PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **15** (1987/1988)
Številka 4
Strani 198–201

Tomi Dolenc:

METODA „OSTREGA POGLEDA“ V PROGRAMIRANJU

Ključne besede: računalništvo, programiranje, izboljševanje programov.

Elektronska verzija: http://www.presek.si/15/902-Dolenc.pdf

METODA "OSTREGA POGLEDA" V PROGRAMIRANJU

Metodo "ostrega pogleda" uporabljamo v vseh vejah znanosti, pa tudi na drugih področjih, ki ustvarjajo probleme. Opis metode je zelo preprost. Zastavljeni problem najprej dobro definiramo in po možnosti kam zapišemo. Nato se ostro zagledamo v njegovo definicijo. Ko ostrina pogleda doseže določeno stopnjo, se nam posveti rešitev. ★

Danes si bomo ogledali dva problema. Za oba obstajata preprosti rešitvi po načelu "kar mi najprej pade na pamet" ali "z glavo skozi zid". Vendar so taki postopki po pravilu potratni, kar bomo ugotovili z oceno časovne zahtevnosti (pojem časovne zahtevnosti smo že spoznali v Preseku XIV/2). Potem si bomo problem ostro ogledali in sestavili kratka, elegantna in predvsem učinkovita postopka za njuno reševanje. Poskusite z reševanjem tudi sami, še preden preberete prvo rešitev!

Naloga 1. Tabelo a z n elementi krožno premakni v levo za k mest. Če je npr. $n = 10, k = 3$ in tabela a

A B C D E F G H I J

je po končani operaciji tabela a takale:

D E F G H I J A B C

Pri reševanju ne smemo uporabljati dodatnih tabel.

Rešitev, ki se je dokaj hitro spomnimo, je naslednja: napišemo del programa, ki tabelo krožno premakne za eno mesto v levo, nato pa ga uporabimo k–krat na isti tabeli.

```
for j := 1 to k do begin
    pom := a [ 1 ];
    for i := 2 to n do a [ i – 1 ] := a [ i ];
    a [ n ] := pom;
end;
```

Brez težav analize lahko ugotovimo, da je število korakov (stavkov, ki se izvršijo) v tem programu približno $k * n$. Ker k ni fiksen, vzamemo kar njegovo povprečno velikost $n/2$. Število korakov je torej neka kvadratna funkcija (polinom) spremenljivke n. Tako ocenimo, da je čas izvajanja programa sorazmeren kvadratu števila podatkov, kar zapišemo kot $O(n^2)$. To pomeni, da program za

obdelavo dvakrat večjega števila podatkov porabi štirikrat več časa. Radi bi našli hitrejši postopek, pri katerem bi čas rasel sorazmerno s količino podatkov.

Zato se ostro zazremo v problem in opazimo naslednje. Vzemimo podprogram *Obrni,* ki zrcali poljuben podvektor tabele *a* (npr. ABC → CBA). Najprej zrcalimo prvi del tabele (od prvega do *k*–tega elementa), nato ostanek, na koncu celotno tabelo. In glej! Krožni premik je opravljen. Oglejmo si na prejšnjem primeru, kako to poteka.

	A	B	C	D	E	F	G	H	I	J
Obrni (1, k);	**C**	**B**	**A**	D	E	F	G	H	I	J
Obrni (k + 1, n);	C	B	A	**J**	**I**	**H**	**G**	**F**	**E**	**D**
Obrni (1, n);	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**A**	**B**	**C**

Tudi podprogram *Obrni* ne dela težav.

```
procedure Obrni (prvi, zadnji : integer);
var i : integer; pom : char;
begin
    for i := prvi to (prvi+zadnji–1) div 2 do begin  { zamenjaj }
        pom := a [ i ]; a [ i ] := zadnji–i+1; a [ zadnji–i+1 ] := pom;
    end;
end; { Obrni }
```

Poskusi ugotoviti, kakšna je časovna verjetnost tega postopka!

Naloga 2. Dana je tabela *a* velikosti *n*, ki vsebuje celoštevilčne vrednosti. Poišči največjo vsoto, ki jo lahko dobimo s seštevanjem členov nekega strnjenega podzaporedja v tej tabeli. Če je torej vhodna tabela

55 −14 −100 11 13 −17 8 44 −16 2

je rešitev vsota elementov *a* [4] .. *a* [8], to je 59. Problem je seveda povsem preprost, če v tabeli ni negativnih števil, saj je iskano število kar vsota vseh elementov tabele. Težave nastanejo, ko naletimo na negativno število – se ga splača vključiti v podzaporedje? Privzemimo, da je iskana vsota enaka 0, če v tabeli nastopajo le negativna števila.

