5, 6, 7, 8, 9

Obecná 01

Amortizovaná složitost

- průměrný čas na vykonání operace v sekvenci operací v nejhorším případě
- nevyužívá pravděpodobnost -> průměrný čas na operaci je skutečně zaručený
- asymptotická složitost:
    - porovnání efektivity a rychlosti algoritmů
    - horní asymptotický odhad
    - dolní asymptotický odhad
    - optimální asymptotický odhad

Prioritní fronta

- abstraktní datový typ
- každý element má přiřazenu svou prioritu
- z fronty jsou vybrány dříve elementy s nejnižší (nejvyšší) prioritou
- operace:
    - void push(Element e) - vloží element s prioritou
    - Element pull() - odebere element s nejnižší (nejvyšší) prioritou

Binární halda

- implementace prioritní fronty
- stromová struktura
- pro všechny prvky platí pravidlo: pokud A je potomek B, pak B <= A
- operace:
    - insert(x)
        - O(log n)
        - 1. přidáme prvek na konec haldy
        - 2. dokud je větší než rodič, tak prohazovat
    - delete(x)
        - O(log n)
        - 1.
    - merge(H1, H2)
        - O(m + n)
    - accessMin()
        - O(1)
    - deleteMin()
        - O(log n)
    - decrease(x, value)
        - O(log n)
- obrázek reprezentace binární haldy v paměti

D-regulární halda

- d udává stupeň štěpení
- pro d = 2 je halda binární halda
- operace a jejich složitost je analogická s binární haldou
- přesná složitost se liší základem logaritmu (základ je d)

Binomiální halda

- množina binomiálních stromů řádu 1, …, log(n)
- každý řád je zastoupený maximálně jedním stromem
- každý vrchol je menší nebo roven svým potomkům
- má 2^i vrcholů
- má hloubku i
- jeho kořen má i synů
- strom řádu i vznikne ze dvou stromů řádu i-1
- operace:
    - insert - amortizovaná složitost je konstantní
    - min - konstantní
    - merge - O(log n)

Fibonacciho halda

Obecná 02

Neorientované a orientované grafy

- graf
  - základní objekt teorie grafů
  - reprezentace množiny objektů, které mohou být propojeny
  - dvojice G = <V, E>
  - vrcholy, spojují je hrany
  - V je neprázdná množina vrcholů
- neorientovaný
  - hrana je dvouprvková množina {u, v}: $E \subseteq \{\{u, v\} \mid u,v \in V, u \neq v\}$
  - na pořadí vrcholů nezáleží
- orientovaný
  - hrana je uspořádaná dvojice <u, v>
  - na pořadí vrcholů záleží

Reprezentace grafu

- matice sousednosti
- matice incidence
- matice vzdáleností
- seznam sousedů

Prohledávání

- do hloubky (DFS)
  - nejdříve se prohledávají potomci, potom sourozenci
  - implementován pomocí
    - zásobníku (LIFO)
    - rekurze (paměťově náročnější)
  - algoritmus:

```
Stack to_visit = empty;
Vertices visited = empty;
to_visit.push(v);
while (!to_visit.isEmpty()) {
    v = to_visit.pop();
    if (visited.contains(v)) {
        continue;
    }
    visited.add(v);
    for (n in v.neighbors()) {
```

```
                        to_visit.push(n);
                    }
                }
```

- do šířky (BFS)
    - nejdříve se prohledávají sourozenci, pak potomci
    - implementace stejná, pouze se místo zásobníku použije fronta (FIFO)

Topologické uspořádání

- taková posloupnost uzlů u_i, u_j, že pro každou hranu mezi dvěma uzly platí $i < j$
- z toho vychází podmínka acykličnosti grafu
- používá se pro plánování na sobě závislých činností
- algoritmus topologického uspořádání vychází z procházení grafu do hloubky (pořadí uzavření uzlů v opačně orientovaném grafu)

Souvislost

- pro každé dva vrcholy x, y existuje cesta z x do y

Strom

- souvislý graf bez cyklů
- přidání libovolné hrany vznikne cyklus
- odebrání libovolné hrany přestane být souvislý
- má |V| - 1 hran
- každé dva vrcholy jsou spojeny pouze jednou cestou
- list = uzel, který nemá žádné potomky
- kořen = uzel, který není potomkem

