})$ pro 0).
- o ε - δ definice – v případě když a i c jsou prvky \mathbb{R} podmínka

$$\lim_{x \rightarrow a} f(x) = c$$

ekviv. požadavku, aby f byla definována na okolí bodu a s možnou výjimkou bodu a samotného a

$$(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x \in D_f)(0 < |x - a| < \delta \Rightarrow |f(x) - c| < \varepsilon)$$

Analogické formule lze formulovat pro různé kombinace případů $a, c \in \mathbb{R}$.

- o **Jednostranná limita funkce** – buďte f reálná funkce reálné proměnné a $a \in \mathbb{R}$. Nechť f je definovaná na levém, resp. pravém okolí bodu a s možnou výjimkou bodu a samotného. Řekneme, že $c \in \bar{\mathbb{R}}$ je **limitou funkce f v bodě a zleva**, resp. **zprava**, právě když pro každé okolí H_c bodu c existuje levé okolí H_a^- , resp. pravé okolí H_a^+ , bodu a takové, že z podmínky

$$x \in H_a^- \setminus \{a\}, \text{ resp. } x \in H_a^+ \setminus \{a\},$$

plyne

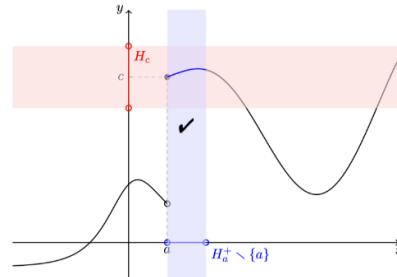
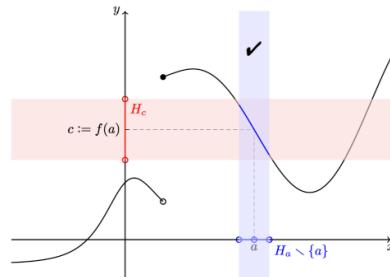
$$f(x) \in H_c$$

Zapisujeme

$$\lim_{x \rightarrow a^-} f(x) = c, \text{ případně } \lim_{a^-} f = c$$

resp.

$$\lim_{x \rightarrow a^+} f(x) = c, \text{ případně } \lim_{a^+} f = c$$



- o **Vlastnosti limit funkcí**

- Nechť $a \in \mathbb{R}$ $\lim_{x \rightarrow a} f(x)$ existuje a je rovna $c \in \bar{\mathbb{R}}$, právě když existují obě jednostranné limity $\lim_{x \rightarrow a_+} f(x)$ a $\lim_{x \rightarrow a_-} f(x)$ a obě jsou rovny c

- Pokud se levá a pravá limita nerovnají, pak oboustranná limita neexistuje
- **Heineho věta** – $\lim_{x \rightarrow a} f(x) = c$, právě když je f definována na okolí bodu a (s možnou výjimkou bodu a) a pro každou posloupnost $(x_n)_{n=1}^{\infty}$ s limitou a a splňující $\{x_n | n \in \mathbb{N}\} \subset D_f \setminus \{a\}$ platí $\lim_{n \rightarrow \infty} f(x_n) = c$
 - **Heine pro jednostranné limity** – $\lim_{x \rightarrow a_-} f(x) = c$, resp. $\lim_{x \rightarrow a_+} f(x) = c$, právě když je f definována na levém, resp. pravém, okolí bodu a a pro každou posloupnost $(x_n)_{n=1}^{\infty}$ s limitou a a splňující

$$\{x_n | n \in \mathbb{N}\} \subset D_f \setminus \{a\} \cap (-\infty, a), \text{ resp. } \{x_n | n \in \mathbb{N}\} \subset D_f \cap (a, +\infty)$$

$$\text{platí } \lim_{n \rightarrow \infty} f(x_n) = c$$
 - Důsledek – nechť f je funkce definovaná na okolí bodu $a \in \bar{\mathbb{R}}$ a $(x_n)_{n=1}^{\infty}, (z_n)_{n=1}^{\infty}$ jsou dvě reálné posloupnosti patřící do D_f , konvergující k a a splňující podmínky $x_n \neq a$ a $z_n \neq a$ pro všechna $n \in \mathbb{N}$. Pokud limity

$$\lim_{n \rightarrow \infty} f(x_n) \text{ a } \lim_{n \rightarrow \infty} f(z_n)$$
 existují a jsou různé, nebo alespoň jedna z nich neexistuje, potom limita $\lim_{x \rightarrow a} f(x)$ neexistuje.
- Věta o **limitě součtu, součinu a podílu** – nechť f a g jsou funkce a a prvek $\bar{\mathbb{R}}$. Potom

$$\lim_a (f + g) = \lim_a f + \lim_a g$$

$$\lim_a f \cdot g = \lim_a f \cdot \lim_a g$$

$$\lim_a \frac{f}{g} = \frac{\lim_a f}{\lim_a g}$$
 platí v případě, že výrazy na pravé straně jsou definovány a v posledním případě za předpokladu, že $\frac{f}{g}$ je definovaná na okolí bodu a s možnou výjimkou bodu a samotného.
- Věta o **limitě složené funkce** – nechť f a g jsou funkce a a, b, c prvky $\bar{\mathbb{R}}$ a platí tři podmínky
 - $\lim_{x \rightarrow b} f(x) = c$
 - $\lim_{x \rightarrow a} g(x) = b$
 - Bud' $(\exists H_a)(\forall x \in D_g \cap H_a \setminus \{a\})(g(x) \neq b)$ nebo $(b \in D_f \text{ a } f(b) = c)$
 Potom $\lim_{x \rightarrow a} f \circ g(x) = c$
 - $g(x)$ jde k a , limita je $b \rightarrow$ vstup $f(x)$ jde k b a ten má limitu c
 - Aby to fungovalo, tak bud' $b \notin D_f$ (tzn. g se do něj nikdy nesmí dostat), nebo $b \in D_f$, tzn. je spojitá
- **Nerovnost limit**
 - Nechť existují limity $\lim_{x \rightarrow a} f(x)$ a $\lim_{x \rightarrow a} g(x)$. Pak platí dvě tvrzení:
 - Pokud $\lim_{x \rightarrow a} f(x) < \lim_{x \rightarrow a} g(x)$, potom existuje okolí bodu H_a bodu a takové, že $\forall x \in H_a \setminus \{a\}: f(x) < g(x)$
 - Pokud existuje okolí bodu a takové, že $\forall x \in H_a \setminus \{a\}: f(x) \leq g(x)$,

