# Algoritmizace – přehled

## 1. Algoritmy a jejich efektivita

- Správnost algoritmu
  - **Konečnost** = pro každá přípustná vstupní data obdržíme v konečném čase nějaký výstup
  - **Částečná správnost** = když výpočet skončí, vydá správný výsledek
  - **Algoritmus je spravny = konečny + částečně správny**
- Složitost (efektivita) algoritmu
  - **Časová** = počet vykonanych operací / rychlost vypočtu programu
    - Provedení podmínky, přiřazení, aritmetické operace
  - **Prostorová** = paměť potřebná na uložení dat při vypočtu programu
    - Velikost datové struktury (pole n, matice mxn)
- Analyza složitosti
  - **Nejhorší případ =** maximální délka vypočtu
  - **Nejlepší případ** = minimální délka vypočtu

    Nad vstupem délky n
  - **Průměrny případ** = součet délek vypočtu nad všemi vstupy délky n

## 2. Asymptotická časová složitost

- Asymptotická časová složitost
  - Počítáme jen nejrychleji rostoucí člen (v $3n^2 + 2n – 4$: $O(n^2)$)
  - **Notace**: f,g: N -> R
- Odhady funkce
  - $\exists$ c > 0 $\exists$ n0 > 0 $\forall$ n ≥ n0 platí:
  - **Horní (big O)**: $f \in O(g)$
    - f(n) je třídy **O(g(n))**
    - $0 \leq f(n) \leq c.g(n)$
  - **Dolní**: $f \in \Omega(g)$
    - f(n) je třídy **Ω(g(n))**
    - $0 \leq c \cdot g(n) \leq f(n)$
  - **Přesny**: $f \in \Theta(g)$
    - f(n) je třídy **Θ(g(n))**

        - f ∈ **O(g) & f ∈ Ω(g)**
- Spektrum časové složitosti
    - Polynomiálně omezeny čas
        - O(1), O(log(n)) – Binární vyhledávání, O(n), O(n*log(n)) – MergeSort, HeapSort, O($n^2$) – BubbleSort, InsertSort
        - **Velká** vstupná data
    - Exponenciální čas
        - O(2^n), O(n!)
        - **Malá** vstupná data

# 3. Algoritmy z teorie čísel

- Testování prvočíselnosti
    - **Vstup**: přirozené číslo N > 1
    - **Vystup:** True pokud N je prvočíslo, jinak False
    - **Časová složitost:** exponenciální (vzhledem k délce vstupu: [log2(n)] + 1
    - **Řešení:** stačí prověřit dělitele <=$\sqrt{N}$
    - Nebo **Eratosthenovo síto**
        - Určete všechna prvočísla od 2 do N
        - v řadě čísel od 2 do N postupně vyškrtáváme všechny násobky jednotlivých prvočísel - co nakonec nebude vyškrtnuto, je prvočíslo
        - **složitost:** O(N.log(log N))

- Určení největšího společného dělitele
    - **Vstup**: přirozená čísla x, y > 0
    - **Řešení**: prvočíselny rozklad,
    - Nebo **Euklidův algoritmus**
        - Pro x>y: NSD(x, y) = NSD(x − y,y)
        - **Zrychlení**: NSD(x, y) = NSD(y,x mod y)
            - NSD(27, 21) = NSD(21, 6) = … = NSD(3,3) = 3
        - **Složitost**: O(log(x)) //iterace cyklu

- Vyhodnocení polynomu
    - $a(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} \ldots + a_1 x + a_0$
    - polynom stupně n
    - **Řešení**: přímy vypočet
        - O($n^2$) operací: n + (n-1) + (n-2) + … + 1 = n.(n+1)/2
    - **Hornerovo schéma**:
        - $a(x) = (\ldots((a_n x + a_{n-1}).x + a_{n-2}).x + \ldots + a_1).x + a_0$
        - O(n) operací

- Převody mezi číselnými soustavami
  - $a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b + a_0$
  - převod z binární soustavy (Hornerovo schéma) – nebo naopak

