PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **20** (1992/1993)
Številka 2
Strani 78–84

Martin Juvan:

NAJVEČJA LUKNJA

Ključne besede: matematika, računalništvo, algoritmi, programiranje, pascal.

Elektronska verzija: http://www.presek.si/20/1127-Juvan.pdf

RAČUNALNIŠTVO

NAJVEČJA LUKNJA

Kontrabantar Prebrisani Pepe je zopet pred velikim poslom. Odločil se je, da bo v temnih nočeh skozi Mračno dolino čez mejo spravljal dragocen tovor. Toda tudi financarji niso od muh in mu pripravljajo zasede. Ker država ni preveč bogata, plače financarjev pa nizke, se Prebrisani Pepe zlahka dokoplje do razporeda čakajočih financarjev. Takole piše:

Klas Miha 132
Stotin Lojze 62
Tolar Polde 218
⋮

Na koncu seznama je še pripisano, da financarji čakajo v ravni vrsti, števila pa pomenijo oddaljenost v metrih čakajočih financarjev od desnega roba doline. Financarji tudi vedno postavijo po enega moža na levi in desni rob. Toda Prebrisani Pepe ne bi bil prebrisan, če ne bi poznal še nekaterih drugih podatkov. Tako ve, kako široka je Mračna dolina, pa tudi, da ga v temni noči financarji lahko opazijo le, če se jim približa na manj kot dvajset metrov. Če pa ga opazijo, potem mu ni rešitve, saj jim s tovorom ne more uiti. Ker je Prebrisani Pepe tudi ljubitelj računalnikov, se je odločil, da sestavi program, ki mu bo vsak večer povedal, ali naj se to noč odpravi čez mejo. Sestavil je takle algoritem:

```
program Luknja1;
{ Pove, ali bo imel Pepe delovno noč. }
  const
    Max_Fin = 30;                          { največje število financarjev }
    Sirina = 666;                                  { širina Mračne doline }
    Vidljivost = 20;                       { Kako daleč vidijo financarji? }
  ftype
    tabela = array [1..Max_Fin] of integer;
    zasedeno = array [1..Sirina] of boolean;
  var
    A: tabela;
    Z: zasedeno;
    i,n: integer;
    NajLuknja,TrenLuknja: integer;
begin
  write('Koliko financarjev je to noc na delu : '); readln(n);
  for i:=1 to n do begin                        { vnos položajev financarjev }
    write('Polozaj ',i,'. financarja : '); readln(A[i]);
  end; {for}
```

```
  for i:=1 to Sirina do Z[i]:=false;
  for i:=1 to n do
    Z[A[i]]:=true;                    { Postavimo financarje na njihova mesta. }
{ Pogledamo, kako veliko luknjo so to noč pustili financarji. }
  NajLuknja:=0; TrenLuknja:=1;
  for i:=1 to Sirina do
    if not Z[i] then TrenLuknja:=TrenLuknja+1
    else begin
      if TrenLuknja>NajLuknja then NajLuknja:=TrenLuknja;
      TrenLuknja:=1;
    end;
  if NajLuknja>2*Vidljivost then          { Ali je luknja dovolj velika? }
    writeln('Nocojsnja noc bo delovna.')
  else
    writeln('To noc bos lahko pocival.');
  readln;
end.
```

Pa naj bo dovolj zgodbic o kontrabantu in oblasti. Poglejmo raje, s kakšnim problemom smo se pravzaprav srečali. Naj bo dana končna množica števil. Označimo jo z A. Števili iz množice imenujemo *sosednji* (glede na množico A), če med njima po velikosti ni nobenega drugega števila iz množice A. Če pogledamo nekoliko abstraktneje, smo rešili naslednjo nalogo:

Poišči največjo razliko med dvema sosednjima številoma iz množice A.

