Algoritmy a datové struktury 1

Tomáš Turek

Přednáška 1

Úvod

Analýza složitosti

- časová složitost a prostorová složitost

Jak přemýšlet nad algoritmy

- lze vymýšlet postupně lepší a lepší řešení

Hrubá síla

- vždy se lze podívat na řešení pomocí hrubé síly
- je to celkem hloupé řešení, ale řeší to problém
- *někdy se hodí na řešení okrajových situací pomocí zarážek*

Rekurze

- nějaké problémy lze řešit pomocí rekurze
- to může být dobré ale i horší, záleží jak se implementuje rekurze
- rekurze se dá postupně zlepšovat
  - předávám méně dat
  - nebudu lézt do míst, které už nemůžou být lepší
  - v nějakých problémech je dobré řešit pozpátku
  - také se hodí cachovat mezivýsledků (memoizace)

Dynamické programování

- viz budoucí přednáška

Jaká bude interpretace

- třeba pomocí grafů
- neboli se převede problém na jiný, který už umíme řešit

- také se dá využít nějaká datová struktura, která má důležité vlastnosti

Vymyšlení chytáku

- tady žádný postup není
- akorát musí být nějaký nápad

Algoritmus

- definujeme pomocí abstraktního stroje RAM (Random access machine)

RAM

- počítá s čísly, vše se kóduje čísly
- paměť je nekonečné očislované pole (... -2 -1 0 1 2 ...)
- operace:
  - přiřazení `kam <- co`
  - aritmetika `z <- x + y`
  - `+ - / * mod`
  - `& | ^` (and, or, xor)
  - `>> <<` bitshift
  - `halt` zastavení (na konci je vždy)
  - `goto` skok na nějakou instrukci `SEM: X <- Y goto SEM`
  - `if X (o) Y` o - (`= < > <= >= !=`)
- operandy
  - literál `42`
  - buňka v paměti `[42]`
  - nepřímá adresa `[[42]]`
- pojmenovávací konvence
  - `A = [-1]`, `B = [-2]`, ... , `Z = [-26]`

Přednáška 2

- výpočet:
  - dostaneme do paměti vstup
  - provádíme instrukce postupně
    - pokud není nějaký skok
- ceny instrukce:
  1. může být jedotková (problém s velkým číslem)
  2. jednotkové ale omezená čísla (omezili jsme prostor polynomem)
  3. logaritmická - # bitů s kterými pracujeme (nepohodlné)
  4. relativní logaritmická - # bitů čísel / log n (horní celá část) - většinou jednotková cena
- Definice pro program a vstup x:
  - čas běhu programu
    - t(x): součet cen instrukcí

    - prostor běhu programu
        * s(x): maximální použité buňky
- Definice pro program:
    - časová složitost
        * T(X): max{t(x), vstup x velikosti n}
            · veliksot se může měřit různými způsoby
    - prostorová složitost
        * S(X): max(s(x))
- časová složitost
    - spočítat přesnou složitost je těžké
    - je lepší odhadovat (asymptotika)
    - konstanty se chovají jinak na strojích a jazycích
    - pro velké vstupy rozhoduje asymptotika
        * naopak pro malé vstupy se může chovat trochu jinak

Grafové algoritmy

Reprezentace grafu

Matice sousednosti

- čtvercová matice přes vrcholy (V)
- když existuje hrana (E) mezi vrcholy, tak je tam 1, jinak 0

Seznam sousedů

- pro každý vrchol si pamatuji seznam jeho sousedů
- může být jako spojový seznam

BFS (prohledávání do šířky)

- nalezený vrchol dám do fronty a pak se spustím na nový z vrcholu
- algoritmus vlny

DFS (prohledávání do hloubky)

- zavolám se na vrchol a zavolám se na sousedy
    - pokud je nenalezen, tak se na něj zavolám rekurzivně
    - jakmile se pak vrátím tak ho zavře
- pamatuji si pro každý vrchol stav
    - stav je (nenalezen/otevřený/zavřený)
- budu mít budíky
    - pro každý vrchol si pamatuji kdy jsem ho otevřel a zavřel
- časová složitost je O(m+n)
    - každý vrchol je otevřen max jednou (n)
    - součet přes stupně výstupů (m)
- **Lemma**:

