# Přehled

## Definice algoritmu

* Výpočetní model RAM
* Čas a prostor konkrétního výpočtu
* Časová a prostorová složitost
* Asymptotická notace: O, Ω, Θ

## Základní grafové algoritmy

* Prohledávání do šířky (BFS)
* Prohledávání do hloubky (DFS)
* Klasifikace hran v DFS
* Hledání mostů
* Algoritmy pro orientované grafy
* Detekce cyklů pomocí DFS
* Acyklický orientovaný graf (DAG), zdroj, stok
* Topologické uspořádání DAGu
* Konstrukce topologického uspořádání
* Princip indukce podle topologického uspořádání
* Počet cest mezi dvěma vrcholy v DAGu
* Silná souvislost, její komponenty, graf komponent
* Rozklad grafu na komponenty silné souvislosti

### Nejkratší cesty

* Vzdálenost v grafu
* Trojúhelníková nerovnost pro vzdálenost
* Dijkstrův algoritmus
* Implementace Dijkstrova algoritmu pomocí haldy
* Obecný relaxační algoritmus
* Bellmanův-Fordův algoritmus

### Minimální kostry

* Jarníkův algoritmus
* Lemma o řezech
* Jednoznačnost minimální kostry
* Borůvkův algoritmus
* Kruskalův algoritmus
* Union-Find

## Vyhledávací stromy

* Rozhraní slovníku, množiny a jejich uspořádaných verzí
* Binární vyhledávací strom (BVS)
* Operace Find, Insert a Delete v BVS
* Dokonale vyvážený strom
* Dolní odhad složitosti pro dokonale vyvážené stromy
* AVL strom, odhad hloubky
* Operace Insert a Delete v AVL stromech

### (a,b)-stromy

* Vícecestný vyhledávací strom a (a,b)-strom
* Odhad hloubky (a,b)-stromu
* Operace Insert a Delete v (a,b)-stromech
* Volba parametrů (a,b)-stromu
* Červeno-černý strom (LLRB strom)
* Isomorfismus LLRB stromů s (2,4)-stromy
* Operace Insert v LLRB stromu

### Písmenkové stromy (trie)

* Definice trie
* Operace Find, Insert a Delete v trii
* Použití trie k reprezentaci čísel

## Hešování

* Hešování s řetězci v přihrádkach
* Operace Find, Insert a Delete v hešování s řetězci
* Dynamické rozšiřování tabulky
* c-univerzální systém funkcí
* Konstrukce 1-univerzálního systému pomocí skalárního součinu
* Průměrná složitost operací při náhodné volbě hešovací funkce z c-univerzálního systému
* Hešování s otevřenou adresací
* Operace Find, Insert a Delete v hešování s otevřenou adresací
* Složitost otevřené adresace pro plně náhodné vyhledávací posloupnosti

## Rozděl a panuj

* Třídění sléváním (Mergesort)
* Násobení n-ciferných čísel v čase O(nlog23)
* Kuchařková věta (Master theorem)
* Strassenův algoritmus na násobení matic (vzorce nezkouším)
* Quickselect – hledání k-tého nejmenšího prvku
* Průměrná časová složitost Quickselectu při náhodné volbě pivota
* k-tý nejmenší prvek v lineárním čase (algoritmus s pěticemi)
* Quicksort
* Průměrná časová složitost Quicksortu při náhodné volbě pivota
* Lemma o harmonických číslech

## Dynamické programování

* Nejdelší rostoucí podposloupnost
* Editační vzdálenost řetězců
* Konstrukce optimálního BVS
* Floydův-Warshallův algoritmus na výpočet vzdáleností v grafu
* Princip dynamického programování
* Grafová interpretace dynamického programování

# Úvod

### Příklad

* Zadání: Je dána posloupnost reálných x1\_xn. Chceme najít nejdelší rostoucí vybranou podposloupnost (tj. vyberu nějaké prvky a zachovám jejich pořadí).
* Řešení:
  + Najdu všechny podposloupnosti a vezmu tu nejdelší
    - Nalezení všech podposloupností → bijekce s posloupnosti 0 a 1 … tj. jsou čísla v dvojkové soustavě, takže projdu všechna čísla do 2^počet prvků
    - **Binární počítadlo**
    - Správnost: ano, protože projdu všechny podposloupnosti
    - Časová složitost: 2^n podposloupností, n korků pro generování jedné posloupnosti → Θ(2^n \* n)
  + Rekurze
    - Opět generuji všechny podposloupnosti, ovšem rekurzivně
  + Vylepšená rekurze
    - Pokud je podposloupnost klesající, negeneruji ji až do konce
  + Rekurze pozpátku (tj. od n k 0)
    - Počítám maximální délku posloupnosti začínající prvkem i
  + Kešování
    - Pamatujeme si, co už jsme spočítali, reps. pro každé i si pamatuji nejdelší rostoucí podposloupnost
    - Časová složitost: O(n^2), resp. Θ(n^2)
  + Přímý výpočet pozpátku
    - Vždy se odkazuji na hodnoty, které už mám spočítané
    - Časová složitost: Θ(n^2)
    - Dynamické programování
  + Převedení na grafovou úlohu
    - V ={1\_n}
    - E = i,j je hrana iff i<j a xj<xi … tj, mezi prvky je orientovaná hrana, když je cílový vrchol větší
    - Pak hledáme nejdelší cestu
      * Můžeme přidat dva vrcholy x0,xn+1, a mezi nimi hledáme nejdelší cestu → neexistuje polynomiální algoritmus
    - Je acyklický
      * Pro acyklické grafy existuje algoritmus se složitosti Θ(V + E)= Θ(n^2)
  + Datová struktura pro přímý výpočet po zpátku
    - Pamatuje si množinu (klíč, hodnota)
    - Metody:
      * Vložit dvojici
      * Najít max. z hodnot pro klíže z intervalu (časová složitost O(log n))
    - Celková složitost bude O(n \* log n)