$\forall x \in H_a \setminus \{a\}: f(x) \leq g(x),$

potom $\lim_{x \rightarrow a} f(x) \leq \lim_{x \rightarrow a} g(x)$.
 - Věta o limitě sevřené funkce – nechť pro funkce f, g, h a body $a, c \in \bar{\mathbb{R}}$ platí, že
 - existuje okolí H_a bodu a takové, že pro každé $x \in H_a \setminus \{a\}$ platí

$$f(x) \leq g(x) \leq h(x),$$
 - existují $\lim_{x \rightarrow a} f(x) = \lim_{x \rightarrow a} h(x) = c$.
 Potom existuje i limita $\lim_{x \rightarrow a} g(x)$ a je rovna c .

- **Spojitost funkce**
 - Nechť f je reálná funkce reálné proměnné a nechť bod $a \in D_f$. Řekneme, že funkce f je **spojitá v bodě a** , jestliže nastává jedna z následujících možností:
 - $\lim_{x \rightarrow a} f(x) = f(a)$
 - Funkce f je definována jen na pravém okolí bodu a , přesněji ($\exists H_a$) ($H_a \cap D_f = H_a^+$), a $\lim_{x \rightarrow a^+} f(x) = f(a)$
 - Funkce f je definována jen na levém okolí bodu a , přesněji ($\exists H_a$) ($H_a \cap D_f = H_a^-$), a $\lim_{x \rightarrow a^-} f(x) = f(a)$
 - Funkce f je **spojitá v bodě a zprava**, pokud $\lim_{x \rightarrow a^+} f(x) = f(a)$. Funkce f je **spojitá v bodě a zleva**, pokud $\lim_{x \rightarrow a^-} f(x) = f(a)$.
 - Funkce f definovaná na okolí bodu a je spojitá v bodě $a \in D_f$, právě když pro každé $\varepsilon > 0$ existuje $\delta > 0$ tak, že pro každé $x \in \mathbb{R}$ splňující $|x - a| < \delta$ platí $|f(x) - f(a)| < \varepsilon$
 - Důsledky vlastností limity funkce:
 - Funkce f definovaná na okolí bodu $a \in D_f$ je spojitá v bodě $a \in D_f$, právě když je spojitá v bodě a zleva i zprava.
 - Součet a součin dvou funkcí f, g definovaných na okolí bodu a a spojitých v bodě a je funkce spojitá v bodě a . Pokud navíc $g(a) \neq 0$, pak podíl $\frac{f}{g}$ je funkce spojitá v bodě a .
 - Buďte g funkce definovaná na okolí bodu a a spojitá v bodě a a f funkce definovaná na okolí bodu $g(a)$ a spojitá v bodě $g(a)$. Potom složená funkce $f \circ g$ je spojitá v bodě a .
 - **Spojitost na intervalu** – funkce f je spojitá na intervalu J , právě když je spojitá v každém bodě interv. J .
- **Derivace funkce** – nechť f je funkce definovaná na okolí bodu $a \in \mathbb{R}$. Pokud existuje limita

$$\lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$

nazveme její hodnotu **derivací funkce f v bodě a** a označíme $f'(a)$. Pokud je tato funkce konečná ($f'(a) \in \mathbb{R}$), řekneme, že funkce f je **diferencovatelná v bodě a** .

- Bud' f funkce s definičním oborem D_f . Nechť M označuje množinu všech $x \in D_f$ takových, že existuje konečná derivace $f'(x)$. **Derivací funkce f** nazýváme funkci s definičním oborem M , která každému $x \in M$ přiřadí $f'(x)$. Tuto funkci značíme symbolem f' .
- Je-li f funkce diferencovatelná v bodě a , pak je spojitá v bodě a . Tj. Platí

$$f'(a) \in \mathbb{R} \Rightarrow \lim_{x \rightarrow a} f(x) = f(a)$$

- **Derivace součtu, součinu a podílu** – nechť funkce f a g jsou diferencovatelné v bodě a . Potom platí
 - $(f + g)'(a) = f'(a) + g'(a)$
 - $(f \cdot g)'(a) = f'(a)g(a) + f(a)g'(a)$
 - $\left(\frac{f}{g}\right)'(a) = \frac{f'(a)g(a) - f(a)g'(a)}{g(a)^2}$, pokud $g(a) \neq 0$

- **Derivace složené funkce** – nechť g je funkce diferencovatelná v bodě a , f je diferencovatelná v bodě $g(a)$. Potom funkce $f \circ g$ je diferencovatelná v bodě a a platí

$$(f \circ g)'(a) = f'(g(a)) \cdot g'(a)$$

- **Derivace inverzní funkce** – buďť f spojitá a ryze monotonné na intervalu $I = (a, b)$ a bod $c \in I$. Má-li inverzní funkce f^{-1} konečnou nenulovou derivaci v bodě $f(c)$, potom má f derivaci v bodě c a platí

$$f'(c) = \frac{1}{(f^{-1})'(f(c))}$$

- Derivací funkce f dostáváme novou funkci f' , jejíž definiční obor ovšem může být menší, než původní D_f . Nyní můžeme znova derivovat f' , tj. Sestrojit f'' . Rekurzivně tedy definujeme