  - po DFS jsou všechny vrcholy buď uzavřený (tak je dosažitelný) nebo nenalezený
    * když jsem ho zavřel, tak jsem se do něj dostal a proto jsem se do něj musel dostat
    * kdyby existovali špatné vrcholy (dosažitelné a nenalezený)
      · najdu předposlední vrchol a ten je dobrý
      · ale ten musel navštívit hranou do toho špatného

Přednáška 3

- dá se značt i uzávorkováním (otevírací, když vlezu do vrcholu a zavírací, když ho zavřu)
- také se dá popsat DFS stromem
- klasifikace nenavštívených hran (DFS klasifikace):
  - zpětná hrana
    * vede do předchůdce
  - dopředná hrana
    * vede už do uzavřeného vrcholu
  - příčná hrana
    * podobná jako dopředná akorát daný vrchol není jeho potomkem
    * vede vlastně 'dozadu v čase'
  - stromová hrana
    * to je hrana která se prošla
  - tyto hrany jsou jediné možné
- pokud se jedná o neorientovaný graf tak není příčná hrana
- typ hrany zle získat v $O(1)$ díky budíkům (čas in a out)
- **Věta**: DFS v čase $\Theta(n+m)$ a prostoru $\Theta(n+m)$ najdu dosažitelné vrcholy a určíme typy hran.

Mosty (v neorientovaných grafech)

- **Df:** Most je hrana, kterou kdyý odeberu, tak se mi zvýší počet komponenet souvislosti.
- **Lemma:** Hrana e není most <=> e leží na kružnici.
- jelikož není příčná hrana, tak se musí najít jestli není zpětná hrana
- **Df:** low(v):= min(in(y) | xy je zpětná hrana a x pod v).
  - celkově je to minimum přes low(s) synů a in(y) vy je zpětná hrana
- **Věta:** Algoritmus nalezne všechny mosty v čase a prostoru $\Theta(n+m)$.

Acyklické orientované grafy (DAGy)

- Je graf DAG?
  - **Lemma:** $\exists$ dosažitelný cyklus <=> DFS najde zpětnou hranu.
    * abychom našli všechny cykly, tak opakujeme DFS pro všechny nenalezené vrcholy, nebo udělá zdroj, který vede do všech vrcholů
- Topologické uspořádání.

  - **Df:** Lineární uspořádání, tak že $\forall xy \in E(G) x \leq y$.
  - Cyklus nemá TU.
  - **Věta:** Pro každý DAG existuje TU.
    * **Df:** Zdroj je vrchol do kterého nevede hrana.
    * **Df:** Stok je vrchol ze kterého nevede hrana.
    * Pokud budu utrhávat Zdroje, tak zjistím jestli to je DAG.
  - **Věta:** Pořadí v němž DFS prohledává vrcholy je opačný oproti TU.
- Topologická indukce - c(v) je počet cest z u do v - pokud je DAG, tak stačí najít TU a jakmile najdu u, tak pro další vrcholy sečtu c(v) jejich předchůdců
- Plánování

Silná souvislost (orientovaný graf)

- **Df:** Relace R: x R y, tak že $\exists$ sled z x do y. Potom RR: x RR y: x R y & y R x.
  - R je jen značení.
  - je to ekvivalence
  - třídy ekvivalence jsou komponenty silné souvislosti, potom graf je silně souvislý pokud # komponent sil. s. je 1
- **Df:** Graf komponent je graf kde vrchol je komponenta sil. s. a hrana pokud vede z jednoho vrcholu do jiného vrcholu hrana s tím, že jsou v jiných komponentách.
- **Lemma:** Graf komponent je DAG.
  - Pokud by existoval cyklus, tak by to byla jedna komponenta silné souvislosti.

Přednáška 4

- $\exists$ aspoň 1 zdoj a 1 stok
- pokud spustím DFS v stokové komponentě, tak najdu jen ty vrcholy ve stokové komponentě
- pokud spustím opakovaně DFS, tak vrchol s největším outem musí být ve zdrojové komponentě
  - tedy pokud udělám opačný graf/transponovaný graf (otočení hran) tak najdu stokovou komponentu
  - pak můžu smazat stokovou komp. a opakovat (to je pomalé)
  - nebo procházím vrcholy v pořadí klesajících outů v G^T
    * pokud vrchol ještě nemá přiřazenou komp. tak spustím DFS v G
    * **Lemma:** Pokud C_1 C_2 jsou komponenty tak že mají v grafu komp. hranu mezi sebou ($C_1 \to C_2$), tak max out(u) > max out(v), kde $u \in C_1$ $v \in C_2$
- Postup algoritmu:
  - Sestrojit G^T a vytvořit zásobník, projít DFS G^T a až opustím vrchol, tak ho přidám do zásobníku. Následně pak vytahuju ze zásobníku a

pokud nemá komp, tak na něj pošlu DFS a označím danou komponentu.

