PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **30** (2002/2003)
Številka 1
Strani 16–21

Martin Juvan:

PROGRAMERSKE PRIGODE

Ključne besede: računalništvo, programiranje, pitagorejske trojke.

Elektronska verzija: http://www.presek.si/30/1502-Juvan.pdf

PROGRAMERSKE PRIGODE

Včasih, ko mi čas dopušča, rešujem programerske naloge z različnih študentskih in srednješolskih tekmovanj. Že kar nekaj časa naloge izbiram s strežnika Univerze v Valladolidu v Španiji, ki je dosegljiv na naslovu

`http://acm.uva.es/problemset/`

Na strežniku pa ni samo obsežna zbirka nalog. Na njem je nameščen tudi Sodnik – posebna programska oprema, ki omogoča, da lahko po elektronski pošti pošljemo svoje rešitve nalog v (avtomatsko) preverjanje.

Pri sestavljanju rešitev, ki jih pošiljamo v preverjanje Sodniku, se moramo držati nekaterih pravil. Vhodne podatke vedno beremo s standardnega vhoda (to je običajno tipkovnica, operacijski sistemi pa omogočajo, da ob zagonu programa za standardni vhod uporabimo npr. datoteko), izpisovanje pa poteka na standardni izhod. Oblika vhodnih in izhodnih podatkov je natančno opisana v besedilu naloge in se je moramo strogo držati. Običajno Sodnik rešitev preveri tako, da prejeti program (ta mora biti napisan v jeziku C, C++, pascal ali java) prevede, nato pa ga požene na nam neznanih vhodnih podatkih. Odgovore, ki jih izpiše naša rešitev, primerja s tistimi, ki jih ima shranjene v svoji bazi pravilnih rešitev. Če se ujemajo, je rešitev sprejeta, sicer je zavrnjena. Pri nalogah, ki dopuščajo več različnih rešitev, je preverjanje bolj zapleteno in vključuje tudi posebne programe, napisane posebej za preverjanje rešitev takih nalog.

Sodnik postavlja še nekaj drugih omejitev. Velikost programov, ki jih lahko pošljemo v presojo, je omejena na 40000 znakov. Sodnik tudi ni "neskončno" potrpežljiv. Ko požene prevedeno rešitev, na odgovore čaka le določen čas (ta je v splošnem odvisen od naloge). Včasih je bil ta čas pri večini nalog enak 90 sekund, po posodobitvi Sodnika pa so ga skrajšali na 30 sekund. Torej: rešitve, ki jih pripravimo, morajo delovati povsem pravilno, ne smejo biti preveč "razvlečene", pa še dovolj hitro morajo najti odgovore (kar včasih ni prav preprosto).

Pred časom sem imel precej opraviti z nalogo številka 106 (na Sodniku so jo poimenovali *Fermat vs. Pythagoras*). Gre za ne prezahtevno matematično nalogo o pitagorejskih trojkah. Spomnimo se, da trojici naravnih števil x, y, z pravimo *pitagorejska trojka*, če zadošča enakosti $x^2 + y^2 = z^2$. Trojka je *primitivna*, če so si števila x, y in z paroma tuja. Kratek razmislek pokaže, da za primitivnost trojke zadošča, da sta si tuji le števili x in y. Najbolj znana pitagorejska trojka je gotovo $x = 3$, $y = 4$ in $z = 5$: $3^2 + 4^2 = 5^2$. Ta trojka je tudi primitivna.

