PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **14** (1986/1987)
Številka 2
Strani 125–128

Sandi Klavžar:

ČASOVNA ZAHTEVNOST ALGORITMOV

Ključne besede: matematika, računalništvo, algoritem, časovna zahtevnost.

Elektronska verzija: http://www.presek.si/14/826-Klavzar.pdf

ČASOVNA ZAHTEVNOST ALGORITMOV

Gotovo so med vami lastniki računalnika SPECTRUM 16 K. Prepričan sem, da so se že večkrat jezili, ker niso mogli naložiti programov, ki so sicer pisani za povsem enak računalnik SPECTRUM 48 K, le da ima slednji na razpolago večji pomnilnik.

Zadnjič sem videl sprogramiran šah za mikroračunalnik, ki ima, kot se to seveda spodobi, več težavnostnih stopenj. Na nižjih stopnjah ga je kaj lahko premagati, če niste prehud začetnik. Problem pa nastane, če bi hoteli ugotoviti, kako igra na najzahtevnejši stopnji. Približni čas, ki si ga računalnik vzame za eno potezo, je namreč nič več in nič manj kot 24 ur. Rad bi ga poznal, ki je z računalnikom odigral ta šah na najzahtevnejši stopnji!

V prvem primeru smo naleteli na težave s prostorom in v drugem primeru na težave s časom. Zato si oglejmo, kako bi prostor in čas, ki ju potrebuje računalnik, natančneje definirali.

Ker nas bolj zanima časovna zahtevnost algoritma, bomo pustili prostorsko zahtevnost ob strani. Algoritem lahko definiramo povsem strogo, vendar naj nam zadošča že intuitivna slika. Recimo, da rešujemo neki problem. *Algoritem* je spisek natančno določenih navodil, ki nas v končnem času po strogo začrtani poti pripeljejo do rešitve problema. Problem lahko rešimo na več načinov (z različnimi algoritmi). Algoritem bo tem boljši, čim manj prostora in čim manj časa bo porabil. Vsak problem ima tudi svojo *velikost.* Velikost problema je mera za obsežnost vhodnih podatkov. Če na primer urejamo neko zaporedje, je velikost problema število elementov zaporedja, ki ga želimo urediti.
Definicija 1. *Časovna zahtevnost* algoritma je čas, ki ga potrebuje algoritem za rešitev problema pri dani velikosti problema.

Čas je funkcija velikosti problema. Ker različni računalniki različno hitro računajo, je smiselno, da za enoto časa vzamemo osnovno računsko operacijo (seštevanje, odštevanje, množenje, deljenje). Ravno tako za enoto štejemo primerjanje vrednosti dveh spremenljivk. Če je časovna zahtevnost $3n^4 - 5n$, algoritem opravi $3n^4 - 5n$ osnovnih računskih operacij in primerjanj.

Podobno definiramo *prostorsko zahtevnost* algoritma.

Zanimala nas bo le hitrost rasti zahtevnosti algoritma in ne točna vrednost, zato definirajmo:
Definicija 2. Funkcija $g(n)$ je $O(f(n))$, če obstaja konstanta $C > 0$, tako da je: $g(n) \leqslant C.f(n)$, za vse dovolj velike n.

Definicija potrebuje nekaj pojasnil. Da $g(n) \leqslant C.f(n)$ velja za vse dovolj velike n, pomeni, da velja za vsa naravna števila, ki so večja od nekega naravne-

ga števila n_0. Izjavo $g(n) = O(f(n))$ preberemo takole: funkcija $g(n)$ je "veliki o od" $f(n)$.

Časovno zahtevnost algoritma (in tudi prostorsko) navadno opišemo s pomočjo velikega o.

Primer 1. Velikokrat imamo opraviti z zahtevnostjo, ki se izraža s polinomom v velikosti problema. Polinom $g(n)$ stopnje k je funkcija oblike

$$g(n) = a_k.x^k + a_{k-1}.x^{k-1} + \ldots + a_1.x^1 + a_0$$

kjer so koeficienti a_i poljubna realna števila in je $a_n \neq 0$. Recimo, da je $g(n) = 2n^4 + n^3 - n^2 - 23n + 10$. Tedaj je zahtevnost algoritma $O(n^4)$. Če namreč za konstanto C vzamemo $C = 3$, potem je pogoj $2n^4+n^3-n^2-23n+10 \leqslant 3n^4$ izpolnjen kar za vsa naravna števila in lahko izberemo $n_0 = 0$. Nasploh velja, da zahtevnost polinoma dobimo tako, da vzamemo vodilno potenco polinoma. Tako je $5n^3 + 100n^2 + 99n = O(n^3)$. Kako velik n_0 moramo vzeti, če izberemo $C = 6$? ($n_0 = 100$). Seveda bi bil n_0 manjši, če bi izbrali večjo konstanto C.

