Mílův stručný souhrn

algoritmů od Dvořáka a Töpfera

Co já vím jak je to helpful? Já si to tak jen opakuju na zkoušky

Programování I (NPRG030)

1. ročník Bc. studia MFF UK - Informatika

Autor: Milan Veselý

Dobře no, možná je tohle formátování úplně zbytečný xd

Sylabus

- algoritmus, časová a paměťová složitost
- asymptotická složitost algoritmů
- složitost problému
- dělitelnost čísel, Eukleidův algoritmus
- test prvočíselnosti, Eratosthenovo síto
- rozklad čísla na cifry
- aritmetika s vyšší přesností („dlouhá čísla“)
- Hornerovo schéma, poziční číselné soustavy
- algoritmy vyhledávání v poli (sekvenční, binární, zarážka)
- třídění čísel v poli - přímé metody, heapsort, quicksort, mergesort
- složitost problému třídění
- třídění počítáním, přihrádkové třídění, radixsort
- vnější třídění
- zásobník, fronta, slovník, halda
- spojový seznam
- rekurze – princip, příklady, efektivita
- binární a obecný strom – reprezentace, průchod stromem, použití
- binární vyhledávací strom, princip vyvažování
- hešovací tabulka
- notace aritmetického výrazu – vyhodnocení, převody
- reprezentace grafu
- průchod grafu do hloubky a do šířky, základní grafové algoritmy
- rekurzivní generování
- prohledávání stavového prostoru do hloubky a do šířky
- metody zrychlení backtrackingu – ořezávání, heuristiky
- programování her, algoritmus minimaxu
- metoda Rozděl a panuj

Materiály

Prezentace

Moje poznámky + moje úkolu u Holana

Knihy

http://ksp.mff.cuni.cz/kucharky/

Obsah

Algoritmus

prezentace 1[1]

Popis řešení problému - Konečnost, částečná správnost = správnost

A další věci, stejně se na to nikdo nebude ptát (hromadnost, jednoznačnost, …)

Efektivita – závislá na vstupu

Časová (kroky výpočtu), prostorová

Nejlepší případ, nejhorší případ, „„

Asymptotická notace (pro časovou složitost)

prezentace 2

O(g(n))

f(n) je třídy O(g(n)), když $0 \leq f(n) \leq c \cdot g(n)$ pro každé $n \geq n_0$

neboli pokud ji od určitého bodu můžeme shora odhadnout $c \cdot g(n)$

Ω g(n))

f(n) je třídy Ω(g(n)), když $0 \leq c \cdot g(n) \leq f(n)$ pro každé $n \geq n_0$

analogicky odhad zdola

Θ(g(n))

Pokud je funkcí O(g(n)) i Ω(g(n))
může se lišit multiplikativní konstanta c

Formálně se píše, že funkce patří do O(g(n)), ale v praxi se píše rovná se

Spektrum

Konstantní, polynomiální (lineární, kvadratické), exponenciální

Měření délky vstupu

Matice, grafy, … (řád matice, počet vrcholů, hran, …)
Přirozené číslo je dolů zaokrouhlený $\log_2 + 1$ (protože počet bitů)

Numerické algoritmy

Prvočísla, Erastothenovo síto, NSD…

Vylepšení, složitost, …

Hornerovo schéma, Rychlé umocňování (Převod exponentu do binárky), …

[1] Prezentace vždy končí tam, kde začíná další

Jo a nepíšu sem kód, ten je v prezentacích… vlastně je tohle spíš přehled než cokoliv jinýho

Třídění

prezentace 3

BubbleSort

Projdi pole a vyměňuj dvojice sousedních prvků

Dva vnořené for cykly

Vylepšení:

- Pokračovat jen na pozici poslední výměny a pak skončit

$\Theta(n^2)$

SelectionSort

Najdi minimální a dej ho na první místo

$\Theta(n^2)$

- Ale lepší než Bubble na málo prvcích
- Navíc jen n-1 výměn

InsertionSort

Vzít prvek a zařadit ho mezi ostatní (průchod zprava)

$\Theta(n^2)$

- Opět je na malý rozsah lepší než Bubble
- Oproti Selection je výhodný pro částečně setříděné
  (průměrně provede polovinu srovnání)

