PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **26** (1998/1999)
Številka 4
Strani 204–208

Martin Juvan:

NENAVADNA FUNKCIJA Z RAČUNALNIKOM

Ključne besede: matematika, analiza, računalništvo, funkcije, naravna števila, rekurzija.

Elektronska verzija: http://www.presek.si/26/1376-Juvan.pdf

NENAVADNA FUNKCIJA Z RAČUNALNIKOM

V prejšnji številki Preseka (Nenavadna funkcija, Presek 26 št. 3, str. 166–169) smo spoznali nekatere zanimive lastnosti funkcije F, ki smo jo definirali rekurzivno z naslednjimi predpisi:

$$F(1) = 1 \quad \text{in} \quad F(2n) = F(n),\ F(2n+1) = F(n) + F(n+1) \text{ za } n \geq 1\,.$$

Tokrat si bomo ogledali nekaj računalniških programov, s katerimi si lahko pomagamo pri raziskovanju lastnosti te funkcije. S programi bomo tudi preverili odgovore na nekatera vprašanja, na katera smo s teoretičnim razmislekom odgovorili že v prejšnji številki Preseka.

Začnimo s programom v logu, s katerim za dano naravno število n izračunamo vrednost $F(n)$:

```
TO F :n
  IF :n < 2 [OUTPUT 1] ;baza rekurzije
  TEST (REMAINDER :n 2) = 0 ;Ali je argument sod?
  IFT [OUTPUT F :n/2] ;Da.
  IFF [OUTPUT (F (:n-1)/2) + (F (:n+1)/2)] ;Ne.
END
```

V gornjem programu računamo vrednost funkcije neposredno iz definicije. Pri taki neposredni uporabi rekurzivnih definicij moramo biti vedno previdni, saj se hitro primeri, da večkrat pokličemo funkcijo z istim argumentom. To pa pomeni odvečno delo in včasih je tega dodatnega dela toliko, da program postane praktično neuporaben. Ker je v našem primeru baza rekurzije le $F(1) = 1$, je pri izračunu vrednosti $F(n)$ funkcija F poklicana z argumentom $n = 1$ natanko $F(n)$-krat. Že pri izračunu $F(9)$ vrednost $F(1)$ računamo npr. 4-krat. Ker pa funkcija F pri majhnih številih ne zavzame velikih vrednosti, je za majhna števila napisani program še vedno dovolj učinkovit.

Z gornjim programom lahko tudi sestavimo tabelo vrednosti funkcije F za števila od 1 do 16. Vrednosti zlagamo v seznam `s`, tega pa na koncu izpišemo:

```
MAKE "s [] MAKE "i 0
REPEAT 16 [MAKE "i :i+1 MAKE "s LPUT F :i :s]
PRINT :s
```

Gornji ukazi tako izpišejo

```
1 1 2 1 3 2 3 1 4 3 5 2 5 3 4 1
```

S programom lahko tudi ugotovimo, koliko je $F(1999)$. Ukaz

```
print F 1999
```

izpiše 49, toliko, kolikor smo naračunali tudi v že omenjenem prispevku v prejšnji številki Preseka.

V nadaljevanju si bomo ogledali, kako lahko vrednosti funkcije F računamo še precej hitreje. Eden od možnih pristopov je, da vrednosti, ki jih izračunamo, sproti shranjujemo, npr. v tabelo. Poglejmo, kako na ta način s programom v programskem jeziku pascal poiščemo največjo vrednost, ki jo funkcija F zavzame na številih od 1 do 1999.

```
program NenavadnaFunkcija;
{ Poišče največjo vrednost funkcije F na številih od 1 do meja. }
  const
    meja = 1999;
  var
    F: array [1..meja] of integer;      { tabela vrednosti funkcije F }
    i: integer;
    max: integer;
begin
  { Izračunamo vrednosti funkcije F od 1 do meja. }
  F[1] := 1;
  for i:=2 to meja do
    if odd(i) then
      F[i] := F[i div 2] + F[i div 2 + 1]
    else
      F[i] := F[i div 2];
  { Med vrednostmi poiščemo največjo. }
  max := F[1];
  for i:=2 to meja do
    if F[i]>max then max := F[i];
  { Izpišemo podatke o največji vrednosti. }
  writeln('Najvecja vrednost funkcije od 1 do ',meja,' je ',max,'.');
  write('Ta vrednost je dosezena pri naslednjih stevilih:');
  for i:=1 to meja do
    if F[i]=max then write(i:5);
  writeln;
end.
```

Ko program poženemo, ta izpiše

```
Najvecja vrednost funkcije od 1 do 1999 je 144.
Ta vrednost je dosezena pri naslednjih stevilih: 1365 1707
```

Seveda so odgovori enaki tistim, ki smo jih s kar precej dolgim in zapletenim razmislekom našli v prejšnji številki Preseka.