    $10111_2 = ´(((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 1 = 23$

    $23 = (((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 1 = 10111_2$
- Rychlé umocňování
  - **Vstup**: (velké) přirozené číslo N a hodnota X
  - **Vystup**: $X^N$
  - **Řešení**: přímočaře - $X^N = X*X*\ldots X$
    - (n-1) násobení – *exponenciální čas*
  - Nebo **Převod exponentu N do binární soustavy**
    - $13 = (1101)_2 = 2^3 + 2^2 + 2^0$
    - $X^{13} = X^8 \cdot X^4 \cdot X$
    - Složitost: O(log(n))

# 4. Třídění a vyhledávání

## Vyhledávání

- Operace se seznamem
  - **přístup** k prvku v čase O(1)
  - **append(x)** – θ(n) (v nejhorším případě), O(1) (průměrně)
  - **del a[i]** – θ(n-i)
  - **in, index(), count()** – Θ(n)
- **Sekvenční**
  - Vstup: uspořádané pole, prvek x
  - **Výstup**: pokud se x v poli vyskytuje, index výskytu False jinak
  - **Časová složitost:** Θ(n)
- **Binární**
  - Pokud:
    - a[n // 2] == x , jsme hotovi
    - a[n // 2] > x , prohledáme jen prvky a[i] < a[n//2]
    - a[n // 2] < x , prohledáme jen prvky a[i] > a[n//2]
  - **invariant**: a[i] = x => dolni <= x <= horni
  - **Časová složitost**: O(log2n)
  - vypočet odmocniny n – půlením intervalu

# Třídění

Vnitřní třídění

- Složitost: $O(n^2)$ //nested for
- *Porovnávací algoritmy*: porovnávají dvojice, setříděny úsek nepoužívá
- *Rozhodovací strom*: reprezentuje průběh porovnání porovnávacím algoritmem**.** Má alespoň **n! listů**
    - dolní odhad: Maximální počet porovnání nad vstupem délky n je ≥ konstanta · n · $\log_2$ n. Časová složitost: Ω(n log n)

- **BubbleSort**
    - projdi pole a porovnej dvojice sousedních prvků
    - v případě potřeby dvojici vyměň
    - po dosažení konce seznamu začni znovu od začátku
    - po i-té iteraci (i = 1,2,…,n-1) je i posledních prvků na svých místech

- **SelectionSort / Třídění výběrem**
    - (pro i = 0,1,…,n-2) mezi a[i],…,a[n-1] najdi minimální prvek, a vyměň s prvkem a[i]
    - Data malého rozsahu
    - Jen (n-1) vyměn, O(n) zápisů do pole a

- **InsertionSort** / **Třídění vkládáním**
    - pole se dělí na setříděný úsek (vlevo) a nesetříděný úsek (vpravo). na začátku je setříděný úsek tvořen pouze prvním prvkem pole
    - první prvek nesetříděného úseku se vždy zařadí do setříděného úseku na místo, kam patří, tím se setříděný úsek prodlouží o jeden prvek
    - **nejlepší časová složitost**: O(n) //pole už je setříděné

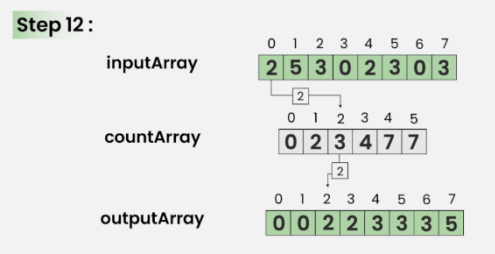
- **HeapSort**
    - **Datová struktura**: binární halda
    - Halda = bin. strom
    - **Operace**: Přidej (prvek), OdeberMin/Max – O(log(n))
    - hodnoty uložené ve vrcholech splňují podmínku *haldového uspořádání*:
        - **max-halda:** hodnota kořene (pod)stromu >= hodnota libovolného dítěte
        - **min-halda:** hodnota kořene (pod)stromu <= hodnota libovolného dítěte
    - **vyška:** h <= log2n

