PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **18** (1990/1991)
Številka 3
Strani 146–152

Ciril Pezdir:

# NASLOV

Ključne besede: računalništvo.

Elektronska verzija: http://www.presek.si/18/1036-Pezdir.pdf

# NASLOV

Kaj je skupnega besedam **'beseda'**, **'tri'** in naslovu tega prispevka? Na nek način opisujejo same sebe. Beseda **'NASLOV'** nastopa na mestu naslova, beseda **'beseda'** opisuje pojem besede in je tudi sama beseda, beseda **'tri'** pa opisuje število tri in je sestavljena iz treh črk. Ali obstaja poleg besede **'tri'** še katera, ki sebe opiše na tak način? Kar štejmo: štiri, pet, šest, sedem, osem, devet, deset,... Počasi izgubljamo upanje, da bo mogoče naslednje število pravo. Pri dovolj velikem številu upanje opustimo in zaključimo, da ga ni. Kako pa je s takimi besedami v drugih jezikih? V angleščini je taka beseda **'four'** v nemščini **'vier'**, v latinščini je sploh ni,... Posebno mesto med besedami, ki opisujejo same sebe, ima beseda **'jaz'**. Poleg tega, da govori o sebi, govori tudi o osebi, ki jo izgovarja.

Poleg besed poznamo tudi stavke, ki opisujejo sami sebe. Npr.: **'Ta stavek vebuje slovnično napako'**, **'Ta stavek brez glagola'**, **'V tem stavku je šest besed'**, **'Ta stavek ima enaintrideset malih črk'**,... Da najdemo zadnji stavek je potrebno že malo več truda. Pa poskusimo najti stavek oblike **'Ta stavek ima ___ črk'**. Po nekaj poskusih vstavitve besede za število črk in štetju črk opazimo, da se nam rešitev izmika. Kljub temu, da ne preverimo vseh števil, zaključimo, da rešitev ne obstaja. Toda pravi matematiki dvomijo v vsako trditev, ki je ne znajo dokazati. Kako pa bi dokazali naš zaključek, da rešitev ne obstaja?

Zanimivo je tudi vprašanje, ali je stavek **'Ta stavek vebuje slovnično napako'** pravilen. Če vzamemo za kriterij pravilnosti slovnično pravilnost, potem seveda ni pravilen. Kaj pa če vzamemo za kriterij pravilnosti resničnost tega, kar stavek govori? Že res, da stavek vsebuje slovnično napako, ampak stavek pravi, da **'vebuje'** slovnično napako! O čem torej stavek sploh govori?

Mogoče se kdo še spominja naloge, objavljene v P IX-4. Naloga je zahtevala, da v stavek

'V tem stavku 0 nastopa __ krat,
1 nastopa __ krat,
2 nastopa __ krat,
3 nastopa __ krat,
4 nastopa __ krat,
5 nastopa __ krat,
6 nastopa __ krat,
7 nastopa __ krat,
8 nastopa __ krat,
9 nastopa __ krat.

vpišemo na črtice števila, pisana na običajen način, tako da bo stavek govoril resnico. Poleg tega pa je naloga še zahtevala, da poiščemo vse rešitve. Kako se naloge lotimo?

Rešitev predstavimo z vektorjem $(x_0, x_1, \ldots x_9)$. Komponenta $x_i$ nam pove, kolikokrat v stavku nastopa številka $i$. Komponente vektorja so torej števila, s katerimi štejemo, to pa so ravno naravna števila. Rešitev lahko iščemo s sistematičnim pregledovanjem 'vseh možnih' vektorjev, s poizkušanjem in popravljanjem ali pa z enostavno metodo, ki jo bomo uporabili kasneje za reševanje podobne, a zahtevnejše naloge.