# Model RAM

* Viz str. 52
* Random Access Machine
* Počítá s čísly … celá (nekonečný rozsah)
* Paměť je nekonečná, indexování celými čísly
  + Konečné algoritmy využívají konečné množství paměti
* Instrukce
  + Přiřazení – kam←co
    - Operandy:
      * Literál (konstanta)
      * Buňka v paměti [45] … adresa 45
      * Nepřímá adresace [[42]] … buňka, jejíž adresa je v buňce 42
    - Konvence
      * Lokální proměnné jsou na záporných adresách
      * Př: A=[-1], B=[-2] … něco jako registry
  + Aritmetické instrukce – kam←něco+něcoJiného
    - Operace:
      * Aritmetické: +, -, \*, / (celočíselné), mod
      * Bitové: &, |, ^
* RAM simulátor: <https://iuuk.mff.cuni.cz/~husek/ram-sim/ram_simulator.html>
* Čas běhu programu
  + Nutné určit cenu instrukce
    - Jednotková
      * Vznik paradoxů, protože v RAM není omezena velikost slova, se kterým pracuji
      * Nutné tedy omezit délku slova
    - Logaritmická - # bitů čísel, s nimiž pracujeme
    - Relativní logaritmická - # počet bitů čísel / log n
* Prostor běhu programu
  + Stejné jako u času
  + Velikost jedné buňky není def. RAM nijak omezena
* Časová složitost: T(n) = max {t(x); x je vstup velikosti n}
* Prostorová složitost: S(n) = max {s(x); x je vstup velikosti n}

# Asymptotická notace

* Viz str. 50
* Ω(n) – spodní odhad složitosti
* O(n) – horní odhad složitosti
* Θ(n) – řádový odhad složitosti (mezi Ω(n) a O(n))

# Grafové algoritmy

## Prohledávání do hloubky (DFS)

* Viz str. 119
* DFS strom
  + Orientovaný podgraf prohledávaného grafu
  + Orientace jeho hran reprezentuje způsob jeho procházení v DFS
* Lemma: DFS doběhne v čase O(n + m) a prostoru Θ(m+n)
  + Důkaz: Nad každým vrcholem a každou hranou strávíme O(1) času a každý navštívíme právě jednou
* Lemma: DFS navštíví právě ty vrcholy, které jsou z v0 dosažitelné. Tj. stav všech vrcholů je na konci buď „zavřený“ nebo „nenalezený“.

### DFS klasifikace hran

* Stromové hrany (součástí DFS stromu)
* Dopředné hrany (přeskakují nějakou hladinu DFS stromu shora dolů)
* Zpětné hrany (přeskakují nějakou hladinu DFS stromu zdola nahoru)
* Příčné hrany (hrany v jedné hladině)
* Věta: DFS doběhne v čase O(n + m) a prostoru Θ(m+n). Navíc je jeho výsledkem dosažitelnost z kořene a klasifikace všech hran.

### Mosty

* Hrana e ∈ E(G) je most ⇔ G-e má více komponent souvislosti než G
* Lemma: Hrana není most ⇔ leží na alespoň jedné kružnici
* Algoritmus na hledání mostů viz str. 124–125

### Acyklické orientované grafy (DAGy)

* Viz str. 127
* Definice: DAG je acyklický orientovaný graf (tj. speciální třída stromu)
* Lemma: V grafu existuje cyklus ⇔ DFS najde alespoň jednu zpětnou hranu

#### Topologické uspořádání

* Viz str. 128
* Vrcholy DAGu lze lineárně uspořádat tak, že hrany povedou ve směru uspořádání
* Definice: Lineární uspořádání < na vrcholech G je topologické ⇔ pokud pro každou hranu xy platí x<y
* Věta: Graf má topologické uspořádání ⇔ je DAG
* Definice: Zdroj = degin(v) = 0 (nevede do něj hrana)
* Definice: Stok = degout(v) = 0 (nevede do něj hrana)
* Věta: V každém DAGu existuje alespoň jeden zdroj a alespoň jeden stok
* Věta: Pořadí, v němž DFS opouští vrcholy je opačné topologické

