Tvorba efektívnych algoritmov

Efektívny algoritmus je taký algoritmus, ktorého vykonávanie je čo najrýchlejšie a nároky na operačnú pamäť sú minimálne. V praxi je veľmi ťažké zladiť dohromady obidve kritéria. Väčšinou je algoritmus buď veľmi rýchly, ale využíva pomerne veľa premenných (hovoríme, že je časovo efektívny) alebo nie je čo najrýchlejší, ale zato všetky jeho premenné zaberajú minimálnu veľkosť operačnej pamäte (hovoríme, že je pamäťovo efektívny). Ak porovnávame dva algoritmy, ktoré riešia správne tú istú úlohu, potom je časovo efektívnejší ten, ktorého vykonávanie pre rovnaké vstupné údaje trvá kratšie. Pamäťovo efektívnejší bude zasa ten, ktorého všetky premenné zaberajú menšiu operačnú pamäť. Úprava algoritmu na efektívnejší tvar sa nazýva optimalizáciou.

Vy ste momentálne v štádiu, kedy vás vôbec netrápi efektivita navrhnutého algoritmu. Ste radi, ak vám program funguje a vypisuje správne výstupne údaje. Napokon, ešte sa musíme čo to naučiť, aby sme mohli jednu úlohu riešiť viacerými spôsobmi. Ale i napriek tomu poďme si povedať pár slov aj o tejto problematike tvorby algoritmov.

Ak program neobsahuje cyklus, výrazne sa jeho vykonávanie urýchliť nedá. V cykle sa všetko robí opakovane, takže každá optimalizácia príkazov tela cyklu znamená dosť veľkú časovú úsporu.

Časová a pamäťová zložitosť algoritmu sa nedá nikdy presne určiť, vždy sa odhaduje iba približne. Napr. ak potrebujeme vykonať cyklus N-krát, potom jeho časová zložitosť O je N.t, kde t je konštanta, ktorá predstavuje čas vykonania tela cyklu. Veľakrát sa konštanta t zanedbáva, a preto časová zložitosť takéhoto algoritmu je N. Ak by sme dokázali vyriešiť zadanú úlohu tak, že nám vystačí vykonať telo cyklu iba N/2-krát, potom získame efektívnejší algoritmu, ktorého časová zložitosť je N/2. Podobne určujeme pamäťovú náročnosť. Veľkosť jednoduchých premenných zvyčajne zanedbávame. Rozhodujúce sú pre nás veľkosti zložených údajových typov. Keďže zatiaľ o nich nič nevieme, nebudeme sa odhadom pamäťovej náročnosti zaoberať.

Príklad 1:

V súbore ***fyzika.pas*** je program, ktorý načíta čas odchodu a rýchlosť vlaku, ktorý odišiel skôr a čas odchodu a rýchlosť druhého vlaku, ktorý ide po rovnobežnej koľaji s prvým vlakom. Rýchlosť sa má udávať v km/h. Čas o koľkej hodine a o koľkej minúte. Program zistí, či sa vlaky dobehnú, za aký čas v hodinách, minútach a sekundách od odchodu neskoršieho vlaku a akú dráhu v km za tento čas prejdú od východzej stanice alebo či sa nedobehnú. Predpokladáme, že vlaky sú nákladné a nikde nestoja a rozdiel ich rýchlostí je väčší ako 10 km/h. V programe je príliš veľa premenných. Upravte ho tak, aby ste na správne vyriešenie tejto úlohy použili čo najmenej premenných.

Riešenie:

Obsah spomínaného súboru je nasledovný:

```
uses Crt;
var h1,m1,h2,m2,m,h:integer;
    t,t2,v1,v2,vs1,vs2,s,km:real;
    hod,min,sek:integer;
    doba,pom:longint;

begin
   clrscr;

   {nacitanie vstupnych udajov}
   writeln('Zadaj cas odchodu vlaku, ktory odisiel
skor ');
   write('     hodina odchodu: ');
   readln(h1);
   write('     minuta odchodu: ');
   readln(m1);
   write('Rychlost vlaku v km/h: ');
   readln(v1);

   writeln;
   writeln('Zadaj cas odchodu vlaku, ktory odisiel
neskor ');
   write('     hodina odchodu: ');
   readln(h2);
   write('     minuta odchodu: ');
   readln(m2);
   write('Rychlost vlaku v km/h: ');
   readln(v2);
   writeln;

   {vypocet doby, za ktoru sa vlaky stretnu}
   if v1<v2 then begin
      if h1>h2 then h2:=h2+24;
      h:=h2-h1;
      if m1>m2 then begin
          m2:=m2+60;
          dec(h);
          end;
      m:=m2-m1;
      t:=h*60+m;
```

```
      t:=t*60;
      vs1:=v1/3.6;
      vs2:=v2/3.6;
      t2:=vs1*t/(vs2-vs1);

      {prevod doby v sekundach na hodiny, minuty
a zvysne sekundy}
      doba:=round(t2);
      hod:=doba div 3600;
      pom:=hod;
      doba:=doba-pom*3600;
      min:=doba div 60;
      sek:=doba mod 60;

      {vypocet dlzky prejdenej drahy}
      s:=vs2*t2;
      km:=s/1000;
      writeln('Vlaky sa dobehnu za ',hod,' hod.,
',min,' min. a ',sek,' sek.');
      writeln('A prejdu od stanice ', km:8:3,' km');
   end
   else writeln('Vlaky sa nedobehnu, lebo vlak, ktory
vyrazil skor je rychlejsi');
   repeat until KeyPressed;
end.
```