S sistematičnim pregledovanjem bi gotovo prišli do rešitve, če ta seveda obstaja, vprašanje pa je, koliko časa bi za to potrebovali. Vseh vektorjev ne moremo pregledati, saj jih je neskončno mnogo. Torej moramo iskanje omejiti. Vektorji, katerih komponente so velika števila, za rešitev ne pridejo v poštev. Za vsako številko lahko razmislimo, največ kolikokrat bi se lahko pojavila v rešitvi. Npr.: številka 9 se v rešitvi verjetno ne bo pojavila več kot 3-krat. Tako pridemo do vrednosti komponent, za katere je še smiselno pregledovati vektorje. Pomagamo si lahko z računalnikom. Vgnezdimo deset **for** stavkov in za vsako kombinacijo števcev preverimo, če je rešitev. Zaradi enostavnosti naj vsi števci **for** stavkov tečejo od 1 do 20.

```
for x0:=1 to 20 do
    for x1:=1 to 20 do
        ...
        for x9:=1 to 20 do
            if resitev(x) then izpisi(x);
```

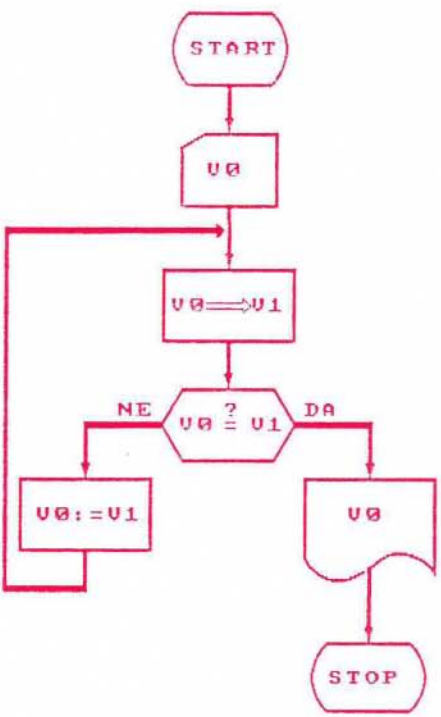
Logična funkcija 'resitev' vrne vrednost 'true', če je vektor $x$ rešitev, in 'false', če ni. Podprogram 'izpisi' pa izpiše vektor $x$. Tako bomo preverili vse vektorje med (1,1,1,1,1,1,1,1,1,1) in (20,20,20,20,20,20,20,20,20,20), teh pa je kar $20^{20}$. Če bi lahko računalnik preveril 1000 vektorjev v sekundi, bi za to delo potrebovali več kot tristo let! Če pa bi bile zgornje meje vrednosti v **for** stavkih po vrsti naslednje: 3,13,5,5,4,4,3,3,3,3, bi za to delo potrebovali "le" 21 minut. Zakaj ravno take zgornje meje? Določil sem jih po "občutku" - čim nižje so, bolj nevarno je, da izpustimo vektor, ki je rešitev. Vidimo, da si lahko z ustreznim razmislekom in nekaj tveganja prihranimo veliko časa. Takemu sistematičnemu pregledovanju možnih rešitev pravimo **metoda grobe sile**. Običajno je primerna za reševanje problemov majhnega obsega.

Še najhitreje pridemo do rešitve s svinčnikom in radirko v roki, s poskušanjem in popravljanjem. Poznam dve rešitvi in ne dvomim dosti, da sta edini.

Če s poskušanjem ne uspete najti obeh, vam bo to gotovo uspelo s pomočjo naslednje preproste metode.

Izmislimo si poljuben začetni vektor, npr. (1,1,1,1,1,1,1,1,1,1). Če v stavek vstavimo ta vektor, je stavek nepravilen. Sedaj izračunamo naslednji vektor tako, da bo pravilno opisoval stavek z začetnim vektorjem. To je vektor (1,11,1,1,1,1,1,1,1,1). Ker je stavek še vedno nepravilen, računanje naslednjega vektorja ponovimo. Dobimo (1,12,1,1,1,1,1,1,1,1), (1,11,2,1,1,1,1,1,1,1), (1,11,2,1,1,1,1,1,1,1), ... Že po treh korakih smo prišli do rešitve. To rešitev označimo z $r_0$, drugo, ki jo boste našli sami, pa z $r_1$.