HeapSort

Definice haldy, grafů, …

- Min-halda (nahoře min), max-halda
- Ukládá se do pole po hladinách: rodič i má děti 2i+1 a 2i+2
  - Nebo do LS seznamu

Operace

- Přidej a OdeberMin
- Oboje jde v O(log n)

Z n prvků postav haldu a n-krát odeber minimum

Celou haldu postavím v O(n log n) *= O(n log n) + O(n log n)*

- A k tomu konstantní pracovní paměť

MergeSort

prezentace 4

- Spojování setříděných polí (zvládnu lineárně)
- Začít na jednoprvkových a postupně spojovat všechny úseky O(n log n)
    - Konstantní prostor O(n) (potřebujeme jedno pomocné pole)
- Vnější třídění – setřídit data, která se nevejdou do RAMA
    - Minimalizace I/O operací
    - Sloučení rozdělování a slevání
- Varianta TimSort je defaultní v Python
    - InsertionSort a MergeSort

Dolní odhad časové složitosti třídění

- Výše zmíněné jsou porovnávací algoritmy
    - Porovnávají dvojice prvků
    - Nevyužívají hodnoty
- Můžeme u nich vytvořit rozhodovací strom
    - Reprezentuje průběh porovnání
    - Má alespoň n! listů
    - Délka cesty, ...

Třídění v lineárním čase

- Counting sort
    - Spočítá počet výskytů dané hodnoty
    - O(n+k)
        - Ale paměť Θ(k)
    - Nefunguje při strukturovaných datech -> Přihrádkově
- Přihrádkové třídění
    - Klíče a satelitní data
    - Používá se kumulovaná četnost (místo 0,2,1,1,3 je 0,2,3,4,7)
    - Opět O(n+k)
- Radix sort
    - Třídění podle i-té položky klíče (např. víceciferná čísla)
    - Dobrá rychlost, špatný prostor
    - Složitost O(d(n+k))

Vyhledávání v poli

prezentace 5

Sekvenčně x binární vyhledávání

Definice pole (array)

Posloupnost položek stejného typu za sebou v paměti

ALE seznam (list) v Python má libovolný typ (odkaz)

Obě mají přístup k prvku v čase O(1)

Operace v poli

Operátor in
Metody index(), count()

Obě mohou být Θ(n)

del a[i] – Θ(n - i) kvůli posunu

append(x) – Θ(n) nejhorší případ (relokace), ale amortizovaně (1)

Druhy vyhledávání

Sekvenčně

Hledání se zarážkou

Občas bývá efektivnější

Binárně

V uspořádaném poli můžeme půlit intervaly

Θ(log n)

Výpočet odmocniny půlením intervalu

Půlím interval a zkusím, jestli je to na druhou epsilon daleko od hledaného

Halda v lineárním čase

prezentace 6

Vytvoříme pole a začneme ho opravovat do tvaru haldy

Budeme opravovat od prvního rodiče

V i-té hladině je 2^i vrcholů -> pro každý nejvýše h-i výměn

$\sum_{i=0}^{h-1} 2^i(h-i) = 2^h \sum_{i=0}^{h-1} \frac{j}{2^j}$ ale to je n*konstanta… důkaz v prezentaci

Máme tedy operace:

MakeHeap - O(n), Přidej - (log N), OdeberMax - O(log N) a Max - O(1)

Další operace: ZvýšeníKlíče a Odstraň

Abstraktní datové typy

Prezentace 7

Různé implementace – pole, lineárně spojový seznam, … (různé třídy)

Přehled: Zásobník, fronta, halda

Zásobník (stack)

„pevné dno“ – přidává se a odebírá se z vrchu

Např. prohledávání do hloubky

Append opět může způsobit relokaci -> O(n)

Fronta (que)

Přidává se na konec, odebírá ze začátku

Prohledávání do šířky, čekající procesy, simulace, …

Prioritní fronta

Prvky se v ní podle priorit předbíhají

Implementace:

Zařazujeme nebo vybíráme podle priority v normální frontě

Samostatné sezamy pro priority

Prioritně řazená halda (prvky jako seznamy)

Binární halda (binary heap)

Prvky musí být porovnatelné

Typická implementace v poli

Slovník (dictionary)

Dvojce klíč – hodnota

Složitá efektivní implementace (hashování, …)

