Racunska zahtevnost

December 18, 2024

1 Računska zahtevnost

Poskusimo odgovorit na par vprašanj, ki si jih lahko zastavimo v zvezi s prejšnjimi urejevalnimi algoritmi.

- Kateri algoritmi so dobri in kateri slabi?
- Kateri algoritem je najboljši oz. kateri izmed dveh je boljši?
- Kako merimo učinkovitost oz. računsko zahtevnost algoritma?

Za algoritma s permuacijami lahko brez škode rečemo, da sta slaba. Poznamo precej hitrejše postopke urejanja, ki niso bistveno kompleksnejši (morda celo enostavnejši). Za ostale osnovne algoritme urejanja pa že ni povsem jasnega odgovora. Poznamo namreč učinkovitejše vendar tudi kompleksnejše algoritme. Tudi osnovni pristopi so lahko povsem primerni.

Pri iskanju najboljšega algoritma naletimo na podobno dilemo. Poleg tega ni jasno, na kakšnih podatkih želimo, da je algoritem najboljši - povsem naključnih, kakšnih posebnih, kako velikih?

To nas pripelje do tretjega vprašanja, kako sploh merimo učinkovitost algoritma?

- Lahko merimo **čas izvajanja**, vendar je te čase problematično primerjati na različnih računalnikih.
- Lahko merimo **število operacij**, ki jih potrebuje algoritem. Dogovoriti pa se moramo, *katere operacije* bomo šteli (primerjave, aritmetične, logične, pomnilniške, …)
- Dogovoriti se moramo, kakšen **primer podatkov** bomo obravnavali (najboljšem, najslabšem, povprečnem).
- Dogovoriti se moramo o **velikosti primerov**, s katerimi imamo opravka. En algoritem je lahko boljši za manjše primere, drugi pa se izkaže pri večjih.

Kot bomo videli v nadaljevanju, običajno ocenjujemo asimptotično zgornjo mejo števila operacij v najslabšem primeru.

Računska zahtevnost (kompleksnost) je količina virov, ki jih potrebuje algoritem za rešitev problema dane velikosti. Pri virih se običajno osredotočamo na čas in prostor, zato govorimo o **časovni** in **prostorski zahtevnosti**.

Ker imamo lahko različne podatke enake velikosti, moramo definirati, ali gre za **najboljšo**, **najslabšo** ali **povprečno** računsko zahtevnost. Običajno se osredotočamo na najslabšo (*worst-case*), če ni določeno drugače.

Natančno količino virov je pogosto težko izračunati, poleg tega pa ni pretirano praktično uporabna. Na računalniku z malenkost drugačno arhitekturo je že lahko drugačna. Poleg tega pa nas za majhne probleme običajno ne zanima, ker je takrat preglednost bolj pomembna od učinkovitosti. Zato se

običajno ukvarjamo z **asimptotično zahtevnostjo**, ki opisuje porabo virov algoritma pri zelo velikih problemih. Pri tem pogosto ocenjujemo neko mejo asimptotične zahtevnosti. Najpogosteje ocenjujemo **zgornjo mejo**, za kar se uporablja **notacija z velikim O-jem** (*Big O notation*). Rečemo, da ima funkcija $f(n)$ kompleksnost reda $g(n)$, kar zapišemo kot $O(g(n))$ ali $f(n) \in O(g(n))$ ali celo kar $f(n) = O(g(n))$ (čeprav ne gre za enakost). Formalno to pomeni:

$\exists k > 0\ \exists n_0\ \forall n > n_0 \colon f(n) \leq k\, g(n)$

ali enakovredno z limitami

$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$.

Poleg zgornje meje asimptotične zahtevnosti (veliki O) poznamo še notacije za druge meje (velika omega - Ω, velika theta - Θ, ...). Več o njih pa pri drugih algoritmičnih predmetih. Omenjene definicije lahko posplošimo tudi na funkcije z več spremenljivkami, če opazujemo časovno zahtevnost algoritma v odvisnosti od več parametrov velikosti problema.

> Najpogostejši primer je analiza zgornje meje asimptotične računske zahtevnosti v najslabšem primeru. S tem postavimo pesimistično oceno za najbolj neugoden primer velikih podatkov. Kadar govorimo o *časovni zahtevnosti*, običajno mislimo kar zgornjo mejo asimptotične časovne zahtevnosti v najslabšem primeru, če seveda ni pojasnjeno drugače.

Recimo, da smo izračunali čas izvajanja oz. število operacij za rešitev problema velikost n s funkcijo $f(n) = \frac{1}{2}(n-1)(n+2)\log n + \sqrt{n}$. Če izraz razširimo, dobimo $f(n) = \frac{1}{2}n^2 \log n + \frac{1}{2}n \log n - \log n + \sqrt{n}$. Časovno zahtevnost takega algoritma bi lahko ocenili kot $O(2n^3)$, kar je sicer pravilno, vendar precej nenatančna meja. Boljša ocena časovne zahtevnosti bi bila $O(n^2 \log n)$. Vsi ostali členi so namreč zanemarljivi v primerjavi z $n^2 \log n$, ko gre n proti neskončnosti (za potrebe zgornje meje bi jih lahko nadomestili z $n^2 \log n$), konstantni člen pred njim pa po definiciji ni relevanten. Primeren (ne pa edini) izbor konstant v zgornji definiciji bi bil npr. $n_0 = 2$ in $k = 3$, ker so vsi trije pozitivni členi manši ali enaki $n^2 \log n$ pri $n >= 2$. V praksi to pomeni, da:

- pri vsoti obdržimo samo najhitreje rastoči člen,
- pri produktu pa lahko zanemarimo konstantne faktorje.