Na sliki 1 je diagram poteka za pravkar opisani postopek.




1. preberemo začetni vektor $V0$
2. s pomočjo vektorja $V0$ izračunamo vektor $V1$
3. če sta vektorja $V0$ in $V1$ enaka, izpišemo rešitev $V0$ in končamo, drugače pa $V0$ postane enak $V1$ in postopek nadaljujemo na 2.

**Slika 1.** Diagram poteka

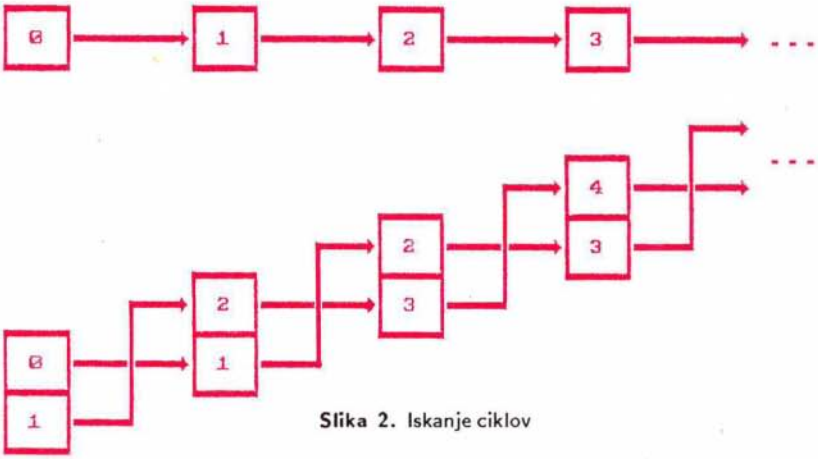
Napišimo še program, ki bo izvajal ta postopek.

```
program samoOpis;
type vektor=array [0..9] of integer;
var v0,v1:vektor;
    i,korak:integer;

  procedure izracun(var v1:vektor;v0:vektor);
                                  { s pomocjo vektorja v0 izracunamo vektor v1}
  begin
    for i:=0 to 9 do v1[i]:=1;                  {vrednost komponent vektorja v1
je najmanj 1}
    for i:=0 to 9 do
      while v0[i] <> 0 do begin
                                  {v0[i] komponento delimo na cifre in povecamo}
        v1[v0[i] mod 10]:=v1[v0[i] mod 10]+1;
                                                        {ustrezno komponento v1}
        v0[i]:=v0[i] div 10;
      end;
  end; {izracun}

  function enaka(v0,v1:vektor):boolean;
                        {=true, ce sta vektorja v0 in v1 enaka, =false drugace}
  var eenaka:boolean;
  begin
    eenaka:=true;
    i:=0;
    while eenaka and (i ≤ 9) do begin
      if v0[i] <> v1[i] then eenaka:=false;
                                        {v0 in v1 primerjamo po komponentah}
      i:=i+1;
    end;
    enaka:=eenaka;
  end; {enaka}

begin {samoOpis}
  writeln;
  writeln('Zacetni vektor: '); {vnos zacetnega vektorja}
  for i:=0 to 9 do read(v0[i]);
  v1:=v0;
  korak:=0;
  repeat                        {ponavljamo postopek dokler nista v0 in v1 enaka}
    v0:=v1;
    izracun(v1,v0);
    korak:=korak+1;
    write(korak:3,'. ');                {izpis zaporednega koraka in vektorja v1}
    for i:=0 to 9 do write(v1[i],' ');
    writeln;
  until enaka(v0,v1);
  write('Resitev: ');                                          {izpis resitve}
  for i:=0 to 9 do write(v0[i],' ');
end. {samoOpis}
```

Po vnosu začetnega vektorja, program ponavlja postopek tako dolgo, dokler ne najde rešitve. Po nekaj poskusih ugotovimo, da nekateri začetni vektorji 'konvergirajo' k rešitvi $r_0$ , nekateri k rešitvi $r_1$ , vsi ostali pa se ujamejo v nek cikel in ne pripeljejo do rešitve.