- **haldové třídění**
  - Z prvků pole **a** vybuduj haldu – vkládej prvky od a[0] do a[n-1], //n … počet prvků v poli
  - Pokud **min-halda**: z haldy postupně odebírej minima (minimum – v kořeni)
  - Pokud **max-halda**: z haldy postupně odebírej maxima (maximum – v kořeni)
  - **Můžeme třídit pomocí pomocného pole/na místě**
  - **Složitost:** O(nlog(n))

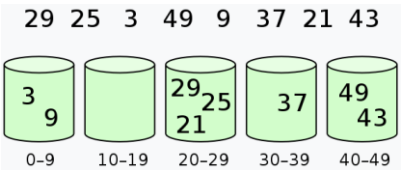
Třídění v lineárním čase / Přihrádkové metody

- **Složitost**: Lineární O(n+k)

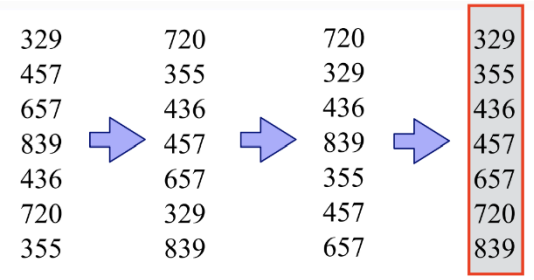
- **CountingSort / Třídění počítáním**
  - pro každé x ∈{0,1,…, k} spočítej # výskytů x na vstupu
  - zapiš x na správnou pozici v setříděné posloupnosti na výstup
  - pomocná datová struktura: c[0…k] #pole s počty vyskytů hodnot, indexované od 0 až do k
  - Časová složitost: O(n + k) (lineární)
    - N – počet indexů v posloupnosti
    - K – počet indexů v pomocném poli
  - Nepraktické pro **velké k**




- **BucketSort / Přihrádkové třídění**
  - Modifikace třídění počítáním, která umožňuje třídit strukturovaná data (záznamy s klíčem)
  - Vstup: pole a s prvky, jejichž klíče leží v {0,1,…, k}
  - Vystup**:** pole b obsahující prvky pole a uspořádané vzestupně dle klíče
  - Pomocná datová struktura: c[0..k] – kumulované četnosti
    - **c[i] < i** v poli a
  - **O(n + k)**




- **RadixSort / Víceprůchodové přihrádkové třídění**
  - setřídí pole podle číslic v klíči
  - Klíč je d-tice
    - *Lexikografické uspořádání*: ∃ k ∈ {0,1, …, d-1} tak, že • že xi = yi pro i = 0,1,…,k – 1 a xk < yk .
  - **Invariant**: Po i-tém průchodu jsou prvky uspořádány dle posledních **i** souřadnic klíče.
  - **Časová složitost:** O(d(n +k)) – pokud stabilně třídí v čase O(n+k)
    - Stabilní třídění = zachová se pořadí záznamu (byly a zůstanou na svém místě)
    - Pokud d=O(1) a k = O(n) => složitost O(n)




Třídění – rozděl a panuj

- = Z řešení podproblémů sestroj řešení původního problému

- **MergeSort/Třídění sléváním**
  - Pole délky n ≤ 1 je již setříděno
  - Jinak pole délky n ≥ 2 rozděl na poloviny, setřiď obě části, potom je slej do jednoho setříděného pole
  - Slévání – **průchod stromem rekurze do hloubky**
  - **Časová složitost:** T(n) = log2n (vyska stromu) * O(n) = O(n*log(n))
  - **Prostorová složitost:** S(n) = O(n)
    - Pracovní paměť: Pomocné pole pro slévání