Seveda lahko takoj zapišemo "butn–verzijo" programa: za vsak par indeksov *levi* in *desni* (1 < = *levi* < = *desni* < = *n*) izračunamo vsoto *a* [*levi..desni*] in pogledamo, če je večja od doslej največje vsote. Ustrezni del programa v pascalu je kratek in lahko razumljiv:

```
maxVsota := 0;
for levi := 1 to n do
    for desni := levi to n do begin
        vsota := 0;
        for i := levi to desni do vsota := vsota + a [ i ]
        { zdaj je to vsota podzaporedja a [ levi..desni ] }
        if vsota > maxVsota then maxVsota := vsota;
    end;
write ('Iskana vsota je ', maxVsota);
```

Vendar pa je ta postopek časovno zelo potraten, kar bomo takoj ugotovili. Iz programa lahko razberemo, da se zunanja zanka "obrne" ravno n–krat, pri tem pa se vsakič (n–*levi*)–krat izvede druga zanka. Torej je število izvajanj notranjega sklopa stavkov (med **begin** in **end**) približno $n^2/2$, torej $O(n^2)$, sam sklop pa zahteva zase $O(n)$ časa (notranja zanka v njem ima največ n korakov). Tako smo ugotovili, da celotni postopek potrebuje za izračun rezultata $O(n^3)$ korakov. To pa pomeni, da se število korakov programa — in s tem čas izvajanja — poveča tisočkrat, če vhodno tabelo desetkrat podaljšamo, ali drugače: če bi kak računalnik opravil s tabelo desetih elementov v npr. 10 milisekundah (kar je zmerna hitrost), bi za tisoč elementov potreboval skoraj tri ure!

Torej se zazrimo zopet v problem.

Zgornji postopek se očitno da izboljšati. Delnih vsot ni treba računati vsakič na novo, saj lahko vsoto zaporedja *a* [*levi..desni*] izračunamo tako, da vsoti elementov *a* [levi..desni–1] prištejemo še vrednost *a* [*desni*]. Tako smo se znebili notranje zanke in zmanjšali število vseh operacij na $O(n^2)$.

```
maxVsota := 0;
for levi := 1 to n do
    vsota := 0;
    for desni := levi to n do begin
        vsota := vsota + a [ i ];
        { zdaj je ta vsota podzaporedja a [ levi..desni ] }
        if vsota > maxVsota then maxVsota := vsota;
    end;
write ('Iskana vsota je ', maxVsota);
```

Dobljeni programček je že prav simpatičen in smo z njim lahko kar zadovoljni. Vendar pa nam žilica ne da miru. Ali bi postopek lahko še pospešili?

Razmišljamo takole: če poznamo rešitev za tabelo *a* [1..*i* – 1], kako bi

dobili rešitev za razširjeno tabelo *a* [1 .. *i*] ? Ta pristop je pogosto uporaben, kadar imamo opravka s tabelami. Za naš primer velja naslednje: podzaporedje z maksimalno vsoto v tabeli z *i* elementi lahko vsebuje novi (*i*–ti) element ali pa ne. Zato ves čas spremljamo dve spremenljivki: *maxDoslej* pove, koliko znaša rezultat za že obdelani kos tabele, *maxDesni* pa je največja vsota takega podzaporedja, ki vsebuje skrajni desni element tega kosa. To si lahko ogledamo na primeru:

1 2 3 –10 1 2 ?

maxDoslej = 6, *maxDesni* = 3

Ko tabelo razširimo, se vrednosti *maxDesni* poveča za vrednost novega elementa (razen če bi tako postala negativna – v tem primeru dobi v skladu z našim dogovorom vrednost 0). Če je s tem postala večja od *maxDoslej*, popravimo tudi to spremenljivko. Ko dosežemo konec tabele, je *maxDoslej* iskani rezultat. Zapis programa je zelo kratek:

```
maxDoslej := 0;
maxDesni := 0;
for i := 1 to n do begin
    maxDesni := maxDesni + a [ i ];
    if maxDesni < 0 then maxDesni := 0;
    if maxDesni > maxDoslej then maxDoslej := maxDesni;
end;
write ('Iskana vsota je ', maxDoslej);
```

Iz programa lahko hitro razberemo, koliko časa porabi: edina zanka se obrne natanko *n*–krat, znotraj nje pa ni stavkov, katerih izvršitev bi bila odvisna od števila podatkov. Maksimalni čas izvajanja programa je torej $O(n)$.

Kaj smo z izboljšavo postopka pridobili razen očitne "elegance" programa? Oglejmo si to na nekoliko pretiranem primeru. Če gornji postopek razumemo, lahko s papirjem in svinčnikom izračunamo rezultat za tabelo z desetimi elementi v približno eni minuti. Za sto elementov bi potrebovali deset minut, za tisoč pa uro in štirideset minut – če bi imeli potrpežljivost računovodje. In že smo prehiteli računalnik, ki računa po prvotnem postopku! Pa tudi za 10000 elementov bi naš pridni računovodja potreboval le 17 ur ali dva zelo delovna dneva, medtem ko bi se računalnik mučil več kot tri mesece! Se pa že splača malo "ostreje pogledati" problem, mar ne?

Tomi Dolenc

★ Metoda je seveda zasnovana optimistično. Kot bomo spoznali v eni od naslednjih številk Preseka, samo od rešitve toliko dlje, kolikor bolj gledamo naravnost vanjo.