Minimální kostra

- kostra H grafu G je podgraf takový, že V(G) = V(H)
- minimální kostra je kostra, která má minimální sumu vah svých hran
- algoritmy:
    - Jarníkův-Primův O(|V(G)| * |E(G)|)
        - z libovolného uzlu rozšiřuji minimální kostru
        - označím potomky a z nich připojím nejmenší hranu
    - Borůvkův O(log |V(G)| * |E(G)|)
        - na začátku jsou všechny uzly samostatné komponenty
        - ke každé komponentě přidám nejmenší hranu
    - Kruskalův O(log |V(G)| * |E(G)|)

Obecná 03

Lexikální analyzátor

- obvykle je to stavový automat
- na vstupu dostává text
- generuje lexikální symboly
  - jsou terminálními pro syntaktickou analýzu
  - popsány regulárními výrazy nebo gramatikou
  - mohou mít atributy (např. hodnotu proměnné)
- ignoruje některé části textu (bílé znaky, komentáře, …)
- nestará se o smysl výstupních konstrukcí
- obvykle implementován pomocí deterministického konečného automatu

Syntaktický strom

- výstup překladače u interpretovaných jazyků
- vnitřní uzly jsou operátory
- listy jsou operandy
- používá se k optimalizaci kódu

Syntaktický analyzátor shora dolů

- = parsování
- bere výstup z lexikálního analyzátoru
- kontroluje správnost podle LL(1) gramatiky
- realizuje se zásobníkovým automatem
  - na začátku tam je neterminál
  - v každém kroku se rozvine podle gramatiky
  - když nejde dál rozvinout (je tam terminál), podívá se na vstup a porovná ho
  - když je zásobník prázdný, slovo bylo přijato

LL(1) gramatiky

- deterministické zpracování
- k rozhodnutí stačí znát jeden následující symbol

Rozkladové tabulky

- pro rozvinutí neterminálu v syntaktické analýze
- tabulka neterminálů na terminály
- výpočet pomocí FIRST (a pokud je prázdný, tak i podle FOLLOW)

- FIRST
- FOLLOW

Obecná 04

- definice konečného automatu
- regulární jazyk
- vztah konečného automatu a regulárního jazyka

Algoritmy vyhledávání v textu s lineární a sublineární složitostí

- naivní algoritmus
- Boyer-Moore
  - tabulka GSS (Good Suffix Shift)

Využití konečných automatů pro přesné a přibližné hledání v textu

- determinisitcký konečný automat
- nedeterministický konečný automat
- Hammingova vzdálenost
- Levenshteinova vzdálenost

Obecná 05

Algoritmus

- = dobře definovaný proces (posloupnost výpočetních kroků), který zpracuje vstup a vydá výstup
- algoritmus A řeší úlohu U, pokud pro každý vstup vydá správné řešení

Správnost algoritmu

- algoritmus se musí zastavit
- po zastavení vydá algoritmus správný výstup
- variant
  - = přirozené číslo, které se s každým proběhnutím cyklu zmenšuje až dosáhne své minimální hodnoty
  - zaručuje ukončení algoritmu v konečném počtu kroků
- invariant
  - = tvrzení, které platí před vstupem do cyklu

  - pokud platí před vykonáním cyklu, platí i po každém provedení cyklu
  - zaručuje správnost řešení

Složitost algoritmu

- master theorem
- omega, omikron, theta

Složitost úlohy

- převod úloh

Třída P a NP

Rozhodovací úloha

- její řešení je buď ano nebo ne
- často v sobě obsahují konstrukční úlohu (např. pokud chci zjisti, zda existuje maximální tok, musím ho najít)
- každou úlohu lze zakódovat pomocí vhodné abecedy do slov

Turingův stroj

- sedmice TS = (Q, E, T, q0, d, B, F)
- TS rozhodne jazyk L, pokud pro každé slovo w náleží E se v konečném počtu kroků zastaví

Nedeterministický algoritmus

Třída P

- rozhodovací úloha U patří do třídy P, pokud existuje TS, který rozhodne jazyk L_u v polynomiálním čase
- příklady:
  - existuje kostra grafu ceny menší nebo rovno C?
  - existuje orientovaná cesta v acyklickém grafu z vrcholu a do b menší než d?
  - existuje přípustný tok alespoň velikosti k?
  - existuje řez, který má kapacitu menší nebo rovnu k?