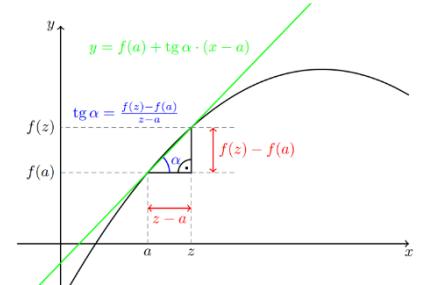
$$f^{(0)}(x) = f(x), f^{(n)}(x) = (f^{(n-1)})'(x), n \geq 1$$

- Derivace elementárních funkcí

$f(x)$	$f'(x)$	podmínky	$f(x)$	$f'(x)$	podmínky
x^n	nx^{n-1}	$x \in \mathbb{R}, n \in \mathbb{N}_0$	$\operatorname{tg}(x)$	$\frac{1}{\cos^2(x)}$	$x \neq \frac{\pi}{2} + k\pi, k \in \mathbb{Z}$
x^n	nx^{n-1}	$x \in \mathbb{R} \setminus \{0\}, n = -1, -2, \dots$	$\operatorname{cotg}(x)$	$-\frac{1}{\sin^2(x)}$	$x \neq k\pi, k \in \mathbb{Z}$
x^α	$\alpha x^{\alpha-1}$	$x > 0 \text{ a } \alpha \in \mathbb{R}$	$\arcsin(x)$	$\frac{1}{\sqrt{1-x^2}}$	$x \in (-1, 1)$
e^x	e^x	$x \in \mathbb{R}$	$\arccos(x)$	$-\frac{1}{\sqrt{1-x^2}}$	$x \in (-1, 1)$
a^x	$a^x \ln a$	$x \in \mathbb{R}, a > 0$	$\operatorname{arctg}(x)$	$\frac{1}{1+x^2}$	$x \in \mathbb{R}$
$\ln(x)$	$\frac{1}{x}$	$x > 0$	$\operatorname{arccotg}(x)$	$-\frac{1}{1+x^2}$	$x \in \mathbb{R}$
$\sin(x)$	$\cos(x)$	$x \in \mathbb{R}$			
$\cos(x)$	$-\sin(x)$	$x \in \mathbb{R}$			

- Geometrický význam derivace

- Tečna – hlavní geometrický význam limity (v podobě derivace)
 - Nechť existuje $f'(a)$. Tečnou funkce f v bodě a nazýváme
 - Přímku s rovnici $x = a$ je-li funkce f spojitá v bodě a a $f'(a) = +\infty$ nebo $f'(a) = -\infty$
 - Přímku s rovnici $y = f(a) + f'(a)(x - a)$, je-li $f'(a) \in \mathbb{R}$



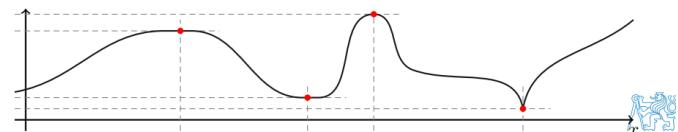
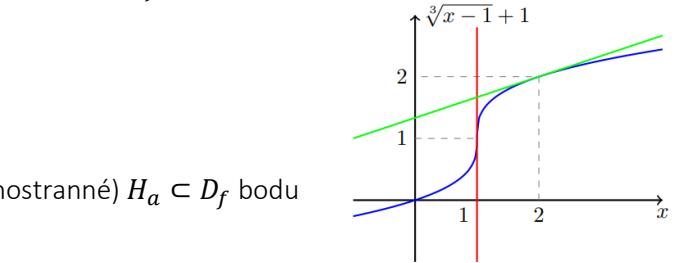
- Vyšetřování průběhu funkce

- Lokální extrémy – řekneme, že funkce f má v bodě $a \in D_f$

1. Lokální maximum
2. Lokální minimum
3. Ostré lokální maximum
4. Ostré lokální minimum

právě když existuje okolí (v krajním bodě jednostranné) $H_a \subset D_f$ bodu a tak, že:

1. $\forall x \in H_a: f(x) \leq f(a)$
2. $\forall x \in H_a: f(x) \geq f(a)$
3. $\forall x \in H_a \setminus \{a\}: f(x) < f(a)$
4. $\forall x \in H_a \setminus \{a\}: f(x) > f(a)$



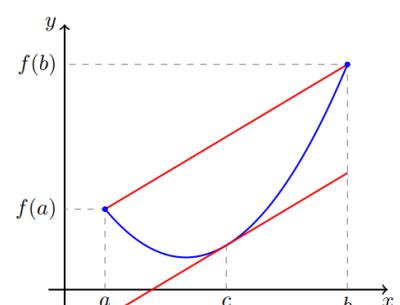
- Pokud má funkce f v bodě a lokální extrém, tak derivace $f'(a) = 0$, nebo neexistuje (tzn využití limity v podobě derivace). Je to ale pouze **nutná** podmínka, může se stát, že to vůbec lokální extrém není, pouze "sedlový bod".
- Globální extrémy mohou existovat pouze v krajních bodech definičního oboru a v bodech kde je derivace rovna 0 nebo neexistuje
- **Věty o přírůstku funkce**

- Rolleova věta – nechť funkce f splňuje podmínky
 - f je spojitá na intervalu (a, b)
 - f má derivaci v každém bodě intervalu (a, b)
 - $f(a) = f(b)$

