Určení Asymptotické Časové Složitosti

Triviálně z pseudokódu (worst case)

U každého kroku / operace určíme worst-case složitost, odhadneme maximální počet opakování každého cyklu (/ volání procedury), příslušně pronásobíme a dostaneme horní odhad celkové složitosti.

Velmi hrubé

Omezení přes prvky vstupu (worst case)

U některých kroků můžeme odhadnou celkový počet provedení lépe omezením dle velikosti vstupu namísto w.c. počtu projití cyklů.

Příklad: BFS,DFS sáhne na každou hranu pouze $\mathcal{O}(1)$-krát, složitost je tedy $\mathcal{O}(n+m)$ namísto $\mathcal{O}(n^2)$ z triviálního odhadu, že projít hrany kolem vrcholu je obecně $\mathcal{O}(n)$

Vhodné hlavně pokud worst-case a typický případ jsou odlišné (v řídkém grafu může být vrchol s w.c. $\mathcal{O}(n)$ sousedy, ale typický vrchol jich bude mít méně)

Omezení pomocí invariantů (worst case)

Pomocí invariantů (a dalších vlastností) můžeme odhadnout počet provedení některých operací (cyklů) lépe.

Příklad: pokud naprogramujeme relaxační algoritmus, počet relaxací závisí na strategii výběru vrcholů a vlastnostech vstupu. Dle vlastností můžeme dostat $\mathcal{O}(n)$ relaxací (Dijkstra), $\mathcal{O}(nm)$ (Bellman-Ford) nebo $O(exp(n))$ (při nevhodné strategii)

Vhodné obzvláště pokud v úloze máme nějaký typ monotonie nebo uspořádání (např. u dynamických algoritmů, které "procházejí jedním směrem").

Potenciálová metoda (worst case)

Definujeme nezáporný potenciál. Některé operace, jejichž počet provedení umíme odhadnout, zvyšují potenciál. Divoké operace, jejichž počet provedení nelze přímočaře odhadnout, potenciál snižují. Odhadem celkového zvýšení potenciálu za dobu běhu algoritmu, a minimálního možného snížení divokou operací dostaneme odhad na počet provedení divoké operace.

Příklad: (ADS2) Tokové algoritmy - Goldberg/Preflow-push

Amortizacovaná složitost

Metoda umožňující "zprůměrování" složitostí většího množství provedení operací (na datové struktuře)

Definujeme nezáporný potenciál Φ, jeho hodnotu po i-té operaci označíme Φ_i ($\Phi_0 = 0$). Amortizovaná složitost i-té operace je pak definována jako součet skutečného času (worst-case) operace a změny potenciálu

$$A_i = Time_i + \Phi_i - \Phi_{i-1}$$

Po provedení m operací, celková amortizovaná složitost všech operací je součet amortizovaných složitostí

$$\Sigma_{i=1}^{m}(Time_i + \Phi_i - \Phi_{i-1}) = \Sigma_{i=1}^{m}(Time_i) + \Phi_m - \Phi_0 \geq \Sigma_{i=1}^{m} Time_i$$

Snadno vidíme, že celková asymptotická složitost je vždy horní odhad na skutečnou složitost (dokud jsou dodrženy podmínky $\Phi_0 = 0$ a $\Phi_m \geq 0$). Protože ale pro jednotlivé operace může A_i být (asymptoticky) mnohem menší než $Time_i$ když potenciál velmi klesne, můžeme tak dostat mnohem lepší celkový odhad.

Vhodné hlavně pro odhady složitostí datových struktur (především líných)

Příklad: rostoucí pole, líné vyvažování, multi-pop zásobník, (v pokročilých kurzech) Fibonacciho halda, Splay stromy, chování (a, b)-stromů v A-sort algoritmu

Poznámky: U výpočtu hodnot/změn potenciálu jsou důležité konstanty! Je důležité, že DS začíná s potenciálem 0. Nelze použít amortizované odhady na strukturu, která již dříve provedla nějaké neznámé operace (a má potenciálně vysoký potenciál). Pokud nenastane předchozí problém, můžeme amortizovanou složitost používat stejně jako worst-case (protože je horním odhadem skutečné složitosti).

Amortizovaná složitost se často označuje subscriptem "a": $\mathcal{O}_a(f(n))$

Očekávaná ("průměrná") složitost

Lze použít pouze u randomizovaných algoritmů! Očekávanou složitost algoritmu nad vstupy délky n určíme následovně. Pro každý vstup délky n určíme střední hodnotu délky běhu v závislosti na náhodných volbách algoritmu. Ze všech středních hodnot pro jednotlivé vstupy vezmeme maximum.