- **QuickSort/Třídění rozdělováním**
  - pole délky ≤ 1 je již setříděno
  - jinak ve vstupním poli zvolíme prvek = pivot, prvky v poli přeskupíme tak, aby výsledné pole tvořily dva navazující úseky:
    - levý s prvky ≤ pivot
    - pravý s prvky ≥ pivot
  - úseky setřídíme rekurzivně
  - netřídí na místě
  - **časová složitost**: T(n) = O(n) + max1≤ i≤n-1(T(i)+T(n-i)) = O($n^2$) #v nejhorším případě
    - **v nejlepším případě: O(n logn)**

# 5. Datové struktury

- Abstraktní datovy typ

- Množina prvků, nad kterou můžeme provádět operace. Určují pořadí prvků
- Příklad: Zásobník, fronta, prioritní fronta, lineárně spojovy seznam atd.

- Zásobník
  - Dno zásobníku (0. index v poli, je na začátku seznamu, odebírá se poslední index)
  - Vždy se odebírá **nejmladší prvek** (a[-1])
  - **Operace**: push – S(n) = O(n) (nejhorší), pop – S(n) = O(1) (odeber poslední prvek)
  - **Použití**: Rekurze, prohledávání do hloubky (DFS)

- Fronta
  - Odebírá se ze začátku (nejstarší prvek), přidává se na konec (nejmladší prvek)
  - **Operace**: enqueue (přidej) – S(n) = O(1), dequeue (odeber) – S(n) = O(n) (posunou se všechny zbyvající prvky doleva)
  - Cyklická fronta = neposouváme prvky v poli, po posledním prvku ve frontě následuje první prvek fronty
  - Oboustranná fronta (**deque**) = lze přidávat a odebírat prvky zprava i zleva (v O(1))
  - **Použití:** Prohledávání do šířky (BFS)

- Lineární spojovy seznam (LSS)
  - Posloupnost prvku, které jsou lineárně provázány odkazy pomocí ukazatelů
  - Každy prvek obsahuje svou hodnotu a odkaz na následující prvek v seznamu
  - Poslední prvek se odkazuje na None
    - V cyklickém LSS se však odkazuje na první prvek

- Prioritní fronta
  - Fronta, v níž každy prvek má svou prioritu
    - Odebírá se vždy prvek s maximální prioritou (je-li jich více, odebereme libovolny)

  - Implementace LSS:
    - Nesetříděny seznam: append – O(1), pop – O(n) (hledáme prvek s max prioritou)
    - Setříděny seznam: append – O(n) (prvek vkládáme – pomocí InsertSort), pop – O(1) (z hlavy)

  - Implementace binární halda: max-halda (odeber max prvek z haldy)

        - V čase O(log(n))

- Binární halda (=strom)
    - V každé hladině (od první do předposlední) je $2^i$ vrcholu
    - Poslední hladina se zaplňuje zleva
    - Viz HeapSort (4. Třídění)

- ADT Slovník
    - Reprezentuje dynamickou množinu s operacemi
        - Vyhledej(klič v slovníku)
        - Ulož(hodnotu v slovníku)
        - Vymaž(klíč ze slovníku)
    - Hashovací tabulky, vyhledávácí stromy

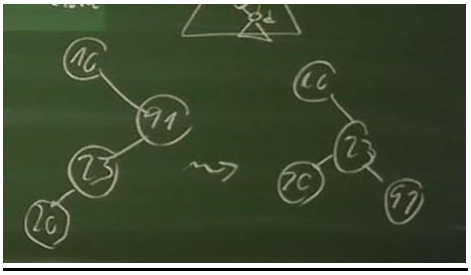
- Binární vyhledávácí strom
    - Uspořádání vrcholů: v levém podstromě s menšími klíči, v pravém s většími klíči, než u klíče kořene (podstromu)
    - Operace **Vyhledej:**
        - pokud hledany klíč < klíč vrcholu, prohledáme levy podstromu. Pokud > klíč vrcholu, prohledáme pravy podstrom. Jinak pokud hledany klíč = klíč vrcholu, vrátíme vrchol
    - Operace **Ulož**:
        - Přidej novy list (s klíčem)