Nejkratší cesta

- v orientovaném grafu a s hodnocenými hranami (nezáporné)
  - pokud bych použil záporné hrany tak mohou nastat problémy
    - neplatí trojúhelníková nerovnost
    - záporné cykly
    - rozdíl mezi použitím sledu a cesty
    - kompromis: povolení záporných hran, ale zakázané záporné cykly
- délka uv cesty je součet všech hran, přes které prochází
- vzdálenost je minimum z délek uv cest/sledů (může být $+\infty$)
- **Lemma:** Pokud S je uv sled, tak $\exists$ cesta P, která je $l(S) \leq l(P)$ jak sled.
- platí i trojůhelníková nerovnost $d(u, w) \leq d(u, v) + d(v, w)$.
- pokud všechny délky hran jsou jedna, tak stačí jen spustit BFS
  - u vrcholů si značím číslo vrstvy a pomatuji si předchůdce

Strom nejkratších cest

- strom na V, podgraf G, orientovaný od kořene u, všechny vrcholy jsou ve stromu
- uloženínejkratších cest z u do v
- **Lemma:** Existuje strom nejkratších cest i v ohodnoceném grafu.
  - prefix nejkratší cesty je zase nejkratší cesta
- Pokud mám ale graf který má délky hran $\in N$, tak to lze převést na stejný problém.
  - tedy dané hrany si rozdělím na n podhran a jen jakmile se něco změní, tak přepočítám vzdálenosti

Přednáška 5

Dijkstrův algoritmus

- přidělení prvnímu vrcholu hodnotu nula a ostatním $+\infty$
- vyberu s nejmenším ohodnocením a tam půjdu
  - pak přenastavím sousedy, pokud se tam mohu dostat rychleji přes nový vrchol
- samotná časová složitost je $O(n^2)$
- použitím vhodné DS - halda
  - potřebné operace:
    - `extract_min`, `insert`, `decrease`
  - při použití je časová složitost $O(n * T_{extract\_min} + n * T_{insert} + m * T_{decrease})$
- **Invariant:** Pokud mám otevřený vrchol o a zavřený z, tak $h(z) \leq h(o)$ a taky $h(z)$ se nemění.

- **Věta:** Dijkstrův algoritmus zavírá vrcholy v pořadí podle vzdálenosti od u každý dosažitelný právě jednou, $h(v)$ po zvření je rovno k $d(u, v)$.

Binární halda

- binární strom, kde každý má dva potomky až na poslední vrstvu, kde se zaplňuje od leva do prava
- předchůdce je menší (minimální halda)
- `insert`
  - přidání na jediné místo kde má být a potom může probublat nahoru, pokud je lehčí
- `extract_min`
  - kořen vyměním se spodním a pak ten propadne dolu
- `decrease`
  - jak najdu prvek, tak ho zmenším a kdyžtak probublá nahoru
  - najdu prvek pomocí ukazatelů na prvky haldy
- všechny operace jsou v $O(\log n)$

Fibonacciho halda

- lepší halda pro tohle použití
- `extract_min` v $O(\log n)$
- `decrease` a `insert` v $O(1)$

Relaxační algoritmus

- do této skupiny patří i Dijkstrův algoritmus
- držím u vrcholů jejich hodnotu $h(v)$ a při vstupu do vrcholu se upraví sousedi
- **Lemma:** Pokud se algoritmus zastaví, tak každý $v \in V$ je dosažitelný z u <=> v je uzavřený <=> $h(v)$ je konečné.
- **Lemma:** Pokud se algoritmus zastaví, pak $v \in V$ $h(v) = d(u, v)$.

Belmanův-Fordův algoritmus

- typ relaxačního algoritmu
- otevřené vrcholy jsou ve frontě
  - zavírá nejstarší z otevřených vrcholů
- **Věta:** B-F algoritmus spočítá vzdálenosti $d(u, -)$ v čase $O(nm)$ pro libovolný graf bez záporných cyklů.
- Fáze výpočtu:
  - F_0 je otevření u
  - F_i zavírání vrcholů z F_{i-1} a oteveríná jejich následníků
- **Invariant:** Na konci fáze F_i $\forall v \in V h(v) \leq$ délka nejkratšího max uv sledu o max i hran.