Potom existuje $c \in (a, b)$ tak, že $f'(c) = 0$

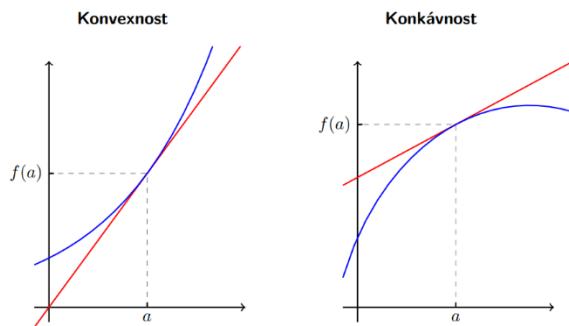
- Lagrangeova věta o přírůstku funkce – nechť funkce f splňuje podmínky
 - f je spojitá na intervalu $[a, b]$
 - f má derivaci v každém bodě intervalu (a, b)

Potom existuje bod $c \in (a, b)$ tak, že $f'(c) = \frac{f(b)-f(a)}{b-a}$



- Monotonie – první derivace funkce $f(x)$ určí, zda funkce v daném bodě roste, klesá nebo je konstantní → rozhoduje znaménko derivace
 - $(f'(x) \geq 0) \Rightarrow f$ je rostoucí na J
 - $(f'(x) \leq 0) \Rightarrow f$ je klesající na J
 - $(f'(x) > 0) \Rightarrow f$ je ostře rostoucí na J
 - $(f'(x) < 0) \Rightarrow f$ je ostře klesající na J
 - $(f'(x) = 0) \Rightarrow f$ je konstantní na J

- **Konvexnost a konkávnost** – zjišťuje se druhou derivací funkce $f(x)$
 - “určuje, na kterou stranu je vypouklá”: $> 0 \Rightarrow$ konvexní, $< 0 \Rightarrow$ konkávní
 - Funkci f definovanou na intervalu J nazveme **konvexní** na intervalu (resp. **konkávní** na intervalu) J , právě když pro každé $x_1, x_2, x_3 \in J$ splňující $x_1 < x_2 < x_3$, leží bod $(x_2, f(x_2))$ buďto pod (resp. nad) přímkou spojující body $(x_1, f(x_1))$ a $(x_3, f(x_3))$, nebo na ní.
 - Funkci f definovanou na intervalu J nazveme **ryze konvexní** na intervalu (resp. **ryze konkávní** na intervalu) J , právě když pro každé $x_1, x_2, x_3 \in J$ splňující $x_1 < x_2 < x_3$, leží bod $(x_2, f(x_2))$ pod (resp. nad) přímkou spojující body $(x_1, f(x_1))$ a $(x_3, f(x_3))$.
 - Nechť funkce f je diferencovatelná v bodě $a \in D_f$. Pokud existuje okolí H_a b. a takové, že pro všechna $x \in H_a \setminus \{a\}$ leží všechny body $(x, f(x))$ nad (resp. pod) tečnou funkce f v bodě a
- $y = f(a) + f'(a)(x - a)$,
- nebo na ní, pak f nazveme **konvexní (resp. konkávní) v bodě a**.
- Buď f funkce diferencovatelná v každém bodě intervalu J a nechť $f'(c) = 0$ pro jisté $c \in J^\circ$.
 - Pokud je f **konvexní** na intervalu J , pak má funkce f v bodě c **lokální minimum**.
 - Pokud je f **konkávní** na intervalu J , pak má funkce f v bodě c **lokální maximum**.



- **Inflexní bod** – nechť f je spojitá v bodě c . Bod c nazýváme inflexním bodem funkce f , právě když existuje $\delta > 0$ takové, že f je ryze konvexní na intervalu $(c - \delta, c)$ a ryze konkávní na intervalu $(c, c + \delta)$, nebo naopak
- **Asymptoty**
 - Řekneme, že funkce f má v bodě $a \in \mathbb{R}$ **asymptotu** $x = a$, právě když $\lim_{x \rightarrow a^+} f(x)$ nebo $\lim_{x \rightarrow a^-} f(x)$ existuje a je rovna $+\infty$ nebo $-\infty$
 - Řekneme, že přímka $y = kx + q$ je asymptotou funkce f v $+\infty$, resp. v $-\infty$, když $\lim_{x \rightarrow \infty} (f(x) - kx - q) = 0$ resp. $\lim_{x \rightarrow -\infty} (f(x) - kx - q) = 0$

- l'Hospitalovo pravidlo – k výpočtu limit funkcí vedoucích k neurčitým výrazům

- Nechť pro funkce f a g a bod $a \in \overline{\mathbb{R}}$ platí
 - $\lim_{a^-} f = \lim_{a^+} g = 0$ nebo $\lim_{a^-} |g| = +\infty$
 - Existuje okolí H_a bodu a splňující $H_a \setminus \{a\} \subset D_{f/g} \cap D_{f'/g'}$
 - Existuje $\lim_{a^-} \frac{f'}{g'}$
- Potom existuje $\lim_{a^-} \frac{f}{g}$ a platí $\lim_{a^-} \frac{f}{g} = \lim_{a^-} \frac{f'}{g'}$

Základy integrálního počtu (primitivní funkce, neurčitý integrál, Riemannův integrál (definice, vlastnosti a geometrický význam))

BI-ZMA

- **Primitivní funkce** – nechť funkce f je definována na intervalu (a, b) , kde $-\infty \leq a < b \leq +\infty$. Funkci F splňující podmínu

$$\forall x \in (a, b): F'(x) = f(x)$$

nazýváme primitivní funkci k funkci f na intervalu (a, b) .