#### Silná souvislost

* Viz str. 130
* Definice: relace R na V(G): uRv ⇔ existuje sled mezi u,v
  + Je to ekvivalence
  + Třídy ekvivalence jsou komponenty silné souvislosti
* Definice: Graf komponent silné souvislosti C(G)
  + Vrcholy jsou komponenty G
  + Hrany odpovídají mostům mezi komponentami
* Věta: C(G) je DAG
  + Důkaz: Pokud by v C(G) existoval cyklus, pak by tvořil komponentu souvislosti
  + → existuje v něm alespoň jedna zdrojová a stoková komponenta
* Pozorování: Je-li komponenta C stoková a c ∈ V(C), pak DFS(v) navštíví právě komponentu C
* Pozorování: Pokud spustíme opakované DFS, pak vrchol s max outem leží ve zdrojové komponentně
* Definice: Transpozice grafu – prohodí se orientace všech hran. GT = (V(G); {vu; uv ∈ E(G)})
  + G je DAG ⇔ GT je DAG
  + G a GT mají stejné třídy ekvivalence
  + Prohodí se Zdroj a Stok
* Algoritmus: Hledání stokové komponenty
  + Najdeme vrchol ve zdrojové komponentě v GT
    - Spustím opakované DFS a najdu vrchol s max outem
  + Z vrcholu v ∈ V(G) projdeme celou stokovou komponentu v G
  + Odstraníme ji
  + Opakujeme
* Vylepšení algoritmus:
  + DFS v GT pustím pouze jednou, pak procházím vrcholy sestupně od nejvyšších outů
  + Z vrcholu v ∈ V(G) projdeme celou stokovou komponentu v G …
* Lemma: Pokud C1,C1 jsou komponenty, kde hrana C1C2 ∈ E(C(G)), pak max\_out(c1 ∈ C1) > max\_out(c2 ∈ C2)
* Finální algoritmus:
  + Sestrojím GT
  + Z = prázdný zásobník
  + Pouštím opakovaná DFS v GT, při opuštění přidám vrchol do Z
  + Pro všechny v: komp(v) = nedefinováno
  + Odebírám vrcholy z Z
    - Pokud komp(v) = nedefinováno
      * Spustíme DFS v G z v a chodíme jen do vrcholů, kde komp(v) = nedefinováno a nastavujeme komp = v
* Věta: Finální algoritmus najde komponenty souvislosti v čase a prostoru Θ(n+m)

## Nejkratší cesty

* Viz str. 143
* Graf G = (V,E)
* Definice: Délka l: E → R0+ (kladná reálná čísla včetně 0)
* Délka uv-cesty P: l(P) = sum l(e), e ∈ P
* Definice: Vzdálenost d(u,v) = min {l(P) ; P je uv-cesta}. Pokud neexistuje uv-cesta, pak d(u,v) = ∞
* Lemma: Pokud existuje uv-sled, pak existuje uv-cesta stejné nebo menší délky, tj. l(P) <= l(S)
  + Důsledek: nejkratší sled má stejnou délku jako jekratší cesta
  + Důsledek: platí trojúhelníková nerovnost
  + Pro záporné hrany neplatí
    - Nejkratší sled nemusí existovat
    - Kompromis: Povolení záporných hran ale zakázání záporných cyklů
* Algoritmus pro spočítání všechny vzdálenosti od u
  + Spustíme BFS z vrcholu u
  + Vrcholům přiřazuji čísla vrstev – vzdálenosti od u
  + Nejkratší cesty jsou seznamy předchůdců
* Složitost algoritmu je O(n+m)
* Hrany předchůdců tvoří strom → strom nejkratších cest
* Lemma: Strom nejkratších cest existuje i v ohodnocených grafech
* Pozorování: Prefix nejkratší cesty je opět nejkratší cesta

### Dijkstrův algoritmus

* Algoritmus viz str. 147
* Časová složitost
  + Každý vrchol zavřu nejvýše 1
  + Na každou hranu se podívám nejvýše 1
  + → O(n2)
* Binární halda
  + Tvar úplného binárního stromu
    - Vyvážený strom zaplňovaný zleva
  + Operace ExtractMin – vrácení minima
    - Vrátí a smaže hodnotu kořene
    - Do kořene vložím hodnotu z posledního listu
      * Nutné opravit hladu – prohodím kořen s menším ze synů
        + … na nižší úrovni znovu
    - O(log n)
  + Operace Insert – vložení vrcholu
    - Na začátku vložím na konec poslední hladiny (tj. na konec pole)
    - Probublám nový prvek nahoru … porovnávám ho s rodičem
    - O(log n)
  + Operace Decrese – snížení ohodnocení existujícího vrcholu
    - Neumím hledat vrchol podle hodnoty v čase lepším než O(n)
    - → musím si o každém vrcholu průběžně pamatovat, kde se v haldě nachází
    - O(log n)
* Časová složitost s haldou
  + O(n\*O\_Insert + n \*O\_ExtractMin + m\*O\_Decrese)
  + → O((n+m)log n)
    - Vyplatí se pouze pro řídké grafy, kde m < n^2
* Fibbonaciho halda
  + Insert, Decrese ~ O(1)
  + ExtractMin ~ O(log n)
  + Vyplatí se pouze pro velké grafy
* Platí pro něj vše, co pro relaxační algoritmy
* Invariant o mononomii:
  + Kdykoliv je o otevřený a ze je zavřený, pak h(o) >= h(z)
  + h(z) se nezmění
* Věta: Dijkstrův algoritmus zavírá vrcholy v pořadí podle vzdálenosti, každý dosažitelný právě jednou. A h(v) v době zavření je rovna vzdálenosti id v\_0. V grafu bez záporných hran.