Třída NP

- rozhodovací úloha U leží ve třídě NP, pokud existuje nedeterministický TS, který rozhodne jazyk L_u v polynomiálním čase

Obecná 06

NP-úplné úlohy

- úloha U je ve třídě NP
- všechny NP úlohy se polynomiálně redukují na U

NP-těžké úlohy

- některá NPC úloha se polynomiálně redukuje na U

Cookeova věta

- = úloha SAT je NP-úplná
- = libovolný nedeterministický Turingův stroj lze v polynomiálním čase převést na problém splnitelnosti booleovských formulí v konjunktivním normálním tvaru
- SAT je splňování formulí v konjunktivním normálním tvaru
- důkaz:
    - úloha SAT je ve třídě NP
    - nedeterministický algoritmus vygeneruje ohodnocení logických proměnných
    - v polynomiálním čase lze ověřit, jestli je formule v daném ohodnocení pravdivá, či ne

Heurisitiky na řešení NP-těžkých úloh

- 2-aproximační algoritmus
    - jestliže instance I splňuje trojúhelníkovou nerovnost, pak existuje polynomiální algoritmus, který pro I najde trasu délky D, kde D <= 2 OPT(I)
    - postup:
        - instanci I považujeme za úplný graf G
        - v G najdeme minimální kostru
        - kostru prohledáme do hloubky
        - zapíšeme první výskyt každého uzlu jako trasu
- Christofidesův algoritmus
    - jestliže instance I splňuje trojúhelníkovou nerovnost, pak existuje polynomiální algoritmus, který pro I najde trasu délky D, kde D <= 3/2 OPT(I)
    - postup:
        - instanci I považujeme za úplný graf G
        - v G najdeme minimální kostru
        - vytvoříme úplný graf H pouze s vrcholy, které mají v kostře lichý stupeň
        - v H najdeme nejlevnější perfektní párování P
        - hrany z P přidáme do kostry
        - sestrojíme eulerovský tah a zapíšeme vrcholy trasy

Pravděpodobnostní algoritmy

- randomizovaný Turingův stroj
    - TS se dvěma nebo více páskami
    - první páska má stejnou roli jako u deterministického TS
    - druhá páska obsahuje náhodnou posloupnost 0 a 1
- třída RP
    - jazyk L patří do RP, pokud existuje RTS takový, že:
        1. jestli w nepatří do L, pak se RTS zastaví v F s pravděpodobností 0
        2. jestli w patří do L, pak se RTS zastaví v F s pravděpodobností 0,5
        3. každý běh RTS trvá maximálně p(n) kroků
    - Millerův test prvočíselnosti
    - Turingův stroj Monte-Carlo
        1. splňuje podmínky 1 a 2
        2. nemusí pracovat v polynomiálním čase
- třída ZPP
    - jazyk L patří do třídy ZPP, pokud existuje RTS takový, že:
        1. jestli w nepatří do L, pak RTS zastaví v F s pravděpodobností 0
        2. jestli w patří do L, pak RTS zastaví v F s pravděpodobností 1
        3. střední hodnota počtu kroků RTS v jednom běhu je p(n)
    - tzn. neuděláme chybu, ale nemůžeme zaručit polynomiální běh
    - Turingův stroj Las-Vegas

Obecná 07

Metoda větví a mezí

- z = ILP(A, B, c, z*)
- řešit pomocí LP
- x_i jsou celočíselné -> konec, jinak:
    - rozdělit na dvě podúlohy
    - z’ = ILP(A’, B’, c, z*) pro x_i <= k
    - z’’ = ILP(A’’, B’’, c, z*) pro x_i <= k + 1
- z více řešení vybrat lepší
- neexistuje přípustné řešení ->konec, jinak:
    - jestli z > z*, tak na krok č. 3, jinak konec

Algoritmy pro celočíselné lineární programování

- větve a meze
- metoda sečných nadrovin
    - řeší se LP
    - v každé iteraci se přidá podmínka tak, že:
        - optimální řešení LP se stane nepřípustným

        - žádné celočíselné řešení se nestane nepřípustným
- (batoh, toky)

Formulace optimalizačních a rozhodovacích problémů pomocí celočíselného lineárního programování