- Nechť F je primitivní funkcií k funkci f na intervalu (a, b) . Pak G je primitivní funkcií k funkci f na intervalu (a, b) právě tehdy, když existuje konstanta $C \in \mathbb{R}$ taková, že

$$\forall x \in (a, b) : G(x) = F(x) + C$$

= nejednoznačnost primitivní funkce – derivováním se ztrácí konstanta z polynomu (pohyb po ose y), kterou zpětně (integrací) neumíme dostat (existuje nekonečně mnoho funkcí s posunem po ose y)

- Postačující podmínka existence primitivní funkce: Funkce f spojité na intervalu $(a, b) \Rightarrow$ funkce f má na tomto intervalu primitivní funkci
 - Základní primitivní funkce:

- #### ○ Základní primitivní funkce:

vzorec	interval, parametry	
$\int x^n dx = \frac{x^{n+1}}{n+1} + C$	$x \in \mathbb{R}, n \in \mathbb{N}_0$	
$\int x^n dx = \frac{x^{n+1}}{n+1} + C$	$x \in \mathbb{R} \setminus \{0\}, n \in \mathbb{Z}, n \leq -2$	
$\int x^\alpha dx = \frac{x^{\alpha+1}}{\alpha+1} + C$	$x \in (0, +\infty), \alpha \notin \mathbb{Z}$	
$\int \frac{1}{x} dx = \ln x + C$	$x \in \mathbb{R} \setminus \{0\}$	
$\int a^x dx = \frac{a^x}{\ln a} + C$	$x \in \mathbb{R}, a > 0 \text{ a } a \neq 1$	
$\int \sin(x) dx = -\cos(x) + C$	$x \in \mathbb{R}$	
		$\int \cos(x) dx = \sin(x) + C \quad x \in \mathbb{R}$
		$\int \frac{1}{\cos^2(x)} dx = \operatorname{tg}(x) + C \quad x \in (-\frac{\pi}{2} + k\pi, \frac{\pi}{2} + k\pi), k \in \mathbb{Z}$
		$\int \frac{1}{\sin^2(x)} dx = -\operatorname{cotg}(x) + C \quad x \in (k\pi, \pi + k\pi), k \in \mathbb{Z}$
		$\int \frac{1}{\sqrt{1-x^2}} dx = \arcsin(x) + C \quad x \in (-1, 1)$
		$\int \frac{1}{1+x^2} dx = \operatorname{arctg}(x) + C \quad x \in \mathbb{R}$

- **Neurčitý integrál** – nechť k funkci f existuje primitivní funkce na intervalu (a, b) . Množinu všech primitivních funkcí k funkci f na (a, b) nazýváme neurčitým integrálem a značíme jej $\int f$ nebo $\int f(x) dx$.

- Najdeme-li k f primitivní funkci F v intervalu (a, b) , zapisujeme tento fakt obvykle

$$\int f(x) dx = F(x) + c$$

- **Existence primitivní funkce** / inverze – pokud funkce g je diferencovatelná v (a, b) , tak z definice plyne $\int g'(x)dx = g(x) + c, x \in (a, b)$
 - **Sčítání a násobení** – $\int(f + g) = \int f + \int g$ a $\int(\alpha f) = \alpha \int f$
 - **Per partes** – nechť funkce f je diferencovatelná v intervalu (a, b) a G je primitivní funkce k funkci g na intervalu (a, b) a nechť existuje primitivní funkce k funkci $f'G$. Potom existuje primitivní funkce k funkci $f'g$ a platí

$$\int f g = f G - \int f' G$$

- Důkaz – přes derivaci součinu dvou funkcí

$$\int fg = \begin{vmatrix} f & g \\ f' & G \end{vmatrix} = \cancel{fG} \quad \int fg = \begin{vmatrix} f & g \\ f' & G \end{vmatrix} = \cancel{fG} - \int f'G$$

- **Substituce** v neurčitém integrálu – nechť pro funkce f a φ platí:

- f má primitivní funkci F na intervalu (a, b)
 - φ je na intervalu (α, β) diferencovatelná
 - $\varphi((\alpha, \beta)) \subset (a, b)$

pak funkce $f(\varphi(x)) \cdot \varphi'(x)$ má primitivní funkci na intervalu (α, β) a platí

$$\int f(\varphi(x)) \cdot \varphi'(x) \, dx = F(\varphi(x)) + C$$

- Jedná se vlastně o „derivaci složené funkce naruhy“

- Nechť f je definována na intervalu (a, b) a nechť φ je bijekce intervalu (α, β) na (a, b) s nenulovou konečnou derivací v každém bodě intervalu (α, β) . Pak platí
$$\int f(\varphi(t)) \cdot \varphi'(t) dt = G(t) + C \Rightarrow \int f(x) dx = G(\varphi^{-1}(x)) + C$$
 - **Bijekce** je zobrazení, které je prosté i na (tzn. zobrazení f , které přiřazuje každému prvku Hf právě jeden prvek z Df)
 - Zobrazení je **na**, když každý prvek cílové množiny má alespoň 1 vzor
 - Zobrazení je **prosté**, když každý prvek má svůj obraz
- **Riemannův integrál** – geometrická interpretace – určuje obsah plochy pod grafem
 - **Infimum** – buď A neprázdná zdola omezená podmnožina množiny reálných čísel. Číslo $\alpha \in \mathbb{R}$ nazveme infimum množiny A , značíme $\inf A$, právě když
 - Pro každé $x \in A$ platí $\alpha \leq x$ (α je **dolní závora** A)
 - Pokud $\beta \in \mathbb{R}$ také splňuje předchozí bod, pak $\beta \leq \alpha$ (α je největší dolní závora A)
Pokud množina A není zdola omezená, pak klademe $\inf A := -\infty$. Pro prázdnou množinu klademe $\inf \emptyset := +\infty$.
 - **Supremum** – buď A neprázdná shora omezená podmnožina množiny reálných čísel. Číslo $\alpha \in \mathbb{R}$ nazveme supremem množiny A , značíme $\sup A$, právě když
 - Pro každé $x \in A$ platí $x \leq \alpha$ (α je **horní závora** A)
 - Pokud $\beta \in \mathbb{R}$ také splňuje předchozí bod, pak $\alpha \leq \beta$ (α je nejmenší horní závora A)
Pokud množina A není shora omezená, pak klademe $\sup A := +\infty$. Pro prázdnou množinu klademe $\sup \emptyset := -\infty$.
 - Příklad:

Pro interval $J = (-2, 1)$ platí

$$\max J = 1, \quad \sup J = 1, \quad \min J \text{ neexistuje}, \quad \inf J = -2.$$
 - **Dělení intervalu a norma dělení** – buď dán interval $\langle a, b \rangle$. Konečnou množinu $\sigma = \{x_0, \dots, x_n\}$ takovou, že $a = x_0 < x_1 < \dots < x_n = b$ nazýváme **dělením intervalu** $\langle a, b \rangle$. Bodům $x_k, k = 1, 2, \dots, n-1$, říkáme dělící body intervalu $\langle a, b \rangle$. Intervalu $\langle x_{k-1}, x_k \rangle$ říkáme **k-tý částečný interval intervalu** $\langle a, b \rangle$ při dělení σ . Číslo
$$v(\sigma) := \max\{\Delta_k | k = 1, 2, \dots, n\},$$
 kde $\Delta_k := x_k - x_{k-1}, k = 1, 2, \dots, n$
 - nazýváme **normou dělení** σ
 - Prakticky jde o to, na kolik částí si daný interval nasekám
 - **Horní/dolní součet**
 - Buďte funkce f definovaná a omezená na intervalu $J = \langle a, b \rangle$ a $\sigma = \{x_0, \dots, x_n\}$ dělení intervalu J . Označme
$$M_i(\sigma, f) := \sup_{\langle x_{i-1}, x_i \rangle} f \quad \text{a} \quad m_i(\sigma, f) := \inf_{\langle x_{i-1}, x_i \rangle} f$$
pro každé $i = 1, 2, \dots, n$. Potom součty
$$S(\sigma, f) := \sum_{i=1}^n M_i(\sigma, f) \Delta_i \quad \text{a} \quad s(\sigma, f) := \sum_{i=1}^n m_i(\sigma, f) \Delta_i$$
nazýváme **horním**, resp. **dolním, součtem** funkce f při dělení σ .

- **Horní a dolní integrál** – pro funkci f definovanou a omezenou na uzavřeném intervalu $J = \langle a, b \rangle$ definujeme čísla

$$\overline{\int_a^b} f(x) dx := \inf\{S(\sigma) | \sigma \text{ dělení } J\} \text{ a } \underline{\int_a^b} f(x) dx := \sup\{s(\sigma) | \sigma \text{ dělení } J\}$$

a nazýváme horním, resp. dolním integrálem funkce f na intervalu J .

- **Riemannův integrál** – pokud pro funkci f definovanou a omezenou na uzavřeném intervalu J platí

$$\overline{\int_a^b} f(x) dx = \underline{\int_a^b} f(x) dx \in \mathbb{R},$$

pak jejich společnou hodnotu nazýváme **Riemannovým integrálem funkce f na intervalu J** a toto číslo značíme symboly

$$\int_a^b f, \quad \text{případně} \quad \int_a^b f(x) dx.$$

- Funkce f musí být na intervalu J spojitá
 - Posloupnost dělení σ_n nazveme **normální**, pokud pro její normy platí
- $$\lim_{n \rightarrow \infty} \nu(\sigma_n) = 0$$
- **Postačující podmínka pro existenci RI** – bud' f spojitá funkce na intervalu $\langle a, b \rangle$. Potom existuje její Riemannův integrál na intervalu $\langle a, b \rangle$. Pokud je navíc $(\sigma_n)_{n=1}^{\infty}$ normální

$$\lim_{n \rightarrow \infty} s(\sigma_n, f) \quad \text{a} \quad \lim_{n \rightarrow \infty} S(\sigma_n, f)$$

posloupnost a dělení intervalu $\langle a, b \rangle$ potom limity existují, a jsou rovny Riemannově integrálu funkce f na intervalu $\langle a, b \rangle$.

- **Integrální součet** – pro funkci f spojitou na uzavřeném intervalu $\langle a, b \rangle$ a dělení $\sigma = \{x_0 = a, \dots, x_n = b\}$ tohoto intervalu definujeme **integrální součet funkce f při dělení σ** předpisem

$$\mathcal{J}(\sigma, f) = \sum_{i=1}^n f(\alpha_i) \Delta_i,$$

kde $\alpha = (\alpha_1, \dots, \alpha_n)$ a α_i patří do intervalu $\langle x_{i-1}, x_i \rangle$, $i = 1, 2, \dots, n$

- Riemannův integrál funkce f spojité na intervalu (a, b) lze tedy počítat i jako limitu z integrálních součtů

- **Vlastnosti RI**

- **Aditivita integrálu** – nechť f a g jsou spojité funkce na intervalu $\langle a, b \rangle$. Potom pro RI funkce $f + g$ (která je také automaticky spojitá na $\langle a, b \rangle$) platí

$$\int_a^b (f + g)(x) dx = \int_a^b f(x) dx + \int_a^b g(x) dx$$

- **Multiplikativita integrálu** – nechť f je spojitá funkce na intervalu $\langle a, b \rangle$ a $c \in \mathbb{R}$ je konstanta. Potom pro RI funkce cf platí

$$\int_a^b (cf)(x) dx = c \int_a^b f(x) dx$$

- **Aditivita integrálu v mezích** – Riemannův integrál funkce f na intervalu $\langle a, b \rangle$ existuje, právě když pro každé $c \in (a, b)$ existují Riemannovy integrály funkce f na intervalech $\langle a, c \rangle$ a $\langle c, b \rangle$. V takovém případě navíc platí

