PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **23** (1995/1996)
Številka 2
Strani 78–82

Darko Zupanič:

NAPAKE V PRIKAZOVANJU ŠTEVK

Ključne besede: računalništvo, računalniško programiranje, pascal.

Elektronska verzija: http://www.presek.si/23/1259-Zupanic.pdf

© 1995 Društvo matematikov, fizikov in astronomov Slovenije
© 2010 DMFA – založništvo

Vse pravice pridržane. Razmnoževanje ali reproduciranje celote ali posameznih delov brez poprejšnjega dovoljenja založnika ni dovoljeno.

NAPAKE V PRIKAZOVANJU ŠTEVK

Na 19. državnem tekmovanju iz računalništva za srednješolce je bila tekmovalcem zastavljena tudi naslednja naloga:

> Na zaslonu je niz sedem-segmentnih (digitalnih) števk dolžine N. Predstavitev števk s segmenti je naslednja: 0 1 2 3 4 5 6 7 8 9. Za prižgane segmente vemo, da so pravilni, medtem ko za ugasnjene segmente ne vemo, ali so pravilni ali pokvarjeni. Poleg tega vemo, da je vsota števk deljiva z 10.
>
> Opiši postopek, ki najde tak niz vrednosti števk, da ustrezajo omejitvam in imajo najmanj popravljenih segmentov.
>
> Primer: 3 7 ima možni rešitvi 3-7 in 8-2. Rešitev je 3-7, ker ima samo en popravljen segment, medtem ko ima 8-2 pet popravljenih segmentov.

Enostavno rešitev problema ponujajo mehanizmi, ki se uporabljajo v logičnem programiranju z omejitvami (Constraint Logic Programming – CLP). CLP je področje umetne inteligence, ki je nastalo z zlitjem področij logičnega programiranja (jezik logičnega programiranja je prolog) in razreševanja omejitev. Kot primer razreševanja omejitev lahko vzamemo reševanje linearnih enačb. Če sistemu damo enačbo $Y = 2X + 3$, si to zapomni kot omejitev (vrednost spremenljivke Y je omejena z vrednostjo spremenljivke X in obratno). Če vnesemo še enačbo $Y = -X+6$, se sproži razreševalec omejitev, ki poskuša čimbolj poenostaviti dane omejitve. V našem primeru sta rezultat poenostavitve kar vrednosti $Y = 5$ in $X = 1$.

Rešitev problema bo prikazana v jeziku pascal. Ker pascal ne vsebuje mehanizmov, ki omogočajo avtomatično razreševanje omejitev, bomo za nekatere podprograme predpostavili, da so že napisani. Prav tako bomo predpostavili obstoj nekaterih podprogramov in funkcij, ki se ukvarjajo z 'malenkostmi'.

Delovanje podprogramov, ki se ukvarjajo z dodajanjem in razreševanjem relacij (omejitev) med spremenljivkami, si lahko predstavljamo kot določanje njihove zaloge vrednosti. Vsako dodajanje relacije med spremenljivkami ustrezno uskladi (doda, odvzame, ohrani) možne vrednosti vseh spremenljivk, ki nastopajo v relacijah.

Primer: Naprej dodajmo relaciji $X < Y$ in $Y < Z$. Ob vnosu dodatne relacije $Z < 5$ se avtomatično zmanjša zaloga vrednosti za spremenljivko Y na števila manjša od 4 in za spremenljivko X na števila manjša od 3. Seveda pa so celoštevilske omejitve samo najenostavnejša vrsta omejitev, ki nastopajo v realnih problemih.

Dejanska rešitev je bila napisana v sistemu za logično programiranje z omejitvami ECLiPSe, iz katerega je vzet tudi potek reševanja našega primera.