- dělení kořisti
- nejkratší cesta v grafu
- problém asymetrického obchodního cestujícího
- logické formule
- alespoň 1 ze 2 podmínek
- metoda větví a mezí

Toky a řezy

- síť = pětice (G, l, u, s, t), kde
    - G je orientovaný graf
    - l je dolní omezení hran
    - u je horní omezení hran
    - s je zdroj a t spotřebič
- tok = ohodnocení hran v síti, kde pro každý vrchol (kromě s, t) platí Kirchhofův zákon
    - přípustný tok: $f_e = <l_e, u_e>$
- problém maximálního toku
    - chceme najít takový přípustný tok,
    - Ford-Fulkrsonův alg. (postupné zlepšování propustnosti toku)
        i. najdi přípustný tok f_e pro všechny e náleží E(G)
        ii. najdi zlepšující cestu; pokud neexistuje, ukonči se
        iii. spočítej kapacitu zlepšující cesty a zlepši tok z s do t, opakuj 2
    - problém minimálního řezu
    - celočíselnost
- rozhodovací problém přípustného toku v síti
- nejlevnější tok v síti

Multikomoditní toky

- nejlevnější multikomoditní toky

Obecná 08

Nejkratší cesty

- Dijkstrův alg.
- Bellman-Fordův alg.

- Floydův alg.

Úloha obchodního cestujícího

- Hamiltonovská kružnice
- důkaz, že je to NP-úplný problém
    - polynomiální redukcí vytvořím instanci TSP tak, že každému vrcholu grafu G odpovídá vrchol v úplném neorientovaném grafu K
    - váha hrany {i, j} v K je:
        - 1, pokud {i, j} náleží E(G)
        - 2, pokud {i, j} nenáleží E(G)
- heuristika Nejbližší soused
- 2-aproximační algoritmus
- 3/2-aproximační algoritmus
    - složitost metrického cestujícího (převodem z HK)
- TSP pomocí ILP

Heuristiky a aproximační algoritmy

- heuristika Nejbližší soused
- 2-aproximační algoritmus
- 3/2-aproximační algoritmus

Metoda dynamického programování

- Rothkopf P||Cmax
- batoh

Problém batohu

- přidání položky
    - jdi na [y + 1, x + cena]
    - nastav na z + váha
- nepřidání položky
    - jdi na [y + 1, x]
    - nastav na zdrojové pole
- při možnosti více řešení se vybere to výhodnější
- O(C * n)
    - lze polynomem omezit délu vstupu n
    - nelze omezit velikost vstupu

Pseudo-polynomiální algoritmy

- pseudopolynomiální algoritmus
- stavový prostor je konstruován díky celočíselným vstupům
- ze dvou řešení si ponecháme to výhodnější

- Rothkopf P||Cmax
- batoh

Obecná 09

Rozvrhování na jednom procesoru

- Bratley's algorithm
- metoda větví a mezí

Rozvrhování na více procesorech

- McNaughtonův
- List scheduling
- Longest Processing Time first
- úrovňový algoritmus P|pmtn,prec|Cmax

Rozvrhování projektu s časovými omezeními

- množina nepreemptivních úloh je reprezentována vrcholy v orientovaném grafu
- každý uzel je ohodnocen p_i (doba vykonání úlohy)
- hrany jsou temporální omezení $l_{i,j}$ ($s_i + l_{i,j} <= s_j$)
- typy:
    - $l_{i,j} = p_i$ … úloha j může začít po dokončení úlohy i
    - $l_{i,j} > p_i$ … úloha j může začít až za nějaký čas po dokončení úlohy i (schnutí laku)
    - $0 < l_{i,j} < p_i$ … dílčí část úlohy i je možné použít pro úlohu j
    - $l_{i,j} = 0$ … úloha i musí začít dříve nebo ve stejný okamžik, jako úloha j
    - $l_{i,j} < 0$ .. úloha i musí začít dříve nebo maximálně o $|l_{i,j}|$ později

Programování s omezujícími podmínkami

- trojice (X, D, C)
- X = {x1, …, xn} … konečná množina proměnných
- D = {D1, …, Dn} … konečná množina domén
    - Di obsahuje možné hodnoty pro xi
- C = {C1, …, Cn} .. konečná množina omezení
- CSP
    - může pokračovat optimalizací (např. větve a meze)
    - možnost složitějších podmínek oproti ILP
- hranová konzistence
- revize hrany