$$\int_a^b f(x) dx = \int_a^c f(x) dx + \int_c^b f(x) dx.$$

- **Nerovnosti mezi integrály** – nechť jsou f a g spojité funkce na intervalu $\langle a, b \rangle$ a nechť platí nerovnost $f(x) \leq g(x)$ pro všechna $x \in \langle a, b \rangle$. Potom pro jejich Riemannovy integrály platí

$$\int_a^b f(x) dx \leq \int_a^b g(x) dx$$

- **Newtonova formule** – nechť f je funkce spojitá na intervalu (a, b) s primitivní funkcí F . Pak platí rovnost

$$\int_a^b f(x) dx = F(b) - F(a) =: [F(x)]_a^b$$

- **Per partes pro určitý integrál** – Nechť f a g jsou funkce spojité na (a, b) , f má spojitou derivaci na intervalu (a, b) a nechť G je primitivní funkce k funkci g na intervalu (a, b) . Potom

$$\int_a^b f(x)g(x) dx = [f(x)G(x)]_a^b - \int_a^b f'(x)G(x)dx.$$

- **Substituce pro určitý integrál** – Nechť pro funkce f a φ platí

- φ a její derivace φ' jsou spojité na (a, b) ,
- f je spojitá na $\varphi((\alpha, \beta))$

$$\int_{\alpha}^{\beta} f(\varphi(t)) \cdot \varphi'(t) dt = \int_{\varphi(\alpha)}^{\varphi(\beta)} f(x) dx$$

Potom

- **Zobecněný Riemannův integrál** – Nechť f je funkce definovaná na intervalu (a, b) pro nějaké $a \in \mathbb{R}$ a $b \in (a, +\infty)$, která je Riemannovsky integrabilní na intervalu (a, c) pro každé $c \in (a, b)$. Pokud existuje konečná limita pak její hodnotu značíme

$$\lim_{c \rightarrow b^-} \int_a^c f(x) dx$$

nazýváme **zobecněným Riemannovým integrálem** funkce f na intervalu (a, b) a říkáme, že integrál

$$\int_1^{+\infty} \frac{1}{x^2} dx = \lim_{c \rightarrow +\infty} \int_1^c \frac{1}{x^2} dx = \lim_{c \rightarrow +\infty} \left(-\frac{1}{c} + \frac{1}{1} \right) = 1$$

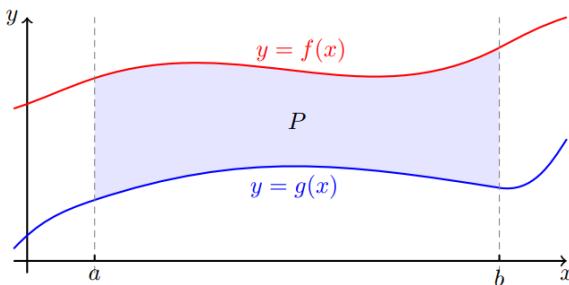
$\int_a^b f(x) dx$ konverguje.

- **Geometrický význam integrálu** – výpočet plochy pod grafem

- **Výpočet plošných útvarů** – Nechť f a g jsou funkce spojité na (a, b) takové, že $f(x) \geq g(x)$ pro každé $x \in (a, b)$. Pak obsah plochy P ohraničené přímkami $x = a$ a $x = b$ a grafy funkcí f a g je rovna

$$P = \int_a^b (f(x) - g(x)) dx.$$

- Pokud je $g(x) = 0$, tak se počítá plocha nad grafem



Číselné řady (konvergence číselné řady, kritéria konvergence, odhadování rychlosti růstu řad pomocí určitého integrálu)

BI-ZMA

- **Reálná číselná posloupnost** – zobrazení množiny \mathbb{N} do množiny \mathbb{R}
 - o Skutečnost, že $a: \mathbb{N} \rightarrow \mathbb{R}$ je posloupnost, zapisujeme také zkráceně symbolem $(a_n)_{n=1}^{\infty}$
 - o **Limita posloupnosti** – řekneme, že reálná posloupnost $(a_n)_{n=1}^{\infty}$ má limitu $\alpha \in \overline{\mathbb{R}}$, právě když pro každé okolí H_α bodu α lze nalézt $n_0 \in \mathbb{N}$ takové, že pro všechna $n \in \mathbb{N}$ větší než n_0 platí $a_n \in H_\alpha$. V symbolech

$$(\forall H_\alpha)(\exists n_0 \in \mathbb{N})(\forall n \in \mathbb{N})(n > n_0 \Rightarrow a_n \in H_\alpha)$$

Tuto skutečnost můžeme napsat několika možnými způsoby:

$$\lim_{n \rightarrow \infty} a_n = \alpha, \lim a_n = \alpha, a_n \rightarrow \alpha$$

- o **Konvergence** – posloupnost $(a_n)_{n=1}^{\infty}$ se nazývá **konvergentní**, právě když její limita existuje a je prvkem \mathbb{R} . V ostatních případech ji nazýváme **divergentní**.

- **Číselná řada** – Formální výraz tvaru

$$\sum_{k=0}^{\infty} a_k = a_0 + a_1 + a_2 + \dots$$

nazýváme **číselnou řadou**. Pokud je posloupnost **částečných součtů**

$$s_n := \sum_{k=0}^n a_k, \quad n \in \mathbb{N}_0$$

konvergentní, nazýváme příslušnou řadou také **konvergentní**. V opačném případě mluvíme o **divergentní** číselné řadě. Součtem konvergentní řady $\sum_{k=0}^{\infty} a_k$ nazýváme hodnotu limity $\lim_{n \rightarrow \infty} s_n$.