    - Operace **Vymaž**:
        - Pomocí operace Vyhledej najdi vrchol **v** s klíčem k
        - Pokud (v = **list**) => změnit v rodiči odkaz na (v -> None)
        - Pokud v má **jediné dítě** d => změnit v rodiči odkaz na v -> odkaz na d
        - Pokud v má **2 děti** => jeho bezprostřední následovník **s** má nejmenší hodnotu v nejlevějším listu pravém podstromu v. Vymažeme vrchol s a jim nahradíme vrchol v. **Alternativa**: nejpravější hodnota v levém podstromě

    - časová složitost: O(vyška(BVS))
        - **minimální** vyška: log2n (bin. halda)
        - **maximální** výška: n – 1 (degenerovany strom = cesta, kde všechny vrcholy kromě posledního mají jednoho dítěte)
        - **Vymaž, ulož, přidej** –v nejhorším případě theta(n)

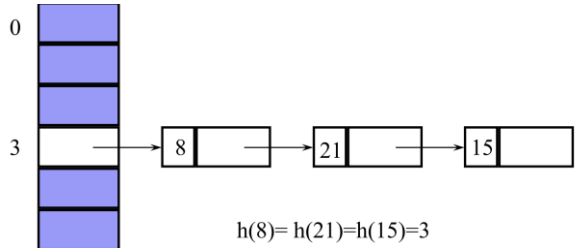
    - Dokonale vyváženy BVS = počet vrcholů v levém / pravém podstromě se liší nejvyše o 1
        - Vyška: log2n
        - Ulož, vymaž – mohou zničit vyvaženost

- AVL (Adelson-Velskiy and Landis) strom
  - **výškově vyváženy BVS**: vyška levého a pravého podstromu se liší o <= 1
  - **porušení vyváženosti**: řešení - pomocí rotací



- Hašovací tabulka
  - datová struktura, která implementuje ADT Slovník
  - lze provést operace (Ulož, Vyhledej, Vymaž) v **čase O(1)**
  - **Motivace - Tabulka s přímym přístupem:**
    - Klíče leží v univerzu U={0,1,…,m-1}, m je malé
    - žádné dva prvky nemají stejný klíč
    - pole s m prvky, která mají klíč z univerza: t[k]

  - **Hlavní idea**
    - hašovací funkce h: U →{0,1,…,m-1}, m << |U|
      - **není prostá**: h(klíč(x)) = h(klíč(y)) (**problém - kolize**)
    - hašovací tabulka: pole t[0..m-1]
      - je-li **h(klíč(x)) = i**, pak **t[i] = x**
    - složitost:
      - S(n) = Theta(|K|)
      - T(n) = O(1) (vypočet h(k))
        - **Vyhledej** – O(1 + (n/m)), kde m = velikost tabulky, n = počet uloženych prvků
    - Řešení kolizí:
      - **separátní řetězení** = klíče z U odkazují na stejny index v seznamu => t[i] obsahuje odkaz na jeho hlavu
        - také funguje s operacemi Ulož, Vymaž, Vyhledej




      - **otevřená adresace**

- všechny prvky jsou uloženy přímo v tabulce
- h: U ×{0,1, …,m-1} → {0,1, …,m-1}
- pak: **h(k,0), h(k,1),…, h(k,m-1)**

# 6. Rekurze, rekurzivní datové struktury

## Rekurze

- Způsob řešení úlohy pomocí podúloh
- Implementace:
  - **Rekurzivní funkcí**: Volá sama sebe, musí mít *podmínku ukončení* (base case)
    - **Bez podmínky**: RecursionError (zacyklení vypočtu)
  - **Zásobníkem**
- Chytrá rekurze – **memoizace / kešování / dynamické programování**
  - Zrychlí rekurzivní funkci
- **Rekurzivní generování** = generování / zkoušení všech možností
  - Generování variací, kombinací, atd. Rozklad čísla na součet sčítanců
- **Příklad**: faktoriál, Fibonacciho čísla, Hanojská věž, Rozklad čísla

