# Algoritmizace

## Osnova

- algoritmus
    - teoretický princip řešení problému
- vlastnosti algoritmů
- druhy algoritmů
- algoritmická složitost
    - big O
- známé algoritmy

# Algoritmus

- teoretický princip řešení problému
- nejde o konkrétní implementaci, ale o návrh
- mimo programování: recept – postup

# Vlastnosti algoritmů

- co musí splňovat, aby se jednalo o algoritmus
- **elementárnost**
  - skládá se z daného počtu jednoduchých kroků
- **konečnost**
  - počet kroků musí být konečný
  - algoritmus musí mít konec
    - pro každý vstup
- **obecnost**
  - neřešíme jeden konkrétní problém (1+2)
  - řeší obecnou třídu obdobných problémů (jak spočítat součet dvou čísel)
- **determinovanost**
  - každý krok musí být jednoznačně a přesně definován
    - musí být také jasno co se bude dít v dalším kroku
  - přirozené jazyky toto nesplňují (lidské jazyky jako Češtiny) → vznik programovacích jazyků
    - každé slovo má přesný význam
- **výstup**
  - má alespoň jeden výstup

# Druhy algoritmů

- **rekurzivní** (wiki rekurze)
  - opakovaně využívají sami sebe
  - funkce, která volá sama sebe
  - příklad: faktoriál
    - ```
      FUNCTION factorial(n) {
        IF (n == 1) return n;
        ELSE return n * factorial(n-1);
      }
      ```
- **pravděpodobnostní**
  - provádí některá rozhodnutí náhodně
  - wiki
- **genetický**
  - napodobují biologické procesy – evoluční
  - game of life
- **heuristické**
  - za cíl si neklade nalézt přesné řešení, ale pouze nějaké vhodné přiblížení
      ```

# Algoritmická složitost

- aby algoritmus skončil v rozumném čase
- jeden problém má více řešení
    - více algoritmů
        - nejjednodušší – nejméně komplexní a časově náročný je většinou nejlepší
- příklad: robot který hraje šachy
    - mohli bychom jednoduše pro každou herní pozici simulovat hru a dostat se k nejlepšímu tahu, který by měl robot zahrát
        - extrémní počet možných kombinací tahů nelze výpočetně zpracovat i když teoreticky by to bylo vcelku jednoduché
            - šílené množství potřebného výpočetního výkonu
            - časově náročné
            - nemožné množství dat pro uložení
    - využívá se jiný algoritmus, který je sice složitější na pochopení, ale je efektivní a méně komplexní

- zápis komplexnosti – „big O“ notace (od velkého O)
    - zápis závislosti výpočetního času na rozsahu úloh
    - O(n) – počet kroků závisí lineárně na počtu vstupních dat
        - n vstupních dat … stejný počet kroků
    - O(log n) – logaritmická závislost
        - n vstupních dat … log n kroků
    - O(1)
        - nezávisí na množství dat, vždy jeden krok
    - **bonus**: zároveň takto můžeme udělat komplexnost datovou (jak moc velký prostor v operační paměti zabere program)
        - rekurzivní faktoriál
            - časová: O(n)
            - datová: O(n)
        - faktoriál pomocí for loopu
            - časová: O(n)
            - datová: O(1)

```
FUNCTION sumRecursive(n)
   IF n == 0 THEN
      RETURN 0
   ELSE
      RETURN n + sumRecursive(n - 1)
   END IF
END FUNCTION

FUNCTION sumFormula(n)
   RETURN n * (n + 1) / 2
END FUNCTION
```

- sumRecursive – rekurzivní funkce
    - časová náročnost O(n)
    - musí udělat n počet kroků
    - popis: zadáme číslo n, funkce potom volá sama sebe s n-1 dokud se nedostane do nuly
    - proces
        - n = 4;
        - **sumRecursive**(4);
            - return 4 + **sumRecursive**(3);
                - return 3 + **sumRecursive**(2);
                    - return 2 + **sumRecursive**(1);
                        - return 1 + **sumRecursive**(0);
                            - return 0;
        - výstup: 4 + 3 + 2 + 1 + 0
        - 4 rekurzivní volání – 4 kroky
- sumFormula – rovnice
    - časová náročnost O(1)
    - musí udělat pouze jeden krok
    - popis: zadáme číslo n, funkce udělá matematickou operaci a vrací odpověď
    - proces
        - n = 4;
        - **sumFormula**(4);
            - return 4 * (4 + 1) / 2;
        - výstup: 4 * (4 + 1) / 2
- lepší je teda udělat sumu pomocí rovnice: rychlejší
- podobná úlohy by byla v pracovním listě a k tomu jedna složitější na popis

# Známé algoritmy

- Dijkstrův algoritmus
    - hledání nejkratší cesty v grafu
    - využití: navigace
- řadící algoritmy
    - bubble sort
        - postupně prochází prvky po dvou a když je jeden z nich menší, tak prohazuje
        - [4, 1, 8, 5, 6]
            - (4, 1) → (1, 4), (4, 8) nic, (8, 5) → (5, 8), (8, 6) → (6, 8)
            - znovu na začátek pro kontrolu
            - (1, 4) nic, (4, 5) nic, (5, 6) nic, (6, 8) nic
            - hotovo, pokud by ale i tady došlo k prohození, tak by se to opakovalo
        - vysoká časová náročnost pro n prvků O(n^2)
    - heapsort
    - quicksort

# Zdroje


- OTÁHAL, Ing Tomáš. Algoritmizace- úvod.
- Algoritmus. In: *Wikipedie* [online]. 2026 [cit. 08.03.2026]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Algoritmus&oldid=25576234
- Řadicí algoritmus. In: *Wikipedie* [online]. 2025 [cit. 08.03.2026]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=%C5%98adic%C3%AD_algoritmus&oldid=25073222