- **Kritéria konvergence**

- o **Nutná podmínka konvergence** – pokud řada $\sum_{k=0}^{\infty} a_k$ konverguje, potom pro limitu sčítanců platí $\lim_{k \rightarrow \infty} a_k = 0$.
- o **Bolzano-Cauchy kritérium** – řada $\sum_{k=0}^{\infty} a_k$ konverguje právě tehdy, když pro každé $\varepsilon > 0$ existuje $n_0 \in \mathbb{N}$ tak, že pro každé $n \geq n_0$ a $p \in \mathbb{N}$ platí
 - Jedná se pouze o použití Bolzanova–Cauchyova kritéria konvergence na posloupnost částečných součtů příslušné řady a přeznačení některých symbolů.
 - Posloupnost (a_n) je konvergentní, právě když pro každé $\varepsilon > 0$ existuje $n_0 \in \mathbb{N}$ takové, že pro každé $n, m > n_0$ je $|a_n - a_m| < \varepsilon$
 - U řad si můžeme zvolit jakoukoliv konstantu a nalézt takový nekonečný konec posloupnosti, který bude menší než konstanta. Tzn. čím menší konstantu volím, tím dál od počátku začíná nekonečný konec posloupnosti částečných součtů.
- o **Absolutní kritérium** – řadu $\sum_{k=0}^{\infty} a_k$ nazýváme **absolutně konvergentní**, pokud řada $\sum_{k=0}^{\infty} |a_k|$ konverguje
 - Pokud řada absolutně konverguje, potom konverguje.
 - Vychází z B-C a trojúhelníkové nerovnosti.
- o **Leibnizovo kritérium – buď** $(a_k)_{k=1}^{\infty}$ monotónní posloupnost konvergující k nule. Potom řada
$$\sum_{k=0}^{\infty} (-1)^k a_k$$
konverguje.
 - Postačující podmínka konvergence
- o **Srovnávací kritérium** – buďte $\sum_{k=1}^{\infty} a_k$ a $\sum_{k=1}^{\infty} b_k$ číselné řady. Potom platí následující dvě tvrzení:
 1. Nechť existuje $k_0 \in \mathbb{N}$ takové, že pro každé $k \in \mathbb{N}$ větší, než k_0 platí odhad $0 \leq |a_k| \leq b_k$ a nechť řada $\sum_{k=1}^{\infty} b_k$ konverguje. Potom řada $\sum_{k=1}^{\infty} a_k$ absolutně konverguje.

2. Nechť existuje $k_0 \in \mathbb{N}$ takové, že pro každé $k \in \mathbb{N}$ větší, než k_0 platí odhad $0 \leq a_k \leq b_k$ a nechť řada $\sum_{k=1}^{\infty} a_k$ diverguje. Potom i řada $\sum_{k=1}^{\infty} b_k$ diverguje.
 - Opět postačující podmínka konvergence
 - Vychází z B-C
 - Podobné limitě o sevřené posloupnosti
- **d'Alembertovo kritérium** – Nechť $a_k > 0$ pro každé $k \in \mathbb{N}$. Potom platí dvě následující tvrzení:
 1. Pokud

$$\lim_{k \rightarrow \infty} \frac{a_{k+1}}{a_k} > 1.$$
 pak řada $\sum_{k=0}^{\infty} a_k$ diverguje.
 2. Pokud

$$\lim_{k \rightarrow \infty} \frac{a_{k+1}}{a_k} < 1,$$
 pak řada $\sum_{k=0}^{\infty} a_k$ (absolutně) konverguje.
 - Máme všechny členy posloupnosti kladné. Limitní podíl následujícího a předchozího členu musí být větší než 1, pak řada diverguje. V případě menší než 1 konverguje.
 - Opět postačující podmínka konvergence
- **Integrální kritérium** – buď $\sum_{n=1}^{\infty} a_n$ číselná řada s kladnými členy. Nechť existuje spojitá a monotónní funkce definovaná na $(0, +\infty)$ taková, že $f(n) = a_n$ pro každé n . Potom:
 - Pokud integrál $\int_1^{\infty} f(x) dx$ konverguje, pak číselná řada $\sum_{n=1}^{\infty} a_n$ konverguje.
 - Pokud integrál $\int_1^{\infty} f(x) dx$ diverguje, pak číselná řada $\sum_{n=1}^{\infty} a_n$ diverguje.
- **Odhadování rychlosti růstu řad pomocí určitého integrálu** – Nechť f je spojitá funkce na $(1, +\infty)$ a $n \in \mathbb{N}$.
 1. Je-li f klesající, pak

$$f(n) + \int_1^n f(x) dx \leq \sum_{k=1}^n f(k) \leq f(1) + \int_1^n f(x) dx$$
 2. Je-li f rostoucí, pak

$$f(1) + \int_1^n f(x) dx \leq \sum_{k=1}^n f(k) \leq f(n) + \int_1^n f(x) dx$$

- Komplikovaná a často neřešitelná úloha
- Přesný součet nás ani nezajímá, jde nám pouze o typické chování pro velká n . Tedy o tzv. asymptotické chování součtu.
- **Geometrický odhad růstu** – příklad:

Pomocí odhadu odhadněte rychlosť růstu posloupnosti

$$a_n = \sum_{k=1}^n k^2, \quad n \in \mathbb{N}.$$

Nyní $f(x) = x^2$ je rostoucí na $(1, +\infty)$ a proto pro každé $n \in \mathbb{N}$ platí

$$1 + \int_1^n x^2 dx \leq \sum_{k=1}^n k^2 \leq n^2 + \int_1^n x^2 dx.$$

Tudíž

$$\frac{1}{3}n^3 + \frac{2}{3} \leq a_n \leq \frac{1}{3}n^3 + n^2 - \frac{1}{3}.$$

Pro velká n je největším členem $\frac{1}{3}n^3$, přesněji

$$\lim_{n \rightarrow \infty} \frac{a_n}{\frac{1}{3}n^3} = 1, \quad \text{tj. } a_n \sim \frac{1}{3}n^3.$$