Python dokonce přímo podporuje

Rekurze

Prezentace 8

Rekurzivní algoritmus nebo rekurzivní volání funkce

Není nutné aby se rekurzivní algoritmus realizoval rekurzivní funkcí

Musí skončit

Příklady: Eukleidův algoritmus, palindrom, faktoriál, …

Pozor Fibonacciho čísla – exponenciální časová složitost (pro n>40 nepoužitelná)

Memoizace, kešování hodnot, dynamické programování

Nebo počítat odspodu, vzoreček či rychlé umocňování matice

Rekurzivní generování

Prezentace 9

„všechny možnosti“

Např. vypsat k-ciferná čísla v soustavě k

Můžeme to řešit buď k for cykly (pro každou cifru)

A nebo rekurzivní funkcí viz. Prezentace

Kombinace bez opakování

Ostře rostoucí k-tice (předchozí číslo + 1)

Doplnění znamének

Snažíme se vytvořit výraz aby výsledek byl konkrétní číslo

Postupně zkusím dát + a –

Rozklad čísel

Aby se neopakoval tak je nerostoucí

Moje úkoly na rekurzi: Loyd, ATest, Mince, Sklenice

Binární strom

Průchod PREORDER

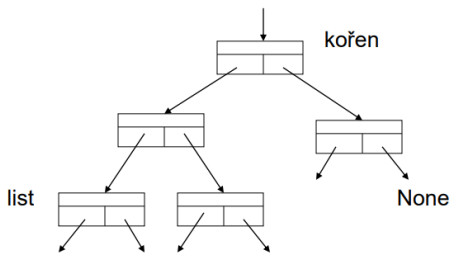
nejdřív vrchol a pak podstromy

Průchod INORDER

nejdřív levý podstrom, pak vrchol, …

Průchod POSTORDER

Nejprve podstromy a pak vrchol



Průchod do hloubky (bez rekurze lze se zásobníkem)

Nejprve konec první větve, krok zpátky a konec druhé větve, …

Průchod do šířky lze (bez rekurze lze s frontou)

Nejprve prvek nalevo, pak napravo, pak dítě toho nalevo, …

Vždy je průchod O(n)

Příklady použití binárního stromu

Binární vyhledávací strom
Aritmetický strom

Obecný strom

Reprezentace

a) Známe maximální stupeň větvení
b) Obecně

V každém uzlu je seznam odkazů

c) Kanonicky

Spíše zajímavost

Binární strom syna a bratrů

Příklady použití

Písmenový strom, který ukládá slova (to btw můžeme i hešováním)

Grafy[2]

Prezentace 10

Pojmy: Vrcholy, hrany, neorientovaný, nehodnocený graf, graf x jeho nakreslení, …
Počet hran v grafu (min, max, …), Cesty v grafu (sled, tah cesta, vzdálenost, …)
Souvislost grafu – existuje cesta (v orientovaném grafu slabě a silně)
Strom, les, kostra, acyklický, bipartitní, topologické uspořádání

Reprezentace

Matice sousednosti (pro malé grafy s velkým počtem hran)

Matice n × n, 1/0 podle toho jestli existuje hrana

Seznam následníků (Primárně pro orientované grafy)

a) Pole n seznamů
b) Matice n × n – 1 + list s počtem stran vrcholů
c) Matice n × r (když známe maximum hran pro jeden vrchol)
d) Víme, že v grafu je nejvýše M hran

[2] Nebudu se věnovat vůbec pojmům, protože jsou z diskrétky (a taky v prezentaci)

Seznam hran

Jednorozměrné pole nebo spojový seznam

Matici incidence (velmi řídké grafy)

Matice zda vrchol leží na hraně

Dynamická reprezentace

Není typická

Rozdíl od stromu je v souvislosti a cykličnosti

Je potřeba tabulka odkazů na uzly a detekce cyklení

Grafové problémy

1. Je graf souvislý? Určit, které vrcholy grafy jsou dostupné z daného vrcholu

   Průchod s počítadlem vrcholů
   Pokud má počítadlo hodnotu N ✓

   Dostupnost uděláme bool seznamem navštíven

2. Určit komponenty souvislosti grafu

   Opakovaný průchod grafem podle seznamu navštíven

3. Obsahuje graf nějaký cyklus? Jinými slovy: Je graf stromem nebo lesem?

   Průchod, ale dávám pozor, jestli byl vrchol navštíven -> cyklus

4. Určit kostru souvislého grafu (jednu libovolnou)

   Analogicky jako 3. (Když narazíme na nenavštívený vrchol -> kostra)