Vprašajmo se, kakšna je časovna zahtevnost algoritma za igranje šaha. V načelu je algoritem tak, da pregleda vse kombinacije in se nato odloči za najboljšo. No, vseh možnih kombinacij do konca partije prav gotovo ne bo pregledal. Za koliko potez vnaprej naj torej gleda vse možnosti? Če gleda samo za eno potezo, tedaj niti ni tako veliko možnosti. Seveda bo igral temu primerno slabo. Če se odloči za dve, mora upoštevati vse možne nasprotnikove odgovore na prvo potezo ... Gotovo slutite, da število kombinacij z vsako naslednjo potezo strahovito hitro raste. Seveda bo igral tem bolje, čim več potez vnaprej bo gledal. Odtod izvira velika poraba časa za igranje na zahtevnejših stopnjah. Ustvarjalci programov za igranje šaha se zato trudijo, da bi z globljim poznavanjem šaha pregledovali samo nekatere kombinacije in se med temi odločili za pravo potezo.

V ocenjevanju zahtevnosti imajo poseben pomen eksponentne funkcije. Funcija $f(n)$ je *eksponentna,* če je oblike $f(n) = a^n$, kjer je a neko pozitivno število. Primer eksponentnih funkcij sta $f(n) = 2^n$ in $f(n) = e^n$.

Naloga. Pokaži, da funkcija $f(n) = 2^n$ ni $O(n^k)$ za noben k.

Glavna značilnost eksponentnih funkcij je, da strahovito hitro naraščajo. Poznamo tudi funkcije, ki naraščajo še hitreje kot eksponentne, na primer

$$f(n) = 2^{2^n}$$

pa tudi take, ki naraščajo počasneje kot eksponentne, pa vendar hitreje kot katerakoli potenca (na primer $f(n) = n^{\log n}$). Kakršnokoli funkcijo izmed omenjenih že srečamo, se nam ne piše dobro.

Naloga. Recimo, da je velikost problema n. Imamo tri algoritme: prvi je zahtevnosti n, drugi n^3 in tretji 3^n. Predpostavimo, da ručunalnik za eno operacijo potrebuje mikrosekundo (10^{-6} s). Kako velika je lahko naloga v vsakem od primerov, če imamo na razpolago uro računalniškega časa ($3.6.10^9$, 1532, 20)?

Algoritem je eksponenten, če je zahtevnosti $O(f(n))$, kjer je $f(n)$ eksponentna funkcija. Podobno je algoritem *polinomski,* če je zahtevnosti $O(f(n))$, kjer je $f(n)$ polinom. Problem igranja šaha je po svoji naravi časovno zelo zahteven, saj je njegova zahtevnost vsaj eksponentna, če ne celo več. Če za dani problem ne znamo poiskati boljšega kot eksponentni algoritem, potem lahko obupamo ali pa malo pogoljufamo pri reševanju problema na različne načine. Pri šahu pogoljufamo tako, da ne gledamo preveč potez vnaprej in s tem naredimo problem majhen.

Naloga. Morda kdo misli, da bodo hitrejši računalniki rešili počasne algoritme. Vendar temu ni tako. Recimo, da imamo na razpolago tisočkrat hitrejši računalnik kot v prejšnji nalogi, torej osnovno operacijo opravi v nanosekundi (10^{-9} s). Za iste zahtevnosti se zopet vprašajmo po največji obsežnosti, ki jo za vsakega med njimi lahko opravimo v eni uri ($3.6.10^{12}$, 15327, 26). Opazimo, da smo pri eksponentnem algoritmu zelo malo pridobili.

Za konec si oglejmo še nekaj primerov.

Primer 2. Iskanje največjega elementa zaporedja.

Recimo, da imamo neko poljubno zaporedje n števil, v katerem moramo poiskati največji element. Prav gotovo moramo pogledati vsa števila, saj bi lahko bilo največje število ravno tisto, ki ga nismo pogledali. Narediti moramo torej vsaj n osnovnih operacij. Sestavi algoritem, ki bo poiskal največji element zaporedja v času $O(n)$.

Primer 3. Urejanje zaporedja.

V praksi se velikokrat srečamo s problemom urejanja zaporedja. Zanj je izdelana že zelo obsežna teorija. Morda najpreprostejši algoritem bi bil naslednji: poiščemo največji element in ga damo na prvo mesto, nato med preostalimi poiščemo največjega in ga damo na drugo mesto, ... Poizkusi dokazati, da je časovna zahtevnost tega algoritma $O(n^2)$. Toliko časa algoritem potrebuje tudi v primeru, ko so vhodni podatki že urejeni. Radovednejšim naj povemo, da je časovna zahtevnost najboljših algoritmov za urejanje $O(n.\log n)$.

Primer 4. Delitev množice.

Imejmo končno množico A z elementi iz množice naravnih števil. Vprašajmo se, ali lahko to množico razdelimo na dve podmnožici A_1 in A_2 tako, da je vsota elementov v podmnožicah enaka. Če je $A = \{13, 4, 10, 11, 8\}$, sta množici $A_1 = \{13, 10\}$ in $A_2 = \{4, 11, 8\}$ rešitev problema delitve množice A. Problem je videti preprost, tako da ga verjetno ni težko rešiti. Vendar nas prvi