### Relaxační algoritmy

* Zobecnění Dijkstrova algoritmu
* Myšlenky
  + Každý vrchol má nějaké ohodnocení h(v)
  + Relaxace = snažím se snížit h(w) na h(v) + l(v,w) a otevřu w
  + Stavy, abych se nezacyklil → otevřený, pokud se od předchozí relaxace změnilo h(v)
* Obecný relaxační algoritmus
  + Inicializace stavu, začínám ve vrcholu v\_0 → v\_0 je otevřený
  + Dokud existuje otevřený vrchol v → relaxuj v
* Invariant – tvrzení platné v průběhu celého algoritmu
* Pro grafy, které mohou mít záporné hrany platí:
  + Invariant o ohodnocení:
    - Pro všechny vrcholy h(v) nikdy neroste
      * Dk: triviální, při relaxaci pouze snižuji
    - Pokud je h(v) konečné, pak je rovno délce nějakého uv-sledu
      * Dk: indukcí
* Lemma o dosažitelnosti
  + Pokud se relaxační algoritmus zastaví, pak jsou uzavřené vrcholy dosažitelné z v\_0.
* Lemma o vzdálenosti
  + Pokud se relaxační algoritmus zastaví, pak pro každý vrchol je jeho ohodnocení rovno vzdálenosti od vrcholu v\_0.

### Bellmanův-Fordův algoritmus

* Viz str. 152
* Relaxační algoritmus
* Otevřené vrcholy si drží ve frontě
* Zavírá nejstarší z otevřených vrcholů
* Věta: B-F spočítá vzdálenosti od v\_0 v čase O(n\*m) pro libovolný graf bez záporných cyklů (záporné hrany nevadí)

## Minimální kostra

* Viz str. 159
* Definice: Je dán souvislý neorientovaný graf. Hrany jsou ohodnoceny váhami funkci f: E → R. Hledáme kostru T, tedy strom (bez cyklů), takovou, že w(T) = min. Váhy všech hran jsou unikátní → existuje právě jedna kostra.

### Jarníkův algoritmus

* Popis: pěstujeme strom z vrcholu v\_0 tak, že:
  + Vybereme nejlehčí z hran mezi T a V\T
* Hladový algoritmus
* Lemma: Jarníkův algoritmus se zastaví po nejvýše n krocích a T je na konci kostra grafu.
* Definice: Množina R ∈ E je elementární řez ⇔ existuje A je podmnožina V, B = V\A, A,B!=prázdno. R = E(A,B)
* Lemma: (Řezové) Nechť G je graf s unikátními vahami a R je elementární řez v G, e je nejlehčí hrana ∈ R a T je min kostra v T. Pak platí, že e ∈ T.
  + Důkaz:
    - Nechť T je kostra a e není v T
    - Označím a ∈ A, b ∈ B vrcholy hrany e
    - Protože a,b jsou v kostře, pak existuje cesta ab
    - Tj. musí existovat hrana f cesty ab, která je v řezu R
    - Pak ale T-f+e je také kostra a její váha je nižší než původního T
* Důsledek: Jarníkův algoritmus najde minimální kostru
  + Důkaz:
    - Hrany mezi T a V\T tvoří elementární řez
    - → Jarníkův algoritmus vybral nejlehčí hranu tohoto řezu
      * Tj. všechny vybrané hrany leží v každé vybrané kostře
* Důsledek: Existuje jediná minimální kostra
* Důsledek: Minimální kostra je jednoznačné určena pořadím vah
* Poznámka: Pokud by váhy hran nebyly unikátní, pak Jarníkův algoritmus pořád najde minimální kostru. Hrany, které jsou si rovny, lze douspořádat … např. pořadím hran, nebo čímkoliv jiným.
* Časová složitost: O(n\*m) – počet korků \* čas na jeden krok

#### Dijkstrova verze

* Pro sousedy T si pamatuji h(v) = nejlehčí hrana mezi v,T
* Krok:
  + Najdu v s min h(v) a přidám hranu do T
  + Přepočítám ohodnocení sousedních vrcholů
* Stavy:
  + Otevřený: soused T
  + Zavřený: součást T
  + Neviděný – ostatní
* Algoritmus:
  + Viz str. 164
* Imeplementace:
  + Pole → O(n^2)
  + Halda → O(m log n)