Tipične časovne zahtevnosti so:

- $O(1)$, konstantna (neodvisna od velikosti problema n)
- $O(\log n)$, logaritemska
- $O(\sqrt{n})$, korenska
- $O(n)$, linearna
- $O(n \log n)$, loglinearna, linearitmična
- $O(n \log^c n)$ za konstanto $c > 0$, npr. $O(n \log^2 n)$ kvazilinearna
- $O(n^2)$, kvadratna
- $O(n^3)$, kubična
- $O(n^c)$ za konstanto $c > 0$, npr. $O(n^5)$, polinomska
- $O(c^n)$ za konstanto $c > 1$, npr. $O(2^n)$, eksponentna

Kako velike probleme lahko rešujemo z algoritmi določene časovne zahtevnosti, npr. $O(n^2)$? Ker ta sintaksa skriva konstantni faktor, tega ne moremo reči natančno. Dobra praktična ocena pa je, da lahko na tipičnem osebnem računalniku trenutno izvedemo približno 10^8 osnovnih operacij na sekundo.

1.0.1 Primeri

Oglejmo si nekaj primerov funkcij, ki predstavljajo računske zahtevnosti, in jih ocenimo z notacijo z velikim O-jem.

- $f_1(n) = 100 + 2n + 3n^2 = O(n^3)$ (ali $O(n^4 \log n)$, kar je sicer pravilna, vendar slaba meja)
- $f_2(n) = 3n\cos(2\pi n) + \frac{n}{\log n} + 2n = O(n)$
- $f_3(n) = 1 + n\log n + n^{1.5} = O(n^{1.5})$ (da logaritem raste počasneje kot koren, se lahko prepričate z uporabo l'Hôpitalovega pravila za izračun $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$)

Funkcija lahko vsebuje vsote kakšnih vrst.

- $f(n) = \sum_{k=1}^{n} n/k = n\sum_{k=1}^{n} 1/k = O(n\log n)$ (Harmonična vrsta)

Pogoste so tudi rekurzivne funkcije.

- $f(n) = n + f(n/2) = n + n/2 + n/4 + ... < 2n = O(n)$

Lahko imamo funkcije več spremenljivk.

- $f(n,m) = an^2 + n\sqrt{m} + b\log m = O(n^2 + n\sqrt{m})$ (a in b sta konstanti)

Parametriziran algoritem Načrtujemo algoritem, v katerem bomo problem velikosti n enakomerno razbili na skupine velikosti k, ki jih bo torej n/k. Izračunali smo, da lahko problem za posamezno skupino rešimo z algoritmom s korensko časovno zahtevnostjo (v odvisnosti od velikosti skupine), časovna zahtevnost postopka združevanja rezultatov več skupin pa je kubična (v odvisnosti od števila skupin). Kako naj izberemo parameter k, da bo časovna zahtevnost algoritma čim boljša?

$f(n;k) = n/k \cdot O(\sqrt{k}) + O((n/k)^3)$. Oglejmo si ekstremne primere. Pri $k = 1$ dobimo $f(n) = n + n^3 = O(n^3)$, pri $k = n$ pa $f(n) = \sqrt{n} + 1 = O(\sqrt{n}) = O(n^{0.5})$. V prvem primeru je večji drugi člen, v drugem primeru pa prvi člen. Želimo, da noben od njiju ne dominira, torej naj bosta enaka. Iz enačbe $n\sqrt{k}/k = (n/k)^3$ lahko določimo $k = n^{4/5}$ in $f(n) = O(n^{2/5}) = O(n^{0.4})$.

Analiza programa Ocenimo časovno zahtevnost spodnjega programa.

```
for (int x = 1; x <= n; x *= 2) {
  for (int i = 0; i < x; i++) {
    for (int j = 0; j < n; j += 2) {
       // konstantno število operacij
    }
    for (int j = 1; j < n; j *= 2) {
      // konstantno število operacij
    }
  }
}
```

Na for zanke se bomo sklicevali kar s prva, druga, tretja in četrta, kot se pojavijo v programu. Določimo, največ kolikokrat se izvede katera od njih: prva $\log n$-krat, druga: n-krat, tretja: $n/2$-krat in četrta: $\log n$-krat. Tretja in četrta se izvedeta zaporedno, pri čemer dominira tretja. Časovno zahtevnost lahko zato ocenimo z $O(\log n \cdot n \cdot (n/2 + \log n)) = O(n^2 \log n)$.

Pri ocenjevanju časovne zahtevnosti pa smo lahko bolj natančni. Število ponovitev druge zanke je namreč odvisno od trenutne iteracije prve zanke (v prejšnjem odstavku smo vzeli kar najbolj pesimistično oceno). Število izvedb druge zanke bo $1 + 2 + 4 + 8 + ... + n = O(n)$, za vsako od teh ponovitev pa tretja zanka prispeva še $O(n)$ operacij. Bolj natančna ocena časovne zahtevnosti je torej $O(n^2)$.