Opis naloge. Naloga zahteva, da napišemo program, ki bo z vhoda bral naravna števila in za vsako prebrano število n na izhod v svojo vrstico izpisal dve števili. Prvo število mora biti število primitivnih pitagorejskih trojk x, y, z, za katere je $1 \leq x < y < z \leq n$; drugo število pa pove, koliko števil iz množice $\{1, 2, \ldots, n\}$ ne nastopa v nobeni pitagorejski trojki x, y, z, ki zadošča omejitvi $1 \leq x < y < z \leq n$. Branje se konča, ko na vhodu zmanjka števil (pogoj *end of file*). Na primer, če so na vhodu števila

```
10
25
100
```

mora program izpisati

```
1 4
4 9
16 27
```

Da bomo nalogo bolje razumeli, poglejmo, kako iz števila $n = 10$ pridemo do odgovorov 1 in 4. S poskušanjem hitro ugotovimo, da je že omenjena trojica $x = 3$, $y = 4$, $z = 5$ edina primitivna pitagorejska trojka, ki izpolnjuje zahtevo $1 \leq x < y < z \leq 10$ (trojka $x = 6$, $y = 8$, $z = 10$ tudi ustreza neenakostim, a ni primitivna). Drugi odgovor, 4, pa pomeni, da so med števili $1, 2, \ldots, 10$ natanko 4, ki niso del nobene pitagorejske trojke x, y, z, ki ustreza omejitvam $1 \leq x < y < z \leq 10$. To so števila 1, 2, 7 in 9.

Nadaljevanje opisa naloge. Pri opisu naloge sem izpustil še eno zelo pomembno podrobnost. Ko sem se naloge lotil prvič, je veljalo, da nobeno število na vhodu ne bo večje od 1000.

Kadar je število različnih možnih vhodnih podatkov majhno (pri naši nalogi 1000), se reševanja lahko lotimo tako, da vnaprej izračunamo vse možne odgovore, te vgradimo v program, dodamo še branje vhodnih podatkov ter pravilen izpis pripadajočih odgovorov in že imamo delujočo rešitev. Ker taka rešitev ne opravlja nobenega zamudnega računanja, saj vse odgovore pozna že vnaprej, je tudi zelo hitra.

Odgovorov seveda nisem iskal s svinčnikom in papirjem, ampak sem sestavil pomožni program, ki je to opravil namesto mene. Program je ob nekaj omejitvah pregledal vse dopustne trojice. Ker pa sem ga zagnal le enkrat, me ni skrbelo, ker ni bil ravno najhitrejši. Izračunane vrednosti sem v primerni obliki shranil na datoteko, dodal še nekaj vrstic kode in že sem imel rešitev (zapisano v jeziku C). Njen bistveni del je podan v nadaljevanju:

```
#include <stdio.h>

#define MAXN 1000

int stevci[MAXN][2] = {
  {0,1},{0,2},{0,3},{0,4},{1,2},
  {1,3},{1,4},{1,5},{1,6},{1,4},

      ⋮    (tu manjka 196 vrstic, narejenih s pomožnim programom)

  {157,202},{157,203},{157,204},{157,205},{157,203},
  {157,204},{158,204},{158,205},{158,205},{158,205}
};

int main(void)
{
  int n;

  while (scanf("%d", &n) == 1)
    printf("%d %d\n", stevci[n - 1][0], stevci[n - 1][1]);

  return 0;
}
```

Nekaj časa sem se lahko ponašal z eno od najhitrejših rešitev, a sčasoma se je na Sodniku nabralo ogromno podobno hitrih rešitev, zato so se sestavljavci nalog odločili, da nalogo nekoliko otežijo. Zgornjo mejo za velikost števil na vhodu so s 1000 dvignili na 10000.