Pri reševanju bomo potrebovali naslednje konstante, tipe in spremenljivke:

```
const
  MxSegment = 7;      {vsaka številka je sestavljena iz 7 segmentov}
  MxStevka = 9;                          {možne številke so od 0 do 9}
type
  SegmentDef = 1..MxSegment;
  StevkaDef = 0..MxStevka;
  MnozicaStevkDef = set of StevkaDef;
  SegmentiDef = array [ SegmentDef ] of boolean;
  MnozicaCelihStevil = set of integer;
var
  MozneStevke : array [ 1..N ] of MnozicaStevkDef;
  VsotaMoznihStevk : MnozicaCelihStevil;
  Razlika : array [ 1..N, StevkaDef ] of 0..MxSegment;
  VsotaRazlik, MinimalnaRazlika : MnozicaCelihStevil;
  VrednostiStevk, Resitev : array [ 1..N ] of StevkaDef;
```

Podatke o številkah na zaslonu in njihovi predstavitvi dobimo z naslednjimi podprogrami:

`function StevkeSegmenta ( Segment : SegmentDef ) : MnozicaStevkDef;`
- funkcija *StevkeSegmenta* vrne množico številk, ki so možne, kadar je segment *Segment* prižgan. Tako so pri prižganem zgornjem segmentu možne številke 0, 2, 3, 5, 6, 7, 8, in 9.

`function Segmenti ( i : integer ) : SegmentiDef;`
- funkcija *Segmenti* vrne tabelo vrednosti (prižgan/ugasnjen) segmentov i-te številke na zaslonu.

```
function RazlicniSegmenti ( Segmenti : SegmentiDef;
  Stevka : StevkaDef ) : SegmentDef;
```
- funkcija *RazlicniSegmenti* vrne število različnih segmentov v *Segmenti* in *Stevka*.

Za razreševanje omejitev bomo uporabljali naslednje podprograme:

```
procedure Presek ( var MnozicaStevk1 : MnozicaStevkDef;
  MnozicaStevk2 : MnozicaStevkDef );
```
- podprogram *Presek* naredi relacijo (omejitev) *presek množic* med *MnozicaStevk1* in *MnozicaStevk2*.

```
procedure Pristej ( var Vsota : MnozicaCelihStevil;
  Stevilo : MnozicaStevkDef );
```
– podprogram *Pristej* naredi relacijo (omejitev) *vsota* med *Vsota* in *Stevilo*.

```
procedure Mod ( var Deljenec : MnozicaCelihStevil;
  Delitelj, Ostanek : integer );
```
– podprogram Mod naredi relacijo (omejitev) *ostanek pri deljenju* med *Deljenec*, *Delitelj* in *Ostanek*.

```
procedure Manjsi ( var i, j : MnozicaCelihStevil );
```
– podprogram *Manjsi* naredi relacijo (omejitev) *manjši* med *i* in *j*.

Da rešitev ne bo predolga, bomo uporabili še naslednji podprogram:

```
function IzberiVrednost ( var MnozicaStevk : MnozicaStevkDef ) :
  StevkaDef;
```
– podprogram *IzberiVrednost* vrne eno izmed vrednosti v *MnozicaStevk* ter jo iz nje izloči.

Problem rešujemo v dveh fazah. Najprej opišemo omejitve problema, nato pa priredimo vrednosti posameznim spremenljivkam.

```
procedure StaticneOmejitve;
begin
  for i := 1 to N do begin
    MozneStevke [ i ] := [ StevkaDef ];
    for j := 1 to MxSegment do begin
      Presek ( MozneStevke [ i ], StevkeSegmenta ( j ));
    end;
  end;

  VsotaMoznihStevk := [ 0 ];
  for i := 1 to N do begin
    Pristej ( VsotaMoznihStevk, MozneStevke [ i ] );
  end;
  Mod ( VsotaMoznihStevk, 10, 0 );

  for i := 1 to N do begin
    for j := 0 to MxStevka do begin
      Razlika [ i, j ] := RazlicniSegmenti ( Segmenti ( i ), j );
    end;
  end;
  VsotaRazlik := [ 0 ];
  for i := 1 to N do begin
    Pristej ( VsotaRazlik, Razlika [ i, VrednostiStevk [ i ]]);
  end;
end;
```

Problem vsebuje tri vrste omejitev.

Vsak prižgan segment določa možne števke. Če naredimo presek možnih števk za vse segmente, dobimo kandidate za posamezno števko zaslona. Za primer iz besedila naloge dobimo možne vrednosti števk [3, 8, 9] in [0, 2, 3, 7..9].