Přednáška 6

Minimální kostra

- kostra je podgraf, který nemá cykly ale obsahuje všechny vrcholy
  - minimální je, že součet hran je minimální
- je dán souvislý neorientovaný graf
  - všechny hrany mají danou váhu

Jarníkův algoritmus

- vybereme si jeden vrchol
- z toho budeme pěstovat strom
  - vybereme nejlehčí z hran mezi vrcholy co jsou ve stromu a těmi co nejsou ve stromu
  - pokud ta nejlehčí nevytvoří cyklus, tak ho přidáme
- jedná se o hladový algoritmus (greedy algorithm)
- Elementární řez: vybrání podmožiny hran (R) a dvě podmnožiny vrcholů (A a B). S tím, že A a B nemají ani jeden společný prvek a mezi nimi vedou hrany jen z R.
  - **Lemma:** Pokud nalezneme v řezu nejlehčí hranu, tak se vyskytuje v každé min kostře. S tím, že váhy jsou unikátní.
    - pokud by tam nebyla, tak tam bude nějaká jiná hrana z řezu a tu pokud vytáhnu, tak dostanu dvě komponenty a mohu přidat danou nejlehčí hranu a najednou kostra je lehčí
- **Lemma:** Jakmile se Jarn. alg. zastaví tak strom bude kostra.
  - hrany mezi T a zbytkem grafu tvoří ele. řez, použitím Lemma o elem. řezu je jasné, že musí být v kostře
  - také existuje právě jedinná min kostra (unikátní váhy)
  - min kostra je jednoznačně určena pořadím hran podle vah
- pokud by váhy nebyly unikátní, tak by vše mělo stále fungovat
- Verze podle Dijkstry:
  - pro všechny sousedy si budu pamatovat min z vah se sousedy ze stromu
  - pak přidám minimální z těchto a přepočítám ohodnocení
  - otevřené vrcholy jsou sousedé, zavřené v T a nenalezené zbytek
  - zase potřeba `extract_min`, `insert`, `decrease`
  - takže je dobré použít bin. haldu

Borůvkův algoritmus

- mám několik stromečků a každý stromeček přidá nejlehčí hranu s okolím
- na začátku jsou všechny vrcholy stromečky
- konec je když už mám jen jeden stromeček
- **Lemma:** $\#fází \leq \log n$
  - každý stromeček na konci fáze vznikne spojením aspoň dvou stromečků
  - takže na konci každé fáze má každý stromeček asapoň 2^k vrcholů

- **Lemma:** Strom je min kostra.
  - využití lemma elem. řazu
  - v každém kroku je elem. řez mezi stromečkem a ostatním (pro všechny)
- **Věta:** Borůvkův algor. najde v čase $O(m \log n)$ min kostru.
- dá se paralelizovat na vícero procesorech

Kruskalův algoritmus

- hladový algoritmus
- setřídíme hrany podle váhy
- pokud přidáním hrany nevznikne cyklus, tak ho přidáme
- **Lemma:** Alg. najde min kostru.
  - podgraf je vždy les, na konci pak strom
  - je minimální (zase díky lemma elem. řezu)
- složitost závisí na testování jestli nevznikne cyklus

Union-FInd (DFU)

- udržujeme komp. souvislosti
- operace
  - Find (u,v):
    - jsou u a v v komponentě
  - Union (u,v):
    - sjednocení u a v komponenty
- lepší než si pamatovat číslo komponenty pro vrchol je reprezentace komponenty pomocí keříků
  - vrchol si pamatuje předchůdce
  - FInd pak najde kořen u a kořen v a porovná ($\log n$)
  - Union zase najde kořeny a nastavíme otce jednoho kořene na druhý. S tím, že připojím mšlčí pod hlubší.
  - **Lemma:** Keřík hloubky h má aspoň 2^h vrcholů.