Seveda se je bilo tudi spremenjene naloge moč lotiti na enak način, le da je imela dobljena rešitev namesto nekaj čez 200 več kot 2000 vrstic. In, zlomka, več kot 40000 znakov. Seveda sem takoj pomislil, da bi izračunane vrednosti lahko zapisal v bolj strnjeni obliki, z manj znaki. Do $n = 10000$ obstaja le 1593 primitivnih pitagorejskih trojk, v nobeni trojki pa ne sodeluje 1669 števil. V program bi tako lahko zakodiral le vrednosti, kjer se spremeni število trojk, na začetku rešitve pa bi iz teh podatkov hitro naračunal prave odgovore. A nisem imel prave volje za take dopolnitve. Raje sem vzel že napisani pomožni program, s katerim sem štel trojke, mu dodal branje podatkov ter prilagodil izpis. Tako sem dobil naslednjo rešitev:

```
#include <stdio.h>

#define MAXN 10000

extern long gcd(long, long); /* Poišče največji skupni */
                             /* delitelj števil. */

int main(void)
{
  int stevci[MAXN][2]; /* tabela odgovorov */
  int vtrojki[MAXN + 1] = {0}; /* Ali je število v kaki trojki? */
  int zunaj = 0; /* Koliko števil od 1 do n ni v nobeni trojki. */
  int trojke; /* Koliko primitivnih trojk z z <= n smo našli. */
  long x, y, z; /* long: kvadrati morda lahko presežejo INT_MAX. */
  int n;

  for (n = 1; n <= MAXN; n++) {
    /* Iščemo trojke, pri katerih je z = n. */
    zunaj++;
    for (x = 1, y = n - 1, z = n; x < y; x++) {
      while (x * x + y * y > z * z) y--;
      if (x * x + y * y == z * z) { /* nova trojka */
        /* Za primitivnost zadošča, da sta si tuja le x in y. */
        if (gcd(x, y) == 1) trojke++;
        if (!vtrojki[x]) { vtrojki[x] = 1; zunaj--; }
        if (!vtrojki[y]) { vtrojki[y] = 1; zunaj--; }
        if (!vtrojki[z]) { vtrojki[z] = 1; zunaj--; }
      }
    } /*for x,y,z*/
    stevci[n - 1][0] = trojke; /* Zapomnimo si število trojk */
    stevci[n - 1][1] = zunaj;  /* in število števil zunaj trojk. */
  } /*for n*/

  while (scanf("%d", &n) == 1)
    printf("%d %d\n", stevci[n - 1][0], stevci[n - 1][1]);

  return 0;
}
```

Ko sem se počasi le sprijaznil z ugotovitvijo, da moja rešitev ni več med najhitrejšimi, so na Sodniku ponovno popravili nalogo. Tokrat so zgornjo mejo za števila na vhodu povečali na 100000. Nič posebnega, sem si mislil. Popravim vrednost `MAXN` na 100000, spremenim tip `n`-ja v `long`, pa bo. A to je bil račun brez krčmarja. Rešitev je bila tako počasna,

da je prekoračila časovno omejitev, ki jo postavlja Sodnik. Tudi majhne izboljšave niso pomagale. Čas je bil za korenite posege.

Poznal sem formule, s katerimi opišemo primitivne pitagorejske trojke. V vsaki primitivni pitagorejski trojki sta števili x in y različne parnosti. Vse primitivne trojke, pri katerih je x sodo število, dobimo iz formul

$$x = 2ab, \quad y = a^2 - b^2 \quad \text{in} \quad z = a^2 + b^2,$$

kjer sta a in b, $a > b$, naravni števili različne parnosti, ki sta si tuji. Z zgornjimi formulami vsako primitivno pitagorejsko trojko dobimo na en sam način. Dokaz naštetih trditev ni težak, najdete ga npr. v [1, razdelek 10]. Gornje formule opisujejo primitivne trojke, pri katerih je x sodo (y pa potem liho) število. Za dano vrednost komponente z je takih trojk seveda natanko toliko kot tistih, pri katerih je $x < y$.