Računanje zaporednih vrednosti funkcije s sprotnim shranjevanjem vrednosti v tabelo je učinkovit postopek, kadar želimo izračunati vrednosti funkcije pri prvih nekaj številih. Da dobimo novo vrednost, moramo le preveriti parnost argumenta in morebiti sešteti dve že znani vrednosti. Slaba stran postopka pa je, da potrebuje precej pomnilnika za tabelo vrednosti, kar omejuje njegovo uporabnost. Polovico tabele lahko prihranimo, če v njej hranimo le vrednosti funkcije pri lihih argumentih. Vrednost pri sodem argumentu potem dobimo tako, da argument delimo z 2, dokler je sod, ko pa postane lih, vrednost funkcije preberemo iz tabele. Na ta način nekoliko povečamo čas, ki ga potrebujemo za izračun naslednje vrednosti, hkrati pa razpolovimo količino potrebnega pomnilnika. Kodiranje programa, ki uporablja opisano spremembo, vam prepuščam za vajo.

Kako pa bi izračunali vrednost funkcije F pri velikih argumentih, recimo $F(1\,789\,569\,707)$ ali pa $F(2\,863\,311\,531)$? S shranjevanjem vrednosti v tabelo si ne moremo pomagati, saj bi potrebovali res ogromno tabelo, pa tudi izračun manjših vrednosti bi nam vzel kar nekaj časa. Pa saj nas prejšnje vrednosti sploh ne zanimajo. Poskusimo lahko s programom v logu, ki smo ga napisali na začetku prispevka, a po nekaj minutah obupamo, saj ni videti, da bi se računanje kaj kmalu končalo. No, res je, da je logo programski jezik, ki se tolmači. Programi v jezikih, ki se prevajajo, taka sta npr. pascal in C, so običajno bistveno hitrejši. Tako lahko rekurzivno definicijo funkcije zapišemo kot program v pascalu ali C-ju. Poskusite, gotovo vam bo uspelo. Če vas skrbi predolgotrajno računanje, lahko uporabite še kakšen dodatni "trik". V tabeli lahko pripravite vrednosti funkcije za "majhne" argumente, recimo do 10000, in te vnaprej izračunane vrednosti uporabite kot (razširjeno) bazo rekurzije. Poskusite lahko tudi s shranjevanjem vmesnih rezultatov. Vse vrednosti funkcije F, ki jih izračunate (tudi tiste z vmesnih korakov), skupaj s pripadajočim argumentom shranjujete v tabelo. Ko vas zanima vrednost $F(n)$, najprej pogledate po tabeli, ali ste to vrednost morda že izračunali. Tako zagotovite, da za vsak argument vrednost funkcije računate le enkrat (seveda pa se lahko zgodi, da vam čez čas po veliko opravljenih računih v tabeli zmanjka prostora).

Za konec prispevka pa si bomo ogledali tisto pravo metodo za računanje vrednosti funkcije F pri velikih argumentih. Postopek bo malce zvit, a preprost, zelo hiter in ne bo zahteval nobenega dodatnega prostora. Recimo, da bi radi izračunali vrednost $F(n)$. Iskano vrednost bomo zapisali

kot celoštevilsko linearno kombinacijo dveh zaporednih vrednosti funkcije F:

$$F(n) = a \cdot F(m) + b \cdot F(m+1)\,.$$

Na začetku vzamemo kar $m = n$, $a = 1$ in $b = 0$. Nato število m razpolovimo in izračunamo nova koeficienta. Pri tem ločimo dve možnosti. Če je število m sodo, $m = 2m'$, potem velja $F(m) = F(m')$ in $F(m+1) = F(m') + F(m'+1)$. Tako dobimo

$$F(n) = a \cdot F(m) + b \cdot F(m+1) = (a+b) \cdot F(m') + b \cdot F(m'+1)\,.$$

Če pa je število m liho, $m = 2m' + 1$, potem imamo $F(m) = F(m') + F(m'+1)$ in $F(m+1) = F(m'+1)$. Tedaj velja

$$F(n) = a \cdot F(m) + b \cdot F(m+1) = a \cdot F(m') + (a+b) \cdot F(m'+1)\,.$$

V obeh primerih lahko torej $F(n)$ zapišemo kot kombinacijo vrednosti $F(m')$ in $F(m'+1)$, pri čemer nova koeficienta na zelo preprost način izračunamo iz prejšnjih dveh.

Zapišimo program v programskem jeziku C, ki za izračun vrednosti funkcije F uporablja zgoraj opisani postopek.

```
#include <stdio.h>

/*
  Hiter izračun vrednosti funkcije F.
  Deluje hitro tudi za velike vrednosti argumenta.
*/

unsigned long F(unsigned long);

int main(void)
{
  unsigned long n; /* argument */

  do {
    printf("Vpisi vrednost argumenta (0 za konec): ");
    scanf("%lu", &n);
    if (n != 0) printf("F(%lu) = %lu\n\n", n, F(n));
  } while (n != 0);
  return 0;
}
```