Da se bomo lažje izražali, se dogovorimo za nekaj pojmov. Rekli bomo, da je oddaljenost vektorja $v_1$ od vektorja $v_2$ enaka $n$, če naš postopek pripelje vektor $v_1$ do vektorja $v_2$ v $n$ korakih. Številu vektorjev, ki se ponavljajo, kadar postopek ne pripelje do rešitve, bomo rekli dolžina cikla. Z $V_n(v)$ bomo označili množico vseh tistih vektorjev, katerih oddaljenost od vektorja $v$ je enaka $n$.

Kako teče postopek za iskanje rešitve, to je "cikla" dolžine 1, smo si že ogledali, shematično pa je prikazan na sliki 2a. Vsak pravokotnik predstavlja vektor z 10 komponentami, število v njem pa nam pove, kolikokrat smo uporabili postopek nad začetnim vektorjem, označenim z $v_0$, da smo dobili ta vektor. Puščice predstavljajo izračun novega vektorja. Postopek končamo, ko sta dva zaporedna vektorja enaka.




**Slika 2.** Iskanje ciklov

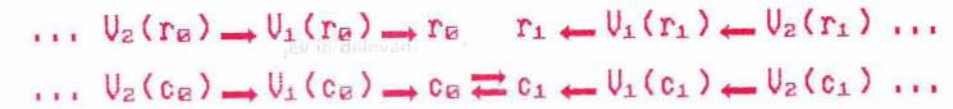
Cikel dolžine 2 lahko iščemo s pomočjo vektorja z 20 komponentami (slika 2b). Na vsakem koraku izračunamo naslednji vektor s pomočjo zadnjega tako, da drugi del naslednjega vektorja izračunamo s pomočjo prvega dela zadnjega, prvi del naslednjega pa iz drugega dela zadnjega. Tudi tu postopek končamo, ko sta dva zaporedna vektorja enaka, cikel dolžine 2 pa smo našli, če sta dela z 10 komponentmi različna. Vseskozi moramo torej poznati zadnja dva vektorja z 20 komponentami. Če pa bi želeli iskati cikle kakršnekoli

dolžine, bi izvajali postopek, kot je prikazan na sliki 2a, pri čemer bi si morali zapomniti vse vektorje in ob vsakem izračunu novega preveriti ali je morda enak kateremu izmed prejšnjih.

Ali se lahko zgodi, da postopek generira vedno nove vektorje, ne da bi se ujel v cikel? To bi bil "cikel" z neskončno dolžino.

Z računalnikom sem preizkusil kar precej vektorjev in edini cikel, ki sem ga našel, ima dolžino dva. Izmenjujeta se dva vektorja, ki ju bom imenoval $c_0$ in $c_1$. Ker ju ni težko najti, vam njuno iskanje prepuščam za vajo.

Porazdelitev vektorjev po množicah in delovanje postopka na njih je prikazano na sliki 3. Ali so s to sliko zajeti vsi vektorji? Če so, potem sta $r_0$ in $r_1$ edini rešitvi, vektorja $c_0$ in $c_1$ pa sestavljata edini cikel.




**Slika 3.** Porazdelitev vektorjev in delovanje postopka

In sedaj:

**Ta stavek, narejen s pomočjo računalnika, vsebuje petindvajset a-jev, dva b-ja, en c, tri č-je, osemnajst d-jev, sedeminpetdeset e-jev, en f, en g, dva h-ja, šestnajst i-jev, dvaintrideset j-jev, tri k-je, dva l-ja, sedem m-jev, enaindvajset n-jev, šest o-jev, šest p-jev, devet r-jev, triindvajset s-jev, pet š-jev, osemindvajset t-jev, tri u-je, devetindvajset v-jev, dva z-ja, en ž in šestdeset praznih mest.**