## Rekurzivní datové struktury

- Binární strom = uspořádány, každy vrchol má nejvyše 2 děti
- **Průchod** binárním stromem:
  - **Preorder**: vyhodnocení kořenu => rekurzivně projdi levy, pak pravy podstrom
  - **Inorder**: rekurze - levy podstrom => vyhodnocení kořenu => rekurze - pravy podstrom
  - **Postorder**: rekurze - levy podstrom, pak pravy podstrom => vyhodnocení kořenu
- **Časová složitost**: O(n)
- **Aritmetické vyrazy**: listy = čísla (operandy), vnitřní vrcholy = operátory
- Notace aritmetických vyrazů
  - I**nfixová**: (1 + 2) * 3
  - **Prefixová**: * + 1 2 3

    - **Postfixová**: 1 2 + 3 *
- **Obecné stromy (k-ární stromy)**
    - Uspořádány strom, kde každy vrchol má nejvýše k děti
    - Reprezentace stromu: místo levy, pravy, má atributy dite1, dite2 … diteK
    - Reprezentace - Spojovy seznam dětí**:** každy vrchol má odkaz na nejlevější (nejstarší) dítě, i na *nejmladšího sourozence* (bezprostředně vpravo). Pokud neexistují, tak jsou obě None

- Průchod do hloubky (DFS – Depth First Search):
    - **Rekurzivně**: preorder, inorder, postorder
    - Nebo pomocí **zásobníku**
    - **Použití:** hry – piškvorky, šachy, sudoku atd.

- Průchod do šířky (BFS – Breadth First Search):
    - **Algoritmus vlny**
    - Používáme **frontu** – vrcholy navštěvujeme po hladinách
    - **Použití**: hledání nejkratší cesty

# 7. Prohledávání stavového prostoru

- Backtracking (prohledávání s návratem)
    - postupně generujeme **částečná řešení úlohy**
    - každé částečné řešení se snažíme (různými způsoby) rozšířit, s cílem získat úplné řešení
    - pokud o některém kandidátovi zjistíme, že nepůjde rozšířit na úplné řešení => **vracíme se zpět** a zkoušíme rozšířit jiného kandidáta (např. průchod bludištěm)

    - slepá cesta: postupujeme kupředu tak dlouho, dokud to jde
    - cyklus: dokud dojdeme na místo, kde jsme už byli
    - backtrack: vrať se na rozcestí a vyber jinou cestu

    - **Průchod do hloubky**
        - Počátečny stav = **kořen**
        - Neúplné řešení, které lze rozšířit = **list**
        - Úplné řešení / Koncovy stav = **list**

    - Složitost:
        - **Časová**: O(n!)
        - Jen pro male vstupy
        - **Prostorová**: O(n) (velikost zásobníku)

- **Příklad**: Problém n dám
    - Jak rozmístit figurky tak, aby se neohrožovaly

- Ořezávání
    - **Odstranění slepych větví** (řešení, které nelze rozšířit na úplné, odřízneme ze stromu)
    - **Problém n dám**: dámu umístíme jen na ty políčka, kde jiné dámy neohrožuje
- Heuristika
    - Metoda řešení problému, která v **krátkém čase** nalezne řešení
    - **Nezaručuje** příznivou časovou složitost
    - **Heuristická funkce** ohodnotí děti zadaného rodiče podle jejich šance na to, že povedou k řešení úlohy
        - Určí pořadí dětí podle počtu dalších pokračování
    - Např. cesta figurkou po šachovnici – hledání nejkratší cesty

- Spojení heuristik a ořezávání
    - Vhodná heuristika (aby se co nejdříve našlo dobré řešení) => zvyší se účinnost ořezávání
    - Např. **Problém obchodního cestujícího**

- Minimaxovy algoritmus
    - Strom hry:
        - Každy vrchol obsahuje hodnotu {-1, 0, 1}
        - Hrana = tah
        - Bíly = “max”, černy = “min”
        - **Listy** = 1 (vyhrál Max), -1 (vyhrál Min), 0 (remíza)
        - **Vnitřní vrcholy**: ohodnocení max, min – z hodnot synů a podle toho, kdo je na tahu