Přednáška 7

Datové struktury

- struktura uchovávající data
- můžeme po ní chtít jisté operace
- ale v podstatě nezáleží jak je implementovaná
- rozhraní (statická - postaví se jednou / dynamická - pořád se mění)
  - fronta, zásobník, posloupnost
  - prioritní fronta (halda)
  - množina
  - slovník (klíč, hodnota)

  - uspořádaná množina
- implementace
  - pole
  - spojový seznam
  - halda
  - vyhledávací strom

Binární vyhledávací strom

- Binární strom má kořen a ten může mít levý a pravý podstrom (záleží na tom jestli je levý nebo pravý).
- hloubka := maximální počet hran do listu (pokud je prázdný tak -1)
- BVS je typický binární strom s tím, že klíče v levém podstromu $<$ klíč v kořenu $<$ klíče v pravém podstromu
- operace
  - `Show` (enumerate) $O(n)$
    - projít celý strom (nejdřív levý podstrom pak kořen a pak pravý)
  - `Find (x)` $O(hloubka)$
    - pokud je kořen stejný jako x, tak jsem našel, jinak jdu do podstromu podle toho jestli je větší nebo menší
  - `Insert(x)` $O(hloubka)$
    - v podstatě hledám jakoby tam byl a pak pokud je tam prázdný vrchol, tak ho tam vložím
  - `Delete(x)` $O(hloubka)$
    - nejdříve najdu a potom záleží jestli to je list / má dva syny / jednoho syna
    - upravím aby to byl list (najít ten nejbližší a vyměnit)
- protože záleží na hloubce stromu, tak bychom chtěli aby strom nedegeneroval

Dokonale vyvážený BVS

- **Df:** Je dokonale vyvážený $<=> \forall v \; ||L(v)| - |R(v)|| \leq 1$.
- hloubka $\leq \log_2 n$ - potomek má $\leq$ polovinu předchůdce
- tento strom je těžký udržovat a pak taky není tak rychlý (přísná podmínka)
- **Věta:** V každé implementaci operací `Insert`, `Delete` v d. v. BVS má aspoň jednu z operací složitost $\Omega(n)$ pro nekonečně moho hodnot n.
  - pokud si vezmu plný strom a zaplněný vzestupně, pak smažu první a přidám n+1 prvek, potom $\Omega(n)$ listů změnili klíč

AVL - Hloubkově vyvážený BVS

- **Df:** Je dokonale vyvážený $<=> \forall v \; |h(l(v) - h(r(v))| \leq 1$.
- **Věta:** Hloubka AVL stromu s n vrcholy je $\Theta(\log n)$.
  - dá se ukázat pomocí Fibonacciho posloupnosti
  - nebo vždy ukázat nejhorší možný strom a potom najít, že další je spojení předchozích a potom mat. indukcí

- je potřeba zajistit aby operace `insert` a `delete` zachovával AVL strom
- v každým vrcholu si budeme pamatovat znaménko
  - hloubka pravého - levého podstromu
- potom pomocná operace rotace
  - jedná se o překlopení hrany (vyměním předchůdce za jednoho jeho následníka)
- a taky dvojitá rotace
  - týká se dvou hran za sebou, které jsou zalomené
  - po tom se ten poslední stane kořenem a vše ostatní už je dané
- `Insert (x)`
  - nejdříve přidá list na svoje místo
  - zjednodušení jen na levou část a to tak, že z levého podstromu přijde signál změny
  1. Pokud je kořen + (že pravý je hlubší)
     - z něho se stane 0 a nemusí posílat signál dál
  2. Kořen je 0
     - levý je teď hlubší než pravý a taky musíme poslat informaci dál
  3. Kořen je -
     - už musíme zasáhnout tři případy
     1. (-) a to že levý podstrom je taky -
        - stačí jen zrotovat hlavní kořen s kořenem levého podstromu
     2. (+) kořen levého podstromu má +
        - tady se použije dvojitárotace s pravým potomkem kořene levého podstromu a kořeny
     3. (0)
        - tohle nikdy nenastane
        - respektive jen když přidám list, ale to není problém
- `Delete (x)`
  - převádíme na mazání listu
  - bude se podílat signál o snížení podstromu (levý)
  1. Pokud kořen je -
     - odeberem a kořen bude 0 a musíme poslat signál
  2. Kořen je 0
     - jen se změní 0 na +
  3. Kořen je +
     - případy podle kořene pravého podstromu
     1. (+)
        - opravíme rotací s kořenem a kořenem pravého podstromu
        - kořen se změní na 0 a musíme poslat signál dál
     2. (0)
        - jak předtím stejná rotace
        - kořen má pak - a nemusíme posílat dál
     3. (-)