5. Určit, zda je graf bipartitní

   Střídavě vrcholy značíme 1 a -1, narazíme-li na stejný tak konec

Není podstatné zda 1. – 5. řešíme průchodem do hloubky nebo do šířky

6. Určit délku nejkratší cesty mezi danými dvěma vrcholy Pro daný vrchol určit vzdálenosti všech ostatních vrcholů.

   nenavštívené vrcholy -1, výchozí vrchol 0 a sousedé +1
   "algoritmus vlny"

7. Nalézt nejkratší cestu mezi danými dvěma vrcholy (jednu libovolnou)
   a) Stejně jako 6, ale navíc si pro každý vrchol značíme předchůdce
   b) Stejně jako 6, ale se zpětným chodem (rekonstrukce cesty odzadu)

Tyto řešíme pouze do šířky

Prohledávání stavového prostoru

Prezentace 11

Do hloubky

Nebo-li depth-first search, backtracking, …

Z výchozího stavu zkusíme všechny možné pokračování stylem, že pokud je cesta neúspěšná, tak se o krok vrátíme

Pokud najdeme řešení resp. projdeme celý strom -> konec

Buď rekurzivně nebo se zásobníkem

Osm dam, jezdcova procházka

Paměťová a časová složitost:

- Paměťově stačí výška stromu
- Časově musíme projít všechny možné uzly
  - Exponenciální a proto se zavádí ořezávání a heuristiky
    - Ořezávání průběžně zkouší „smysluplnost“
    - Heuristika se snaží uhodnout co zkusit prvně

Algoritmus minimaxu

- Hodně podrobně to mám v zápočťáku (v dokumentaci)
- Hry dvou hráčů s úplnou informací a střídání tahů
- Strom hry…
- Malé hry
  - Lze postavit celý strom
  - Ohodnotí listy a střídavě vybírá nejlepší tah (minimax)
- Velké hry:
  - Postavit pouze částečně (určitá hloubka)
  - Alfa-beta prořezávání

Do šířky

Prezentace 12

Breadth-first search, algoritmus vlny

Velmi paměťově náročné – nutnost si pamatovat všechny pozice

Na druhou stranu díky tomu můžeme najít nejkratší řešení

Obdobné jako u grafů (použijeme zásobník)

Nejkratší cesta koněm na šachovnici

Rozděl a panuj

Metoda rekurzivního návrhu

Problém se rozdělí na dva podproblémy stejného typu jako hlavní problém

Podmínka: nemůžou být závislé -> opakování výpočtu (např. Fibonacci)

Vyhodnocení aritmetického výrazu

- Najdeme znaménko, které se vyhodnotí jako poslední
- Vyhodnotíme obě strany od znaménka a pak provedeme operaci
- Nejhorší případ $O(n^2)$ ale průměrný $O(n \,.\, \log n)$

Hanojské věže

- Hlavní funkce přenes N disků z A na B pomocí C (PŘENES(N, A, B, C)):
  - PŘENES(N-1, A, C, B), přenes disk z A na B, PŘENES(N-1, C, B, A)

MergeSort

Rozdělení na dvě části, buď je triviální a nebo spojíme dvě setříděné posloupnosti ↺

Dva úseky spojíme v n a úseků je log n -> $O(n \cdot \log n)$

Nevýhoda je, že máme paměť na každé hladině a tedy $O(n \cdot \log n)$

- Dá se alternativně naimplementovat ve dvou seznamech $O(n)$
- Do druhého pole měníme to co aktuálně spojujeme a zbytek necháme

Quick sort

Náhodně vybereme prvek -> pivot

Prvky řadíme doprava a doleva podle toho jestli jsou větší nebo menší

Opět stejný problém s pamětí

- Vyměňování prvků zleva a zprava dokud se ukazatele nepotkají

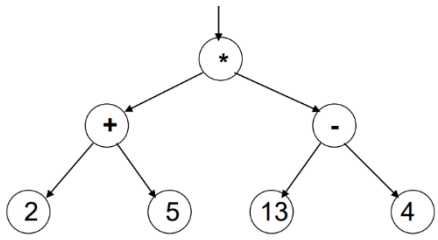
Časově v nejhorším případě $O(n^2)$, ale průměrně $O(n \cdot \log n)$