### Borůvkův algoritmus

* Popis: Na rozdíl od Jarníkova algoritmu pěstuje více stromečků. V každém kroku – fázi si každý stromeček najde svoji nejlehčí sousední hranu. Zastavíme, když máme jediný stromeček.
* Nutné, aby byly váhy hran unikátní, jinak mohou vznikat cykly
* Lemma: Počet fází <= log n
  + Důkaz: Na konci k-té fáze mají všechny stromečky alespoň 2^k vrcholů
    - IP: pro k=0 zřejmě platí – platí pro k
    - IK: platí pro k+1
      * Každý stromeček na konci fáze vznikne srůstem alespoň dvou stromečků, které podle IP mají >= 2^k
      * 2^k + 2^k >= 2^(k+1)
* Lemma: Výstupem je minimální kostra
  + Důkaz: Každý přidaná hrana je nejlehčí v elementárním řezu mezi stromečkem a zbytek grafu → leží v jediné minimální kostře (opět využívám řezové lemma)
* Časová složitost: O(m log n)
* Věta. Borůvkův algoritmus najde min. kostru v čase O(m log n)

### Kruskalův algoritmus

* Popis: Setřídíme hrany podle vah a postupně je přidáváme. Pokud by vznikl cyklus, hranu vynecháme.
* Lemma: Kruskalův algoritmus najde minimální kostru. (zřejmě najde vždy les)
  + Vytvoří strom – hranu mezi komponentami jsme museli přidat
  + Kostra je minimální – využiji řezové lemma

#### Union-Find

* Udržujeme komponenty souvislosti grafu
* Operace
  + Find(u,v) → vrátí, zda jsou u,v v téže komponentně
  + Union(u,v) → přidá hranu mezi u a v
* Časová složitost Kuskalova algoritmu s Union-Find s implementací s polem má složitost O(n^2)
* Časová složitost Kuskalova algoritmu s Union-Find s implementací s keříky má složitost O(m log n)

# Datová struktura

* Viz str. 81
* Definice: Černá krabička na ukládání dat a práci s nimi. Uživateli poskytuje pouze rozhraní, skrývá vlastní implementaci.
* Příklady
  + Fronta, zásobník, posloupnost
  + Prioritní fronta
  + Množina
    - Podmnožina prvků Universa (např. N, řetězce)
    - Operace
      * Find
      * Delete
      * Insert
  + Slovník
    - Pamatuje si množinu dvojic klíč-hodnota
  + Uspořádaná množina
    - Operace
      * Min, Max
      * Před, Succ (předchůdce, následník)

## Binární strom

* Definice: Zakořeněný strom, každý vrchol má svůj levý a pravý podstrom.
* Statické – naplním při inicializaci
* Dynamické – obsah mohu měnit průběžně
* Značení:
  + Syny vrcholu v značíme l(v) a r(v)
  + Všechny potomky levého syna včetně – levý podstrom značíme L(v)
  + Všechny potomky pravého syna včetně – levý podstrom značíme R(v)
  + Celý podstrom v včetně značíme T(v)
  + Hloubku stromu, tj. počet hladin, měřeno v hranách, značíme h(v). Odpovídá max. počtu hran mezi v a listem.
  + Celkový počet hran značíme m, celkový počet vrcholů n.

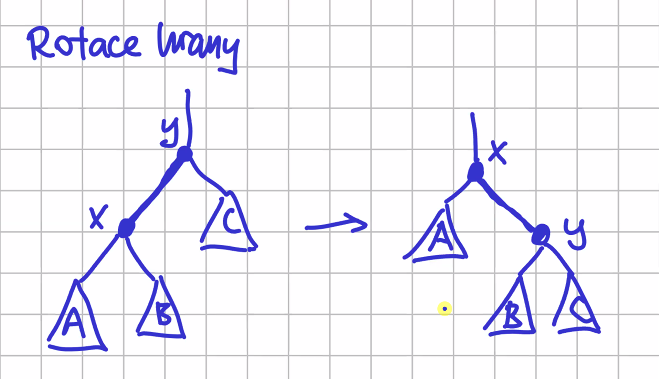
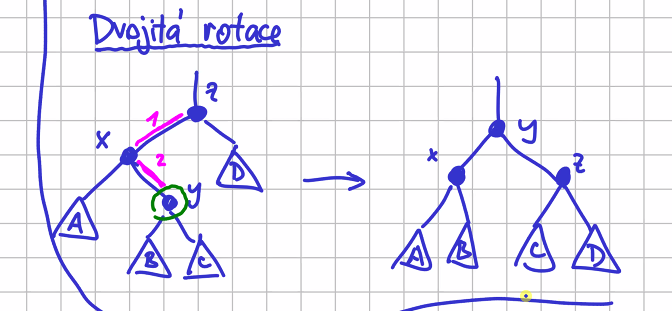
### Binární vyhledávací strom (BTS/BVS)