        - dvojitá rotace
        - hlavní kořen a pravý podstrom a levý podstrom pravého podstromu
        - kořen bude 0 a musí se poslat dál signál
- **Věta:** `Insert`, `Delete` a `Find` mají časovou složitost $\Theta(\log n)$.
- externí vrcholy jsou prázdné vrcholy, které jsou zároveň listy
    - všechny internívrcholy mají dva syny
    - externí vrcholy značí intervaly

Vícecestný vyhledávací strom

- podobný jako BVS, ale v interních vrcholech je více klíčů, kterérozdělují jejich podstromy
- každý vrchol má k klíčů a k+1 podstromů
- zase může degenerovat, proto všechny externí vrcholy budou na jedné hladině
    - vypadá hodně striktně, ale zase může být více klíčů

(a,b) stromy

- vícecestný vyhledávacístrom
- $a \geq 2$ a $b \geq 2a - 1$
- všechny externí vrcholy jsou na stejné hladině
- int. vrcholy mají a až b synů
    - vyjma kořene tem má 2 až b

Přednáša 9

- **Lemma:** (a,b) strom s n vrcholy má $\Theta(\log n)$ hlounku.
    - pokud budu mít minmimální počet klíčů, tak na každé hladině mám $2a^{i-1}$ vrcholů
    - součet všech vrcholů pomocí geometrické řady
    - potom pro maximální počet klíčů
- `Find (x)`
    - konstantnína hladinu a jinak podobně BVS
- `Insert (x)`
    - hledám kam přidám a pokud bych ho přidal místo externího, tak
        * vždy dáme na poslední vnitřní hladinu (přidáním klíče)
        * pokud přeteče, tak
            · předám prostřední klíč otci a rozdělím syna na půl (štepení vrcholů)
            · pokud přeteče i otec, tak opakujeme dál
- `Delete (x)`
    - nejprve převést na Delete na nejnižší hladině (njebližší prvek pravé minimum)
    - teď vrchol mohl podtéct

        * najdeme si sourozence (záleží kolik má bratr klíčů)
            · pokud má více jak minimum, tak jeho nejbližší otci vyměním s otcem a jeho si vezmu já
            · pokud má právě minimum, tak se sleváním spojíme a ještě ubereme otcovi
            · pak je možné že otcův vrchol podtekl a tak pokračujeme dál
- je potřeba volit a a b
    - typicky chceme blízké b a malé a
    - nejpoužívanější je (2, 3) strom a (2, 4) strom
    - na disk je dobré použít (256, 512) strom
    - pak zasse pro keš je dobrý (4, 8) strom

LLRB

- left leaning red and black tree
- dá se zakódovat z (2, 3) stromu
- je to BVS s externími vrcholy a hranami (červené nebo černé)
- **Axiomy:**
    1. Dvě červené hrany nejsou za sebou.
    2. Pokud vede jen jedna červená hrana, tak doleva.
    3. Hrany do externích listů jsou černé.
    4. Na každý cestě kořen-list je stejný počet černých hran.
- **Lemma:** Existuje bijekce mezi LLRB a (2, 3) stromy.
- při implementaci je dobré si barvu pamatovat v dolním vrcholu
- dvě pomocné operace:
    - rotace R hrany
        * přehození otce a syna na červené hraně
        * zachovává černé axiomy, ale ne červené
    - přebarvení čtyřvrcholu (třešnička)
        * červené se stanou černé a tyto budou mít nahoru červenou
        * zachovává B axiomy ale ne R axiomy
- `Insert (x)`
    - směrem dolů přebarvujeme vrcholy
    - nahradíme externí vrchol a pod ním budou nové dva externí vrcholy, vnitřní je připojený červenou hranou
    - směrem nahoru pak rotacemi spravujeme R axiomy
- `Delete (x)`

Přednáška 10

Trie (písmenkový strom)

- kdybychom řetězce ukládali do BVS, tak bude pomalejší protože logaritmus se násobí délkou řetězce
- v každém vrcholu se rozhodujeme jaký je momentální character