- a má nejmenší multiplikativní konstantu a tedy se často používá
- nejhorší se dá opravit vybráním mediánu a nebo průběžnou kontrolou
  - ale zhorší se konstanta

Quicksort lze řešit i bez rekurze a to zásobníkem

- Ušetří se režie rekurzivních volání a stačí zásobník log výšky

Reprezentace aritmetického výrazu



[Prezentace 13]

Binární strom

Vyhodnocování od kořene:

self.levy "operace" self.pravy

Notace:

PREFIX	* + 2 5 – 13 4
INFIX	2 + 5 * 13 – 4 (musí být bez závorek)
POSTFIX	2 5 + 13 4 - *

Z jediného infixu není zřejmé jak výraz vypadá

Vyhodnocování:

V postfixu je to snadné

Průchod zleva doprava + zásobník na hodnoty
(Pozor na to, že na vrchu zásobníku je pravý operand)
Čas O(n)

V prefixu

Jako postfix ale odzadu
Nebo zleva ale zásobník na ukládání znamének
Nebo rekurze s globálním indexem
když je to číslo, tak vrátí hodnotu
když je to znaménko, tak se dvakrát zavolá a pak spočítá

Převod Infix (se závorkami) -> postfix

Číslo:	zapsat na výstup
Levá závorka:	zapsat do zásobníku
Pravá závorka:	na výstup vložit všechno po poslední levé
Znaménko:	vložit do zásobníku, před tím přenést na výstup určitá znaménka

Postavení stromu ze zápisu

Analogicky jako vyhodnocování
Do zásobníku se ukládá odkaz na uzel a tomu se přiřazují synové

Nalezení k-tého nejmenšího prvku

1. Setřídit $O(n \cdot \log n)$
2. Postavit haldu a k-krát odebrat minimum $O(n + k \cdot \log n)$
3. Postavit haldu z n-k+2
   Pak k-2 krát odebrat minimum a místo toho přidat z těch bokem
   Odeberme dvakrát minimum – (k-1). nejmenší a k. nejmenší
   Např pro medián je halda poloviční (o jednu hladinu méně)
4. QuickSelect
   Po rozdělení pivotem pracujeme jen s vhodnou částí (podle délky)
   Nejhůře se provede celý QuickSort a tedy $O(n^2)$
   Průměrně ale O(N)
   Ještě se to zlepší s inteligentní volbou mediánu

Vyhledávací stromy

Prezentace 14

Datová struktura pro vyhledávání podle klíče

Napravo mají větší klíč, nalevo menší

Není potřeba projít celý strom ale pouze cestu k listu

Výška stromu, maximálně n, průměrně $\log_2 n$

Hledání

Buď jdeme doprava nebo doleva podle velikosti

Přidávání

Přidává se do listu

Odebírání

a) Pokud nemá následovníky -> ten před ním na None
b) Pokud má jednoho následovníka, tak se akorát přepojí
c) Pokud má dva následníky, tak ji nahradím maximem zleva a nebo minimem zprava, původní vrchol už smazat umím

Hledání se zarážkou (nepříliš praktické)

Obdobně jako v poli, všechny levy a pravy s None nahradíme hledanou

Vyvážené stromy

Cíl zajistit výšku O(log n)

Dokonale vyvážený strom

Rozdíl levého a pravého podstromu nejvýše jedna

Obtížné ho udržet

AVL strom

AVL podle jmen

Výška jeho levého a pravého podstromu se liší max o 1

Je maximálně o cca. 45% vyšší než dokonale vyvážený strom

Stavba vyvážených stromů

Dokonale (se setříděným seznamem):

1. Vytvořím prázdný strom a pak postupně přepisují
2. Psát je při vytváření a používat indexy

AVL:

Každý vrchol má položku balance

Určuje jak se liší výška L a P

Pomocí ní přidávání a odebírání v O(log n)

Přidávání:

Mají-li všechny vrcholy na cestě balance 0

Přidám a změním u nich balance na 1/-1

Existuje na cestě alespoň jeden nenulový

Před tím nejbližším všechny 0 přepíšeme

Buď pivot bude 0 a tedy konec a nebo 2/-2

➔ Rotace (nevysvětlovali jsme si)

Odebírání je složitější, protože musíme projít celou cestu