* Definice: Vrcholy obsahují klíče k(v) ∈ U a navíc pro vrchol v platí, že všechny klíče v levém podstromu jsou menší a všechny klíče v pravém podstromu jsou větší.
  + Tj. pro všechna l ∈ L(v) k(l) < k(v), a pro všechna r ∈ R(v) k(r) > k(v)
* Pokud chybí syn, pak r(v), resp. l(v) = None a T(None) = None a h(None) = -1
* Operace
  + Show: (vyjmenuj všechny klíče v setříděném pořadí)
    - Θ(n)
  + Find(x):
    - Θ(hloubka stromu)
  + Insert(x):
    - Θ(hloubka stromu)
  + Delete(x):
    - Pokud je x list – pouze ho smažu
    - Pokud je x vnitřní vrchol a má jen jednoho syna – pouze smažu a nahradím jeho synem
    - Pokud má x dva syny – nahradím x minimem jeho pravého podstromu nebo maximem jeho levého podstromu
    - Θ(hloubka stromu)
* Závislost operací na hloubce stromu → chceme mělké stromy

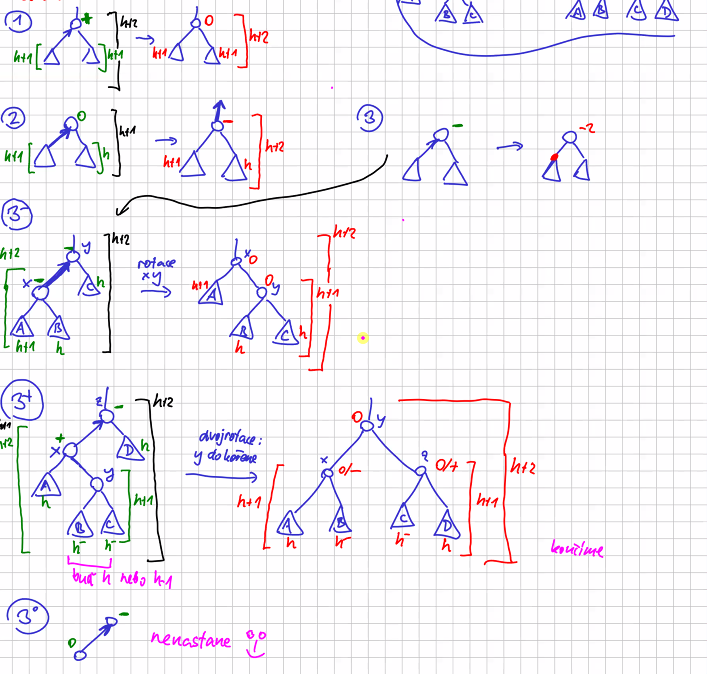
#### Dokonale vyvážený BVS

* Definice: BVS je dokonale vyvážený ⇔ v každém vrcholu ||L(v)|-|R(v)|| <= 1
* Algoritmus: Ze setříděné posloupnosti S vyrobí dokonale vyvážený strom
  + Vyrobí kořen z prvku uprostřed S → s
    - Prvky nalevo od s budou v levém podstromu
    - Prvky napravo od s budou v pravém podstromu
  + Θ(n)
* Věta: Hloubka dokonale vyváženého BVS je <= log2(n)
  + Důkaz: |T(u)| <= |T(v)|/2, kde u je syn v
* Věta: V každé implementaci operací Insert, Delete v dokonale vyváženém BVS má alespoň jedna z operací složitost Ω(n) pro nekonečně mnoho hodnot n.
  + Důkaz:
    - Zvolíme n = 2k-1
      * Pro taková n je tvar dok. v. BVS určen jednoznačně
    - Klíče označíme 1-n
      * Pak existuje jen jedna podoba stromu
      * V listech jsou lichá čísla
    - Po provedení operací Insert(n+1), Delete(1) mám pořád 2k-1, ale všechny prvky jsou o jedna větší
      * Podoba stromu je opět jednoznačná
      * V listech jsou sudá čísla
    - Prohodila se čísla v listech a vnitřcích vrcholech (sudá a lichá)
      * Pro Ω(n) vrcholů změn ukazatelů – změnilo se n/2 vrcholů, co už nejsou listy a jiné je nahradily
    - Opět provedu operaci Insert(n+2), Delete(2)
      * Opět změna Ω(n) ukazatelů vrcholů
    - → čas na dvojici je Ω(n) → Insert() nebo Delete() trvá Ω(n)

#### Hloubkově vyvážený BVS – AVL strom

* Definice: Pro každý vrchol v platí, že |h(l(v))- h(r(v))| <= 1
* Věta: Dokonale vyvážené stromy jsou hloubkově vyvážené.
* Věta: Hloubka AVL stromu s n vrcholy je Θ(log n).
  + Důkaz:
    - Kolik vrcholů Ah může mít min AVL strom dané hloubky h.
    - Máme-li strom hloubky h, pak podstromy kořeny mají hloubky h-1/h-1 nebo h-1/h-2
      * V minimu Ah=1+ Ah-1+ Ah-2
    - Dokážeme indukcí, že Ah <= 2h/2
  + Dolní odhad hloubky je asymptoticky Θ(log n)
    - TODO důkaz
* Definice: Rotace hrany e mezi vrcholy x,y.  
  