Na osnovi zgornjih formul sem sestavil naslednjo rešitev, ki je na Sodnikovem seznamu še vedno ena od najhitrejših:

```
#include <stdio.h>

#define MAXN 100000

extern long gcd(long, long);

int main(void)
{
  const int meja = 316; /* koren iz MAX je zgornja meja za a */
  long n, x, y, z, xx, yy, a, b;
  long stevci[MAXN][2] = {{0,0}};
  long prva[MAXN] = {0}; /* največje število v najmanjši trojki */

  /* Generiranje pitagorejskih trojk. */
  for (a = 2; a <= meja; a++)
    for (b = 1; b < a && (z = a * a + b * b) <= MAXN; b++)
      if (a % 2 != b % 2 && gcd(a, b) == 1) {
        /* Imamo novo primitivno pitagorejsko trojko. */
        x = 2 * a * b; y = a * a - b * b;
        stevci[z - 1][0]++; /* ena primitivna trojka več */
        /* Pregledamo večkratnike in beležimo "najmanjše" trojke. */
        for (xx = x, yy = y, n = z; n <= MAXN;
             xx += x, yy += y, n += z) {
          if (prva[xx-1] == 0 || prva[xx-1] > n) prva[xx-1] = n;
          if (prva[yy-1] == 0 || prva[yy-1] > n) prva[yy-1] = n;
          if (prva[n -1] == 0 || prva[n -1] > n) prva[n -1] = n;
        } /*for*/
      } /*if*/
```

```
  for (n=1; n < MAXN; n++) stevci[n][0] += stevci[n-1][0];  /*(1)*/
  for (n=0; n < MAXN; n++)                                  /*(2)*/
    if (prva[n] != 0) stevci[prva[n]-1][1]++;
  for (n=1; n < MAXN; n++) stevci[n][1] += stevci[n-1][1];  /*(3)*/
  for (n=0; n < MAXN; n++) stevci[n][1] = n+1-stevci[n][1]; /*(4)*/

  while (scanf("%ld", &n) == 1)
    printf("%ld %ld\n", stevci[n - 1][0], stevci[n - 1][1]);

  return 0;
}
```

Program zahteva kratko razlago. Z zankama `for` pregledamo vse primerne vrednosti za a in b ($1 \leq b < a \leq \sqrt{10^5}$, test parnosti in test tujosti) ter tako odkrijemo vse primitivne pitagorejske trojke do 10^5. V prvem delu tabele `stevci` štejemo, koliko je takih trojk z izbrano vrednostjo z-ja. Te kasneje v zanki (1) še seštejemo od začetka do tekočega elementa in tako ugotovimo, koliko je primitivnih trojk, ki imajo vse komponente manjše od izbrane vrednosti.

Za izračun drugih delov odgovorov pa je pomembna pomožna tabela `prva`. V njej za vsako število zabeležimo najmanjšo pitagorejsko trojko (glede na vrednost komponente z), ki to število vsebuje. Vrednost $k > 0$ v i-tem elementu tabele (ta ima indeks $i - 1$) tako pomeni, da smo našli pitagorejsko trojko, ki vsebuje število i, katere komponenta z je enaka k, trojke z manjšo vrednostjo komponente z pa nismo našli. Vrednost $k = 0$ po koncu obeh zank `for` pomeni, da število i ne nastopa v trojkah s komponentami do 10^5. V nadaljevanju iz vrednosti v tabeli `prva` izračunamo druge dele odgovorov. Najprej v zanki (2) za vsako število $n \leq 10^5$ ugotovimo, koliko je števil m, za katera je n največje število v najmanjši pitagorejski trojki, ki vsebuje m. Ta števila so se pri tem n-ju prvič znašla v kaki pitagorejski trojki. Nato v (3) podobno kot v (1) izračunamo vsote začetkov tabele, da ugotovimo, koliko števil do tekočega je v kaki pitagorejski trojki. V (4) nato v tabelo vpišemo "nasprotne" vrednosti, tiste, ki povedo, koliko števil do tekočega ni v nobeni pitagorejski trojki. Pri vseh operacijah z elementi tabel moramo paziti na indekse, saj ima v C-ju prvi element tabele indeks 0.

Nedavno so na Sodniku zgornjo mejo za števila na vhodu še enkrat dvignili, in sicer na 10^6. Rešitve skoraj ni bilo treba spremeniti, le vrednosti za konstanti `MAX` in `meja` je bilo treba popraviti na 10^6 oz. 1000.

Literatura

[1] J. Grasselli, Diofantske enačbe, Knjižnica Sigma 38, DMFAS, 1984

Martin Juvan