Každá fyzikálna veličina má v programe svoju premennú. Počet premenných je 19, čo je veľmi veľa. Ak by sme prerobili príslušnú časť tohto programu tak, ako je napísané ďalej, potom ušetríme jednu premennú (premennú h) a správnosť programu sa nenaruší.

```
   {vypocet doby, za ktoru sa vlaky stretnu}
   if v1<v2 then begin
      if h1>h2 then h2:=h2+24;
      h1:=h2-h1;
      if m1>m2 then begin
          m2:=m2+60;
          dec(h1);
          end;
      m:=m2-m1;
      t:=h1*60+m;
      t:=t*60;
```

Skúste porozmýšľať a ušetriť podobným spôsobom ďalšie premenné. Ak ich ušetríte dokopy 10, potom bude váš program pamäťovo efektívny.

Príklad 2:

V predchádzajúcom príklade sme si ukázali pamäťovú optimalizáciu programu. V tomto príklade si ukážeme časovú optimalizáciu programu, ktorý načíta dve prirodzené čísla nerovné nule a vypíše ich najväčšieho spoločného deliteľa. Ako prvé by vás určite napadlo toto riešenie:

Riešenie 1:

```
uses Crt;
var a,b,nsd:integer;
begin
   clrscr;
   write('Zadaj prirodzene cislo: ');
   readln(a);
   write('Zadaj dalsie prirodzene cislo: ');
   readln(b);
   if a<b then
      begin
      nsd:=a;
      a:=b;
      b:=nsd;
      end;

   nsd:=b;
   while ((a mod nsd <> 0) or (b mod nsd <> 0)) and
(nsd>1) do dec(nsd);
   if nsd=1 then writeln('Zadane cisla nemaju
spolocneho delitela')
      else writeln('Najvacsi spolocny delitel
zadanych cisel je ',nsd);
   readln;
end.
```

Pre dvojicu čísel 24 a 42 by sme vykonali telo cyklu 18-krát. Pre dvojicu čísel 12 a 7 by sme vykonali telo cyklu 6-krát. Skúsme nájsť iné riešenie.

Riešenie 2:

Využime pre hľadanie NSD Newtonov algoritmus. Jeho podstata je veľmi jednoduchá a matematicky by sme ju mohli zapísať takto:

```
            pre a<b je NSD(a,b-a)
NDS(a,b) =  pre a=b je a
            pre a>b je NSD(a-b,b)
```

Ukážme si to na príklade čísel 15 a 18.

```
   a=15, b=18,          keďže 15<18, tak
   a=15, b=18-15=3      keďže 15>3, tak
   a=15-3=12, b=3       keďže 12>3, tak
   a=12-3=9, b=3        keďže 9>3, tak
   a=9-3=6, b=3         keďže 6>3, tak
   a=6-3=3, b=3         keďže 3=3, tak NSD(15,18)=3
```

Program Newtonovho algoritmu pre výpočet NSD vyzerá takto:

```
uses Crt;
var a,b:integer;
begin
   clrscr;
   write('Zadaj prirodzene cislo: ');
   readln(a);
   write('Zadaj dalsie prirodzene cislo: ');
   readln(b);

   while (a<>b) do
      begin
      if a>b then a:=a-b
             else b:=b-a;
      end;

   if   a=1   then   writeln('Zadane   cisla   nemaju
spolocneho delitela')
         else   writeln('Najvacsi   spolocny   delitel
zadanych cisel je ',a);
   readln;
end.
```

Pre dvojicu čísel 24 a 42 by sme vykonali telo cyklu 4-krát. Pre dvojicu čísel 12 a 7 by sme vykonali telo cyklu 5-krát. Toto riešenie je oproti predošlému nielen časovo efektívnejšie, ale aj pamäťovo, lebo sme ušetrili jednu premennú.