* Definice: Dvojitá rotace dvou hran z,x a x,y, které jsou nad sebou a mají opačný směr. Vzniká složením rotací hrany x,z a následně z,y.  
  
* Definice: Znaménko vrcholu δ(v) = h(r(v)) – h(l(v))
  + Může být -1, 0, +1 (-1 = levý je hlubší, +1=pravý je hlubší)
* Metody
  + Insert(x)
    - Do vrcholu přijde z podstromu signál, že se po přidání prohloubil, tj. hloubka se zvýšila. Uvažuje búno, že přichází vždy zleva. Zprava je to symetrické (prohodí se znaménka).
    - Případy:

1. Vrchol má znaménko +1, přijde do něj zleva příznak, že se levý podstrom prohloubil
   * Pak δ(v)=0 → není nutné ve vyrovnávání pokračovat
2. Vrchol má znaménko 0, přijde do něj zleva příznak, že se levý podstrom prohloubil
   * Musím příznak poslat výše, znaménko se změní na -1
3. [-] Vrchol má znaménko -1, přijde do něj zleva příznak, že se levý podstrom prohloubil
   * Znaménko se změní na -2 → musím zasáhnout – opravím to rotací hrany x,y
   * Jednoduchá rotace
4. [+] Vrchol má znaménko -1, přijde do něj zleva příznak, že se pravý podstrom prohloubil
   * Dvojitá rotace
5. [0] Nikdy nenastane, resp. nastane pouze když vyrážíme z nově vzniklého vrcholu



* + Delete(x)
    - Buď mažu list, nebo vrchol s jedním synem. Pokud má dva syny, pak jej nahradím minimem zprava nebo maximem zleva.
    - Na rozdíl od insertu tentokrát přijde do vrcholu signál, že se hloubka podstromu snížila
* Věta: Insert, Delete a Find mají v AVL stromu logaritmickou složitost

### Vícecestné vyhledávací stromy

* Definice: Externí vrcholy. Tam, kde vrcholům stromu chybí listy je doplníme externími vrcholy.
  + Pak platí:
    - Všechny externí vrcholy jsou list.
    - Všechny interní vrcholy mají právě dva syny
    - Externí vrcholy odpovídají intervalům mezi klíči
* Definice: Vícecestný vyhledávací strom je zakořeněný, má interní a externí vrcholy a synové každého vrcholu mají pořadí. V interních vrcholech jsou vzestupně uložené klíče (alespoň 1) x\_1 … x\_k a platí, že počet synů = k + 1. Klíče v podstromech jsou pak oddělené klíči ve v.

#### (a,b)-stromy

* Definice: (a,b)-strom je vícecestný strom, kde a>=2 a b>=2a-1. Kde všechny externí vrcholy jsou stejně hluboko a interní vrcholy mají a až b synů, tj. (a-1) až (b-1) klíčů s výjimkou kořene, který má 2 až b synů, tj. 1 až (b-1) klíčů.
* Lemma: (a,b)-strom s n klíči má hloubku Θ(log n). (konstanta v Θ závisí na a, b)
  + Ptám se. Kolik nejméně klíčů může mít (a,b)-strom hloubky h
  + Pozorování: každý vrchol má minimální počet synů (klíčů) … v minimálním případě
  + TODO … nějaké odvozování
  + Důsledek: Každý strom má 2ah-1-1 klíčů → počet vrcholů roste exponenciálně → max. hloubka roste logaritmicky ~ logan
    - Minimální hloubka stromu roste také logaritmicky ~ logbn
* Find(v)
  + Při O(1) na hladinu trvá O(log n)
* Insert(v)
  + Nemůžu nový vrchol jen přilepit dolů. Porušil bych to, že všechny externí vrcholy leží na stejné hladině.
  + Přidám tedy v k otci listu, tj. na nejnižší vnitřní hladinu – může se stát, že přeteče, tj. má víc synů, než je povoleno.
  + Při přetečení má tedy b synů. Vrchol tedy rozdělím na dva, které připojím k jeho otci. Tomu přidám prostřední klíč z aktuálního.
  + Při dělení vrcholu ne nestane, že by jeden z nových měl méně než a klíčů, protože b >= 2a-1.
* Delete(v)
  + Vrchol v smažu a nahradím jej nejmenším synem z pravého podstromu
  + Může se stát, že vrchol, ze kterého jsem odebral klíč z vrcholu, má teď méně než a-1 klíčů.
    - Pokud má jeho bratr a-1 klíčů – spojím je
    - Pokud má jeho bratr více klíčů – jeden mu vezmu
* Velikost stromu
  + Nechceme b >> 2a … typicky buď b=2a-1 nebo b=2a
  + Nechceme velké a … optimální jsou (2,3), (2,4)