- každý vrchol má vlastně slovo (prefix) jednoznačně dané
- ne všechny slova končí až v listech (jen prefix)
  - každý vrchol má bool jestli to je slovo nebo ne
  - popřípad i hodnota (slovník)
- ve vrcholu je pole indexované abecedou s ukazateli
- `Member (y)`
  - jestli je řetězec ve stromu
  - hledá postupně a pokud tam je a je i označené jako slovo
- `Insert (y)`
  - jako hledání a pokud třeba, tak přidám a na konci označím vrchol
- `Delete (y)`
  - nejdřív najde, smažu značku a pak pokud je slepá větev, tak smažu vrchol
- všechny časy jsou $\Theta(|y|)$
- paměť je $O(\sum_i |y_i|)$
- Velká abeceda?
  - `Insert (y)`, `Delete (y)` a paměť se zvětší
  - proto je lepší použít BVS
    - paměť pak bude stejná a čas bude $\Theta(|y| \log n)$
- lze upravit na číslicový strom

Hešování

- abstrakce
  - m přihrádek a pak přes funkci dostanu kam to patří
- kolize
  - více prvků v přihrádce
  - lze spravit daním seznamem do přihrádky
- čas závisí na zaplnění přihrádek
  - lepší rovnoměrně rozházet
- Volba hešovací funkce:
  - lineární kongruence
    - $x \to (ax) \mod m$
    - m je prvočíslo a a je nesoudělné s m ($a \approx 0.618$)
  - multiply shift
    - $x \to (ax \mod 2^w) >> w - k$
    - vyříznutí bitů nahoře
  - skalární součin
    - $x_0 \dots x_n \to (\sum_i^n a_i x_i) \mod m$
  - polynom
    - $x_0 \dots x_n \to (\sum_i^n a^i x_i) \mod m$
- ze začátku se nevím jak dlouhý bude vstup a tedy i tabulka
  - proto je dobré přehešovávat v průběhu
  - sledujeme $\alpha = \frac{n}{m}$
    - m je faktor naplnění a n je konstanta
  - pokud α zroste příliš, tak uděláme víe přihrádek a přehešujeme

        * zvolíme dvojnásobek počtu přihrádek
        * časově je to lineární, ale amortizovaně je to konst

- je dobré volit náhodnou hešovací funkci z nějakého systému funkcí
  - *zajímavost: security (PHP)*
- **Df:** Systém funkcí z universa do [m] je c-universální (c>0), tak že $\forall x \neq y \in U : Pr_{h \in H}[h(x) = h(y)] \leq \frac{c}{m}$.

Přednáška 11

- **Lemma:** Nechť H je c-universální systém, $x_0 \dots x_n \in U$ navzájem různé a $y \in U$. Potom $E_{h \in H}[\#i : h(x_i) = h(y)] \leq c\frac{n}{m} + 1$.
  - pokud y se nerovná žádnému x_i tak bude menší jak zlomek a pokud se bude rovnat tak jen jednou proto + 1
  - pomocí indikátorů a střední hodnoty
- otevřená adresace
  - v každé přihrádce může být max 1 prvek
  - kolize - zkusíme jinou přihrádku
  - **Df:** Každému z z universa určíme posloupnost h(x,0), h(x,1), ... , h(x, m-1)
  - `Insert`
    * pokud je přihrádka obsazená, tak přiřazuji na další funkci
    * pokud narazí na pomníček, tak přepíše
  - `Find`
    * postupně hledat v posloupnosti
  - `Delete`
    * místo prvku vložím pomníček
  - příklady:
    * lineární přidávání
      · pokud už je přihrádka obsazená, tak půjdu doprava (cyklicky)
    * dvojité hešování
      · h(x, i) = (f(x) + ig(x)) mod m
  - **Věta:** Pokud vyhledávací posloupnosti jsou nezávislé plně náhodné permutace, potom $E[\#\text{přihrádek navštívených při neúspěšném } Findu] \leq \frac{1}{1-\alpha}$.