Naslednja omejitev je vsota števk, ki mora biti deljiva z 10. Naj opozorimo, da s podprogramoma *Pristej* in *Mod* le določimo relacijo med spremenljivkami. V našem primeru je *VsotaMoznihStevk* lahko le med [3..18] in mora biti deljiva z 10. Od tod razreševalec omejitev lahko sklepa, da je *VsotaMoznihStevk* enaka 10, in možne vrednosti števk skrči na [3, 8] in [2, 3, 7]. Kot vidimo, uporabljeni sistem za razreševanje omejitev ne zna vedno sklepati do konca (ne izloči 3 za drugo števko). Vendar pa je bistveno, da ne izloči kakšne vrednosti preveč.

Zadnja vrsta omejitev je povezana s pogojem, da je rešitev tisti niz števk, ki vsebuje najmanj popravljenih segmentov. Za vsako možno vrednost števke na izbranem mestu zaslona preštejemo število segmentov, ki jih je potrebno ugasniti, da dobimo vzorec na zaslonu. Omejitev je vsota popravkov na vseh mestih zaslona. To vsoto kasneje minimiziramo. Zopet samo vzpostavimo relacije (omejitve). Vsota razlik še ni znana, saj temelji na *VrednostiStevk*, ki jih še nismo določili. Vendar pa razreševalec omejitev lahko naredi nekatere sklepe. Razlika na prvi števki je lahko [0..2] (tokrat ne izloči 1), na drugi pa [1, 3..5] (ne izloči 4 in 5), kar pomeni, da je *VsotaRazlik* med [1..7].

```
procedure PrirediVrednosti ( i : integer );
begin
  if i = N then begin
    Resitev := VrednostiStevk;
    MinimalnaRazlika := VsotaRazlik;
  end else begin
    while MozneStevke [ i ] <> [] do begin
      VrednostiStevk [ i ] := IzberiVrednost ( MozneStevke[i] );
      Manjsi ( VsotaRazlik, MinimalnaRazlika );
      PrirediVrednosti ( i + 1 );
    end;
  end;
end;
```

Sedaj lahko preidemo v drugo fazo, kjer prirejamo možne vrednosti posameznim števkam. Postopek je tak, da vsaki števki prirejamo samo vrednosti, ki ustrezajo omejitvam. Po vsaki prireditvi dodamo omejitev, ki preveri, če je *VsotaRazlik* manjša kot *MinimalnaRazlika*. Naj še enkrat opozorimo, da so samo nekateri členi vsote znani in da je *VsotaRazlik* pravzaprav spodnja meja. Z vsako prireditvijo vrednosti posamezni števki

se sproži razreševalec omejitev, ki zmanjšuje množice možnih vrednosti še nedoločenim števkam. V našem primeru se prvi števki priredi vrednost 3. To sproži razreševalec omejitev, ki sedaj natančno ugotovi, da je možna vrednost za drugo števko 7 in da je *VsotaRazlik* enaka 1. To postane trenutno najboljša rešitev (*MinimalnaRazlika*). Ker dodamo omejitev, da mora biti *VsotaRazlik* manjša od *MinimalnaRazlika*, in od prej velja, da je *VsotaRazlik* [1..7], razreševalec omejitev javi, da ni boljše rešitve. Rešitev torej dobimo z enim prirejanjem vsaki števki, kar je bistveno bolje od 100 ($10^N, N = 2$) prirejanj, ki bi jih bi potrebovali pri najenostavnejši rešitvi problema.

V praksi je veliko problemov, ki so podobni naši nalogi: zgradba šolskega urnika, načrtovanje zaporedja operacij, razni razrezi materialov, ... Za vse take probleme je značilno, da so rešitve znotraj nekih omejitev. Poleg tega imamo še kriterijsko funkcijo, na podlagi katere izločimo najboljšo rešitev. Npr. na čim manjšem kosu materiala (kriterijska funkcija) moramo izrezati razne elemente (določene z omejitvami).

Principi in mehanizmi, ki smo jih uporabljali pri reševanju zastavljene naloge, nam v teoretičnem smislu sicer ne odpravijo problema, vendar pa se v praksi (npr. naš primer) izkaže, da lahko z njimi večkrat rešimo probleme bistveno hitreje kot s klasičnimi prijemi.

Darko Zupanič