Množico A predstavimo s tabelo števil, ki pa seveda ni urejena. Ogledali si bomo štiri rešitve gornjega problema:

1. rešitev: Prvo rešitev smo že spoznali. Postopek ni predolg in tudi ne prezapleten, tako da ga zlahka razumemo. Ideja je preprosta. Množico A, ki je najprej predstavljena z neurejeno tabelo števil, predstavimo raje s tabelo logičnih vrednosti, torej tako, kot tudi pascal predstavlja množice. Tak pristop ima tudi veliko pomanjkljivost. Količina dodatnega prostora, ki ga potrebujemo poleg vhodnih podatkov, je odvisna tudi od vrednosti vhodnih podatkov, ne le od njihovega števila. Tako je velikost dodatne logične tabele sorazmerna z razliko med največjim in najmanjšim številom v množici A. Ta razlika pa je lahko zelo velika tudi v primeru, ko je moč množice A majhna. Prav tako je tudi čas, potreben za rešitev problema s tem postopkom, sorazmeren z razliko med največjim in najmanjšim številom v množici. Prva rešitev je tako primerna le za množice, pri katerih je razlika med največjim in

najmanjšim številom majhna. Dodatna pomanjkljivost te rešitve je še, da je ne moremo uporabiti pri množicah realnih števil.

Opisane pomanjkljivosti nas vzpodbudijo, da poskušamo poiskati rešitev, pri kateri bosta količina dodatnega prostora in čas reševanja odvisna le od moči množice A, ne pa tudi od vrednosti samih števil.

Pri programiranju bomo privzeli naslednji deklaraciji:

```
const
  Max_n = 10000;
type                          { Seveda lahko vzamemo tudi celoštevilsko tabelo. }
  tabela = array [1..Max_n] of real;
```

2. rešitev: Ta rešitev se povsem izogne težavam s prostorom, ki smo jih srečali pri prejšnji rešitvi, pa tudi njena časovna zahtevnost je zadovoljiva. Ideja je takale. Za vsak par števil iz množice A pogledamo, če sta števili sosednji, in če sta, je absolutna vrednost njune razlike kandidat za končni rezultat. Trenutno največjo razliko ves čas vodimo v spremenljivki NajLuknja. V njej je na koncu tudi rezultat, ki ga vrne funkcija.

```
function Luknja2(var A: tabela; n: integer): real;
  var
    i,j,k: integer;
    NajLuknja: real;
    Sosednji: boolean;
begin {Luknja2}
  NajLuknja:=0;
  for i:=1 to n-1 do
    for j:=i+1 to n do begin          { Ali sta števili A[i] in A[j] sosednji? }
      Sosednji:=true;
      k:=1;
      while Sosednji and (k<=n) do begin
        if (A[k]-A[i])*(A[k]-A[j])<0 then Sosednji:=false;
        k:=k+1;
      end; {while}
      if Sosednji and (abs(A[i]-A[j]))>NajLuknja then
        NajLuknja:=abs(A[i]-A[j]);
    end; {for}
  Luknja2:=NajLuknja;
end; {Luknja2}
```

Kaj ugotovimo s pogojem

$$(A[k]-A[i])*(A[k]-A[j])<0 \ ,$$

se sprašujete? Ta pogoj je izpolnjen natanko tedaj, ko število *A[k]* leži strogo med številoma *A[i]* in *A[j]*. Ta rešitev je varčna z dodatnim prostorom, saj

smo uporabili le pet dodatnih spremenljivk in nobene nove tabele. Tudi čas iskanja rešitve je odvisen le od števila podatkov. Žal pa je ta odvisnost kubična, kar pomeni, da se čas računanja veča sorazmerno s tretjo potenco števila podatkov. Če dolžino vhodne tabele A podvojimo, se čas reševanja pomnoži z osem. Ko tako funkcijski podprogram *Luknja2* preizkusimo na tabeli z nekaj sto elementi, hitro ugotovimo, da je še vedno zelo počasen. (Če se je bralec morda ustrašil, da bo moral sam odtipkati nekaj sto števil, da bo zadovoljivo preizkusil podprogram, mu svetujem, naj napiše podprogram, ki bo v vhodno tabelo A zapisal želeno število naključnih števil in tako opravil pripravo vhodnih podatkov namesto njega. V Turbo pascal vgrajena podprograma *randomize* in *random* mu bosta pri tem v veliko pomoč.)