Rozděl a panuj

- typicky rekurzivní řešení
- příklad
  - mergesort
    * rozdělení na podposloupnost a slít dané podposloupnosti
- analýza složitosti
  - strom rekurze
  - rozepsání rekurze

Přednáška 12

- řešení násobení čísel o délce n
  - každá cifra s každou (kvadratická)
    - dobré pro malé čísla
  - eozdělení čísel na poloviny a prosčítat mezi sebou
  - 4 násobení n/2 čísel a n na spojení
    - pořád kvadraticky
    - mohu náobit jen třikrát
  - volba základu soustavy
- obecná rekurence
  - každý vrchol má a synů
  - na i-té hladině:
    - a^i podproblémů velikosti $\frac{n}{b^i}$
    - čas na pp $(\frac{n}{b^i})^c$
  - součet pro složitost je geometrická řada, záleží na velikosti q

Master theorem (kuchařková věta)

- pro řešení rekurence
- Rekurence $T(n) = aT(n/b) + \Theta(n^c)$, pro $a \geq 1, b > 1, c \geq 0$, má řešení:
  - $\frac{a}{b^c} > 1 \rightarrow T(n^{\log_b a})$
  - $\frac{a}{b^c} = 1 \rightarrow T(n^c \log n)$
  - $\frac{a}{b^c} < 1 \rightarrow T(n^c)$
- násobení matic
  - lze upravit na 8 násobení a pak sčítání stejnš je to kubické
  - Strassenův alg. jen 7
- selekce
  - hledání k-tého nejmenšího prvku

QuickSelect

- podobné jako quicksort, ale netřídíme vše, jen tam kde může být
- dost záleží na volbě pivota
  - ideálně medián (lineární slož.)
  - nejhůře extrém (kvadratická slož.)
  - možné je skoromedián (v prostřední polovině)
    - randomizace volby
    - **Věta:** $E[$čas *složitost*$] = \Theta(n)$
      - Pravděpodobnost, že se trefím je 1/2.
      - **Lemma (o džbánu):** Pokud pokus uspěje s pravděpodobností p, pak $E[\#$ *pokus*ů *do* 1. ú*spěchu*$] = 1/p$.

Přednáška 13

- volba pivota
  - rozdělení na pětice
  - najdeme mediány pětic
  - najdeme medián mediánu pětic
  - tento alg. má lineární složitost

QuickSort

- vybrat pivota, rozdělit podle pivota a setřiď další části
- **Věta:** QuickSort s náhodnou volbou pivota má průměrnou časovou složitost $\Theta(n \log n)$.
  - intuitivní postup jako předchozí
  - nebo přes střední hodnoty a potom jak se sčítá Harmonická řada
    - ten se dá odhadnout integrálem (pak je to $\log n$)

Dynamické programování

- dá se popsat jako pattern úvah
- FIbonacciho číslo
  - dá se řešit rekurzivně
    - velký strom rekurze a roste exponenciálně
  - opakujeme stejné výpočty dokola
    - proto si pořídíme paměť (keš) na pamatování mezivýsledků
    - v tento moment už je lineární
  - lze ale vyplňovat pole bez rekurze
- pamatování mezivýsledků (**memoizace / kešování**)
- **Postup dyn. prog:**
  1. rekurzivní řešení
  2. mockrát počítám stejnou hodnotu
  3. memoizace / kešování
  4. rekurze předevedeme na plnění keše cyklem

Přednáška 14

- Editační vzdálenost (Levenštejnova)
  - editační operace
    - přidáníznaku, výměna znaku a smazání znaku
  - editačnívzdálenost je pak minimální posloupnost editačních operací
  - je to metrika
  - lze rekurzivně zkoušet všechny možnosti operací
    - je ale hodně pomalé
    - dá se pamatovat mezivýsledky
  - pokud bychom chtěli vyplmňovat tabulku
    - tabulka bude dvojrozměrná a začínám od konce po řádcích
  - lze se na to podívat grafově (hledání nejkratší cesty)

Optimální BVS

- v nějaký moment je lepší upravit BVS s ohledem na dotazy
- optimální strom se dá spočítat rekurzivně
  - vezme se nějakývrchol jako kořen a pak se bude počítat podstromy a přidá se k tomu součet všech vah (protože přes ty se musí projít)
- pokud chceme rekunstruovat strom, tak si pamatujeme optimální kořen

Obecn Dyn. Prog.

- systém podproblémů (stavy DP) a závislosti mezi nimi
  - tvoří DAG
  - procházíme stavy v topologickém pořadí

Floydův-Marshallův algoritmus

- pro hledání nejkratších cest z i do j pro všechny vrcholy v orientovaném grafu
  - lze i n * Dijkstrův algoritmus a nebo n * Bellman Forduv
- v každém kroku mám hodnoty pro všechny vrcholy a podmnožinu vrcholů, přes které může vést sled
- potom přidám dalšívrchol do podmnožiny a vyberu si minimum buď z předchozího ij a nebo součtu předchozího ik a kj