#### Červeno-černé stromy

* Překlad (2,4)-stromu a BVS

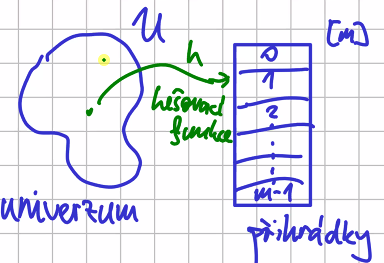
### Reprezentace řetězců

* Definice: Řetězec je množina znaků abecedy ∑ (nejvýše spočetné)
* Použití BVS pro ukládání a vyhledávání v řetězcích Θ(log n \* délka řetězce)

#### Písmenkový strom (trie … kombinace z tree + retrival)

* Viz. Str 91
* Reprezentace BVS, v každém vrcholu pole ukazatelů indexované abecedou a označení, zda vrchol ukončuje nějaké slovo z množiny slov
* Metody
  + Member(y:slovo)
    - Θ(|y|)
  + Insert(y:slovo)
    - Θ(|y|)
  + Delete(y:slovo)
    - Odznačí konec slova
    - Zdola maže vrcholy, které nejsou označené a už nemají žádné syny
    - Θ(|y|)
* Paměťová složitost Θ(suma|x\_i|\*|∑|)
* Pro velké abecedy je nevhodné idexovat v každém vrcholu abecedou. Místo pole budeme mít v každém vrcholu BVS se syny

### Hešování

* Hešovací funkce přidělí vstupu nějakou hodnotu z univerza (přihrádku)
* Pro řešení kolizí bude v každé přihrádce seznam prvků, které tam patří
* Volíme hešovací funkci náhodně z nějaké množiny (systému) funkcí
* 
* Definice: Systém funkce z U(univerza) do [m] je c-univerzální pro c ∞ ⇔ pro všechna x,y ∈ U, x !=y Pravděpodobnost[h(x), h(y)] <= c/m
* Existuje 1-univerzální sysém
* Lemma: Nechť H je c-univerzální systém fcí z U do [m], x1\_xn ∈ U navzájemn různé, y ∈ U. Pak platí, že Existuje h ∈ H,
* **TODO – doplnit**
* Důsledek: Existují časové složitosti operací Find, Insert, Delete v O(n/m)

#### Typy hešovacích funkcí

* Lineární kongruence
  + x → (ax) mod m, kde a je nesoudělné s m, m je prvočíslo
* Mutiply-shift a\*x >> 2 … tj. vynásobím a vezmu horní dva bity
  + X → (ax mod 2^w) >> w-k
* Hašování řetězců
  + Skalární součin s náhodným vektorem mod m
    - X → (suma a\_i\*x\_i) mod m
  + Polynomem
    - X → (suma a^i\*x\_i) mod m

#### Nafukovací hašovací tabulka

* Sledujeme n/m – poměr mezi počtem prvků a počtem přihrádek. Tj. průměrný počet prvků přihrádkách. Chceme jej udržovat pod nějakou konstantou
* Vzroste-li n/m příliš, přehešujeme do více přihrádek (konstanta-krát)
  + Zvolíme m → 2m

#### Otevřená adresace

* TODO
* Příklady
  + Hešování s lineární přidáváním

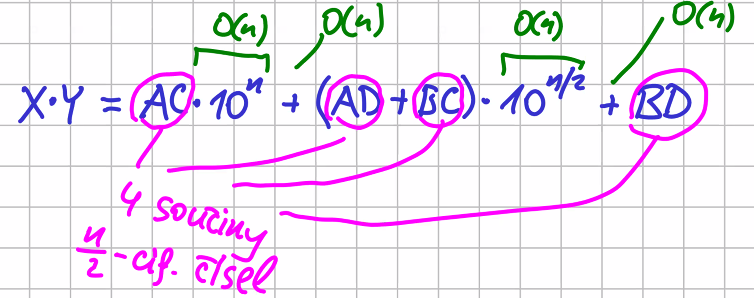
# Rozděl a panuj (Divide et impera)

## Příklady

### MergeSort

* Algoritmus třídění sléváním
  + TODO …

## Násobení čísel

* Týká se velkých čísel (např. o n cifrách, desítková soustava)
* Algoritmy:
  + Školní algoritmus Θ(n^2)
  + Nápad: 
    - Složitost: T(n) = 4\*T(n/2) + n, T(1)=1
    - → strom rekurze má n^2 listů … tj. není lepší, než školní algoritmus
    - Pokud se mi povede místo 4 násobení udělat jen → složitost bude lepší než kvadratická
  + Pro malá n se vyplatí použít školní algormitmus

## Rekurence obecně

* T(n) = a\*T(n/b) + Θ(n^c), T(1) = 1