Formálně: $Alg_{\mathbb{E}}(n) = max_{I:|I|=n}\{\mathbb{E}_R[Alg(I, R)]\}$ Kde maximum jde přes vstupy I velikosti n, a střední hodnota je počítána přes náhodné bity R. $Alg(I, R)$ je doba běhu algoritmu nad fixním vstupem I s fixními náhodnými bity R.

Příklady: hešování, quicksort

Důležité: Nejedná se o průměr přes vstupy! Náhodnost tedy nesmí pocházet ze vstupu! Není možné zafixovat konkrétní hešovací funkci nebo deterministickou volbu pivota v quicksortu. Algoritmy se potom chovají na fixním vstupu vždy stejně, a vniknou "toxické" vstupy, které odhad rozbijí. Klíčové je, že při opakovaném puštění algoritmu nad stejným vstupem dostáváme různé chování jehož typický případ je dobrý. Pokud se nám chování alg. nelíbí (příliš

mnoho kolizí hešovací funkce), spustíme algoritmus znovu (nová hešovací funkce / přehešování).

Očekávaná složitost se často označuje subscriptem se symbolem střední hodnoty: $\mathcal{O}_{\mathbb{E}}(f(n))$. Toto rozlišení je důležité (narozdíl od amortizované vs. worst-case složitosti), očekávaná složitost není horní odhad worst-case složitosti! Vždy existuje riziko překročení odhadu!

Master's Theorem (worst case)

Master's Theorem ("kuchařková věta") je nástroj na řešení rekurzivních vztahů časové složitosti. Existuje v mnoha různých verzích a stupních síly.

Pro předpis rekurzivní složitosti:

$$T(n) = A * T(n/B) + \mathcal{O}(f(n))$$

Lze $T(n)$ určit následovně (předpokládáme $T(1) = \mathcal{O}(1)$)

- "Těžký kořen" pokud $f(n) = \mathcal{O}(n^c)$ a $A < B^c$ potom
$$T(n) = \mathcal{O}(f(n)) = \mathcal{O}(n^c)$$
- "Těžké listy" pokud $f(n) = \mathcal{O}(n^c)$ a $A > B^c$ potom
$$T(n) = \#RecursionLeaves * O(1) = \mathcal{O}(A^{\log_B n}) = \mathcal{O}(n^{\log_B A})$$
- "Rovnoměrné rozdělení" $f(n) = \mathcal{O}(n^c \log^k n)$ (pro $k \geq 0$) a $A = B^c$, potom
$$T(n) = \mathcal{O}(f(n) \log n) = f(n) = \mathcal{O}(n^c \log^{k+1} n)$$
- ... v obecnosti spoustu dalších případů

Intuitivně:

Pro jednoduchost předpokládejme, že $f(n) = \theta(n^c)$. Porovnějme složitost uzlu $\mathcal{O}(n^c)$ rekurze a jeho synů (dohromady) $\mathcal{O}(A * (n/B)^c)$. Z toho vidíme, že složitost vrstvy rekurze (součet přes uzly na dané úrovni) je A/B^C-násobek složitosti vrstvy o jedna výše. Celková složitost algoritmu je rovna součtu přes všechny vrstvy. Pokud sečteme složitost všech vrstev, dostaneme geometrickou řadu s koeficientem A/B^C.

Pokud $A/B^C < 1$, geometrická řada klesá, a její součet je asymptoticky roven velikosti prvního členu, což je kořen rekurze se složitostí $O(f(n)) = O(n^C)$.

Pokud $A/B^C > 1$, geometrická řada roste, a její součet je asymptoticky roven složitosti posledního členu, což je složitost listů. Složitost listu je $O(1)$, důležitý je tedy počet listů. Počet listů je $A^{\log_B n}$ (A je větvící faktor a $\log_B n$ je hloubka rekurze). Upravíme pomocí $A = B^{\log_B A}$ a $n = B^{\log_B n}$. Dostaneme složitost (počet listů) $\mathcal{O}(n^{\log_B(A)})$ (intuitivně, čím větší A a menší B, tím vyšší mocnina).

Pokud $A/B^C = 1$,geometrická řada degeneruje na sumu členů stejné velikosti. Celková složitost je tak násobek jedné (libovolné) vrstvy (třeba kořene, $\mathcal{O}(n^c)$) a počtu vrstev ($\mathcal{O}(\log_B n) = \mathcal{O}(\log n)$). Pro $k > 1$ projde téměř totožný argument.