    - Vítězná strategie: Všechny listy reprezentují pozice, v níž jeden z hráčů zvítězí
    - Neprohrávací strategie**:** listy = pozice, kde dojde ke vítězství (jednoho z hráčů), nebo k remíze
    - Průchod do hloubky: **rekurzivně**
        - **Omezená hloubka**
        - Koncová pozice: +∞ (vyhraje Max), -∞ (Min), 0 (remíza)
        - Nekoncová pozice: → +∞ (vyhodnější pro Max), → -∞ (pro Min)
    - **Alpha-beta prořezávání**
        - **A** – nejlepší volba hodnoty pro Max na této cestě, **B** – pro Min
        - Pokud vygenerovany vrchol není v intervalu (a, b) => Odřízneme podstrom

- Nejhorší časová složitost: $O(b^h)$
  - **b** – počet dětí
  - **h** – počet tahů

# 8. Grafové algoritmy

- Neorientovany graf = hrany jsou **2-prvkové podmnožiny vrcholů**
- Orientovany graf = hrany jsou **uspořádané dvojice vrcholů**
- Multigraf = mezi 2 vrcholy je **více hran** (jsou rovnoběžné)
- Hypergraf = (hyper)hrana může spojovat **více než 2 vrcholy**

- Základní pojmy (viz Diskrétka)
  - **Tah** = mohou se opakovat vrcholy
  - **Sled** = mohou se opakovat vrcholy i hrany
  - **Cyklus** = mohou se opakovat vrcholy
  - **Vzdálenost vrcholů u,v** = délka nejkratší cesty mezi u,v
  - **Graf je souvisly** = pokud mezi každou dvojicí vrcholů existuje cesta

- Základní pojmy – Stromy
  - **Strom** = souvisly acyklicky graf, deg(list) = 1
  - **Les** = acyklický graf, komponenty = stromy
  - **Předchůdce - následník (relace)**
    - u,v – vrcholy v kořenovém stromě. Pokud u leží nad v na cestě z kořene do listu, **u** = předchůdce v, **v** = následník u

- Reprezentace grafu
  - **Matice sousednosti**
    - Čtvercová matice řádu n
    - Symetrická, pokud neorientovany graf
    - Prostorová složitost: $\text{theta}(n^2)$
    - Dobré pro transformace grafů
  - **Pole seznamů sousedů**
    - Pole sousede[0…n-1]
    - Kde index = **číslo vrcholu u**, na každém indexu je pole sousedních vrcholů (které jsou z množiny V)
    - Prostorová složitost: $O(n + m)$

- Graf průchodu do hloubky (DFS)
  - Obsahuje všechny vrcholy
  - Ale pouze ty hrany, které vedou do nenavštívenych vrcholů
  - Detekce zpětné hrany

        - Neorientovany graf: O(n)
        - Orientovany graf:
            - Hrany - **Stromové**, **dopředné** (předchůdce -> následník), **zpětné** (naopak), **příčné** (vrcholy nejsou v relaci předchůdce = následník)
            - Existence zpětné hrany: O(n + m)
    - **Určení komponent**
        - **Komponenty[0…n-1]**
        - **Komponenty[u]** = číslo komponenty, do níž patří vrchol u
        - Komponenty očíslujeme od 1…k
        - Časová složitost: **O(n + m)**

- Graf průchodu do šířky (BFS)

    - Časová složitost: **O(n + m)** (pro orientované a neorientované grafy)
    - Hledání **nejkratší cesty** mezi 2 vrcholy

# 9. Zrychlení převypočtem

- Cíl: zlepšit časovou složitost
    - Ale zaplatíme přitom vyšší prostorovou složitosti
- Příklady:
    - Nejčastější číslo:
        - Hrubá síla: spočítat vyskyty všech čísel
        - Předvypočet: na jeden průchod najít nejdelší úsek stejnych hodnot
    -