Bo mogoče kdo preveril njegovo pravilnost? Da sem prišel do tega stavka, sem uporabil zgoraj opisani postopek. Stavek iz prejšnje naloge je opisovala deseterica števil, ta stavek pa opiše šestindvajseterica. Sestaviti stavek, ki na tak način opisuje (dokumentira) samega sebe, je precej zahtevnejša naloga od prejšnje. Tu se postopek ne ustali tako hitro niti v rešitvi niti v kakšnemu ciklu. Za iskanje ciklov sicer nisem naredil programa, sem pa s pomočjo stolpičnega diagrama opazoval, kako se vektor med postopkom spreminja. Če bi bil cikel dovolj kratek, bi to opazil. Za občutek naj povem, da sem za določene začetne vektorje opazoval tudi po 1000 korakov, pet vsako sekundo. Program je bil napisan v turbo pascalu, izvajal pa sem ga na računalniku IBM PC-XT, 8MHz.

Program sem preuredil tako, da računalnik sam nastavi začetni vektor, katerega komponente so slučajna števila med 1 in 80, in ga spusti v postopek. Če v 50 korakih postopek ne pripelje do rešitve, nastavi nov začetni vektor. Kolikšna je verjetnost, da program izbere vektor, oddaljen od rešitve največ

50 korakov? Kaj vpliva na izbiro optimalnega števila korakov, ki jih mora prehoditi začetni vektor, dokler ga ne zamenjamo z novim? Pri tem moramo upoštevati tudi čas, to je našo nestrpnost.

Program za iskanje samodokumentiranega stavka sem pognal večkrat. Enkrat je prišla rešitev že po nekaj sekundah, večkrat pa sem moral nanjo počakati tudi po več kot 20 minut. Zadnje pomeni, da je računalnik moral pred tem preizkusiti več kot 120 vektorjev. Zgoraj zapisana rešitev je edina, ki jo poznam. Poskušal sem najti rešitve tudi za stavke drugačne oblike, npr.: **Ta stavek vsebuje** x **a-jev,** y **b-jev, ...** ali pa samo x **a-jev**, y **b-jev**, ..., pa nisem našel ničesar. Včasih sem pustil program teči tudi več kot uro, ročno sem vnašal začetne vektorje, spreminjal število korakov, grafično opazoval in zvočno poslušal spreminjanje vektorja, toda brez uspeha.

Za konec pa še nekaj primerov, ob katerih boste morda dobili kakšno idejo.

Stavki, ki se opisujejo z rimskimi števili: 1=I, 5=V, 10=X, 50=L, 100=C, 500=D, 1000=M.

**V tem stavku**
**I nastopa IX-krat,**
**V nastopa I-krat,**
**X nastopa II-krat,**
**L nastopa I-krat,**
**C nastopa I-krat,**
**D nastopa I-krat,**
**M nastopa I-krat.**

Stavki, ki se opisujejo z binarnimi števili, pri katerih številko 0 zamenjamo s črko *O* in številko 1 s črko *I*.

**V tem stavku I nastopa IOO krat, O nastopa II krat.**

**TA STAVEK VSEBUJE IIO A-JEV, IO B-JA, I C, I Č, I D, IIOO E-JEV, I F, I G, I H, IOOOIO I-JEV, IIOO J-JEV, IO K-JA, I L, I M, I N, IOOIO O-JEV, I P, I R, II S-JE, I Š, II T-JE, IO U-JA, IOOI V-JEV, I Z, I Ž.**

Za zadnji stavek poznam še eno rešitev, ki pa jo lahko poiščete tudi brez računalnika.

O stavkih, ki opisujejo sami sebe, je zanimivo in obširno pisal **Douglas R. Hofstadter** v rubriki **Metamagical Themas** revije **Scientific American**. Svoje prispevke je objavljal od januarja 1981 do julija 1983. Zbrani in dodatno opremljeni s komentarji so izšli v knjigi **Metamagical Themas**, ki jo je leta 1985 izdala založba Basic Books.

*Ciril Pezdir*