3. rešitev: Kaj kompliciram, pravite! Rešitev naloge je na dlani. Tabelo števil, ki predstavlja množico A, uredimo, nato pa med pari zaporednih števil poiščemo tisti dve, med katerima je največja razlika. Tule je podprogram:

```
function Luknja3(var A: tabela; n: integer): real;
  var
    NajLuknja: real;
    i: integer;
begin {Luknja3}
  Uredi(A,n);                                { Nepadajoče uredimo tabelo A. }
  NajLuknja:=0;      { Poiščemo največjo razliko med pari zaporednih števil. }
  for i:=1 to n-1 do
    if A[i+1]-A[i]>NajLuknja then NajLuknja:=A[i+1]-A[i];
  Luknja3:=NajLuknja;
end; {Luknja3}
```

Kvaliteta zgornje rešitve je seveda odvisna od prave izbire podprograma za urejanje števil. O metodah za urejanje je napisanih kopica knjig. Najlažje dostopna je verjetno znamenita Wirthova knjiga Računalniško programiranje. V njenem drugem delu je narejen pregled standardnih postopkov za urejanje. Solidna metoda je hitro urejanje (quicksort po angleško) z nekaj izboljšavami (nerekurzivna varianta, urejanje kratkih podzaporedij s preprostim vstavljanjem, pazljiva izbira delilnega elementa), v zadnjem času pa se uveljavlja tudi izboljšano urejanje s kopico (več o njem si bralec lahko prebere v Obzorniku za matematiko in fiziko, letnik 38, številka 2, strani 45 do 52). Časovna zahtevnost vseh teh metod pa je reda vsaj $n \cdot \log n$. To pomeni, da se čas, potreben za rešitev problema, povečuje nekoliko hitreje, kot raste velikost vhodnih podatkov.

Toda naš problem se zdi vendarle preprostejši, kot je urejanje celotne

tabele. Sestavljanje postopka, ki reši nalogo v času, sorazmernem z velikostjo vhodne tabele, je tako pravi izziv za vsakega nadebudnega programerja.

4. rešitev: Priznati moram, da sem ves čas mislil na tole rešitev. Morda bo koga presenetilo, da si pri njeni izpeljavi pomagamo z varianto Dirichletovega principa. Ta pravi tole: če n stvari, v našem primeru števil, porazdelimo v $n+1$ predalčkov, v našem primeru intervalov, ostane vsaj eden od predalčkov prazen. Postopek reševanja je naslednji:

1. Poiščemo najmanjše in največje število v tabeli in ju shranimo v spremenljivki *min* in *max*.
2. Interval med številoma *min* in *max* razdelimo na $n+1$ enako dolgih podintervalov. Dolžina posameznega podintervala je torej $\frac{max-min}{n+1} =: d$.
3. Za vsakega od podintervalov ugotovimo, ali je v njem katero od števil iz vhodne tabele. Za vsak neprazen podinterval nato poiščemo najmanjše in največje število v njem.
4. Ker je podintervalov $n+1$, vhodnih števil pa je le n, bo vsaj en podinterval ostal prazen. Ker pa mejna podintervala nista prazna, bo največja razlika med sosednjima številoma večja od dolžine podintervala d. Torej bosta števili, ki določata najbolj oddaljen sosednji par, pripadali različnima podintervaloma. Da dobimo pravilen rezultat, tako zadošča za vsak neprazen podinterval pregledati razliko med najmanjšim številom v tem podintervalu in največjim številom v prvem nepraznem podintervalu na levi, torej med podintervali, ki vsebujejo manjša števila.

Vsakega od štirih korakov bomo sprogramirali tako, da bo porabljeni čas sorazmeren z dolžino vhodne tabele. Ker bomo potrebovali dve pomožni tabeli, podprogram pa želimo uporabljati tudi pri velikih vhodnih tabelah, bomo nekaterim pomožnim spremenljivkam prostor dodelili šele med izvajanjem.

```
function Luknja4(var A: tabela; n: integer): real;
  type
    intervali = array[0..Max_n] of real;
    prazno = array[0..Max_n] of boolean;
  var
    MinI,MaxI: ↑intervali;
    P: ↑prazno;
    Min,Max,d,NajLuknja: real;
    i,j: integer;
begin {Luknja4}
  Min:=A[1];                { Poiščemo najmanjše in največje število v tabeli A. }
  Max:=A[1];
  for i:=2 to n do begin
```

```
    if Min>A[i] then Min:=A[i];
    if Max<A[i] then Max:=A[i];
  end; {for}
{ Če so vsa števila v tabeli enaka, potem končamo. }
  if Min=Max then begin Luknja4:=0; exit; end;
{ Sicer pa izračunamo dolžino podintervalov in pripravimo pomožni tabeli. }
  d:=(Max-Min)/(n+1);
  GetMem(MinI,SizeOf(real)*(n+1));
  GetMem(MaxI,SizeOf(real)*(n+1));
  GetMem(P,SizeOf(boolean)*(n+1));
  for i:=0 to n do P↑[i]:=true;      { Na začetku so vsi podintervali prazni. }
{ S sprehodom po tabeli A pregledamo vsa števila in jih porazdelimo po}
{ ustreznih podintervalih. }
  for i:=1 to n do begin
  { Izračunamo, v kateri podinterval spada število A[i]. }
  { Če je število na meji dveh podintervalov, je vseeno, kam ga uvrstimo. }
    j:=trunc((A[i]-Min)/d); if j=n+1 then j:=n;
    if P↑[j] then
      begin P↑[j]:=false; MinI↑[j]:=A[i]; MaxI↑[j]:=A[i] end
    else
      begin {else}
        if A[i]<MinI↑[j] then MinI↑[j]:=A[i];
        if A[i]>MaxI↑[j] then MaxI↑[j]:=A[i];
      end; {else}
  end; {for}
  i:=0; NajLuknja:=0;                    { Poiščemo največjo luknjo. }
  j:=1;                  { j bo indeks naslednjega neprazнega podintervala. }
  while i<n do begin              { Poiščemo naslednji neprazni podinterval. }
    while P↑[j] do inc(j);      { Novo luknjo primerjamo z dosedaj največjo. }
    if MinI↑[j]-MaxI↑[i]>NajLuknja then
      NajLuknja:=MinI↑[j]-MaxI↑[i];
    i:=j;
    inc(j);
  end; {while}
  Luknja4:=NajLuknja;
end; {Luknja4}
```

Na koncu si oglejmo še rezultate meritev, ki smo jih dobili s primerjavo opisanih rešitev. Prikazuje jih tabela na naslednji strani. Preizkusi so bili opravljeni na običajnem AT računalniku. Izračunani časi so povprečje nekaj meritev in so podani v stotinkah sekunde.

Kot smo predvidevali, je druga rešitev bistveno slabša od tretje in četrte, čeprav se zdi, da je njena povprečna časovna zahtevnost nekoliko boljša od kubične. Zadnja rešitev je boljša od tretje, njena prednost pa postaja tem bolj očitna, čim večja je tabela, s katero delamo. Seveda pa moramo to večjo hitrost reševanja problema tudi plačati. Tako zadnja rešitev potrebuje

n	Luknja2	Luknja3	Luknja4
100	575	0	0
200	3037	5	5
300	7218	11	11
1000	–	39	39
3000	–	124	110
5000	–	223	181
10000	–	500	357

precej več pomnilnika kot tretja. Na celoštevilskih tabelah smo primerjali tudi prvo in zadnjo rešitev. Izkaže se, da sta približno enako dobri takrat, ko je razmerje med dolžino tabele in intervalom, s katerega so vrednosti, okoli ena proti dvajset (to razmerje je seveda odvisno tudi od tipa računalnika, na katerem opravljamo meritve). Če je razmerje manjše, je primernejša zadnja rešitev, sicer pa prva.

Če imamo opraviti le s celoštevilskimi tabelami, lahko zadnjo rešitev še nekoliko izpilimo, tako da uporablja le celoštevilsko aritmetiko (in morda še nekoliko več dodatnega prostora). Ker pa je prispevek že precej dolg, to izboljšavo prepuščamo zvestim bralcem za (neobvezno) domačo nalogo.

Martin Juvan

DVE IGRI S ŠTEVILI - Rešitev s str. 5

1. Kdor začne, ta lahko zmaga. Na prvem koraku napiše 6. Drugi lahko zapiše eno od števil iz naslednjih parov: $(4,5),(7,9),(8,10)$. Prvi v odgovor na poljubno potezo drugega zapiše drugo število iz para.

2. Vsota je 30.
   Prvi igralec drugemu ne more preprečiti, da bi imeli števili 19 in 20 enaka predznaka. Ko prvi igralec izbere predznak pred enim od števil v parih $(1,2),(3,4),\ldots,(17,18)$, izbere drugi nasprotni predznak pred drugim