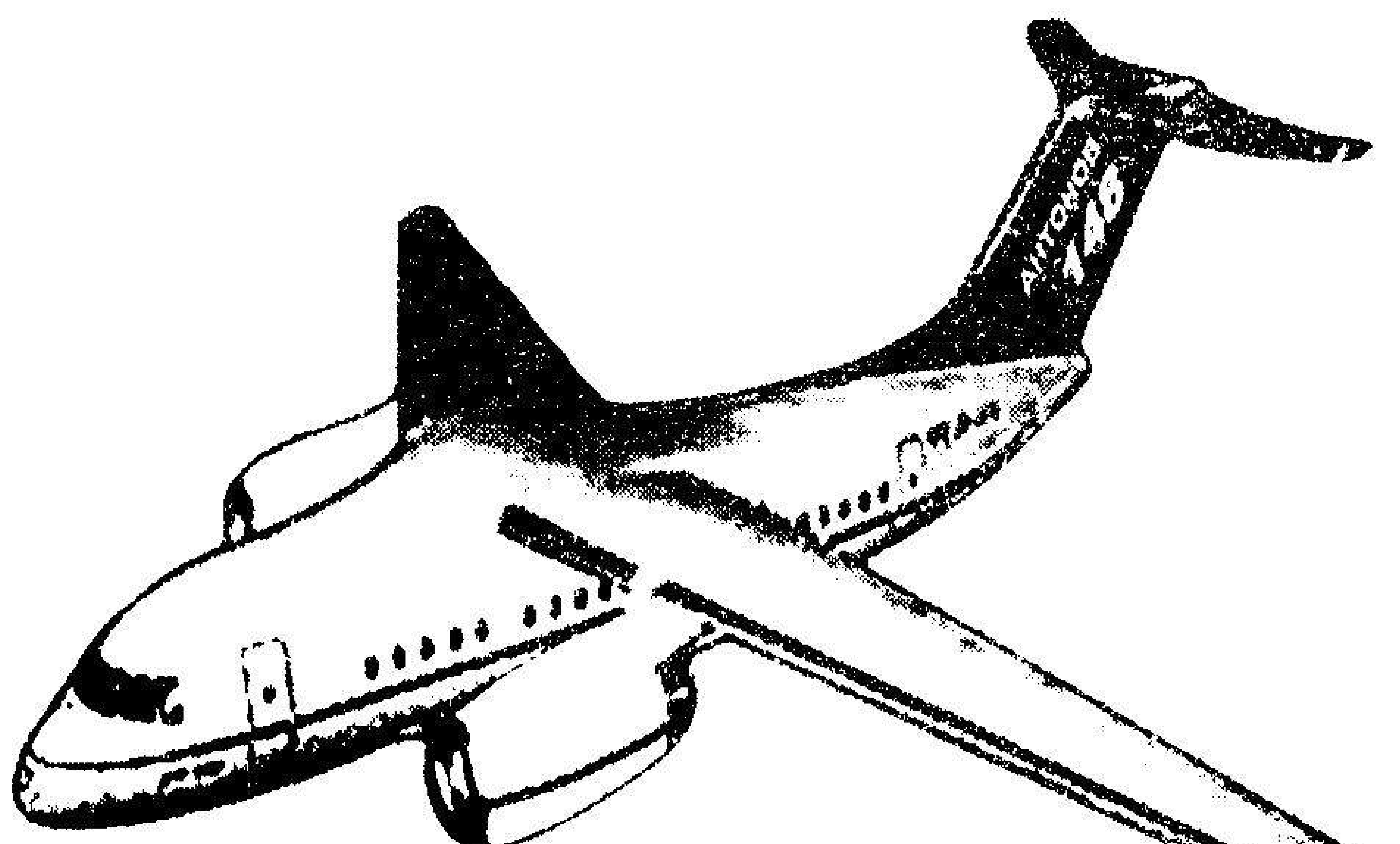
МЕТОДИКА РОЗВ'ЯЗАННЯ ЗАДАЧ З ІНФОРМАТИКИ НА ВСЕУКРАЇНСЬКІЙ СТУДЕНТСЬКІЙ ОЛІМПІАДІ



2007

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М.Є. Жуковського
"Харківський авіаційний інститут"

МЕТОДИКА РОЗВ'ЯЗАННЯ ЗАДАЧ З ІНФОРМАТИКИ НА ВСЕУКРАЇНСЬКІЙ СТУДЕНТСЬКІЙ ОЛІМПІАДІ

Практичний посібник

Харків «ХАІ» 2007

УДК 004.01(075.8)

Методика розв'язання задач з інформатики на Всеукраїнській студентській олімпіаді / І. Б. Сіроджа, О. Ю. Соколов, О. Г. Кириленко, Т. А. Клименко, О. С. Лістрова, П. О. Лучшев, В. Л. Петрик, Є. В. Соколова, Л. О. Філіпковська, Ю. К. Чернишов, О. В. Ярова. – Практичний посібник. – Харків: Нац. аерокосм. ун-т «Харк. авіац. ін-т», 2007. – 127 с.

Розглянуто розв'язання оригінальних задач з інформатики на Всеукраїнській студентській олімпіаді різних років і номінацій. Задачі орієнтовані за номінаціями: комп'ютерні науки, інженерні науки, економічні науки . Наведено приклади розв'язання задач, що обов'язково допоможуть підготуватися до участі в конкурсах та олімпіадах різного рівня, правильно оцінити свій рівень підготовки.

Для студентів і фахівців технічних, комп'ютерних та економічних спеціальностей.

Іл. 51. Табл. 9. Бібліогр.: 18 назв

Автори: І. Б. Сіроджа, О. Ю. Соколов, О. Г. Кириленко, Т. А. Клименко, О. С. Лістрова, П. О. Лучшев, В. Л. Петрик, Є. В. Соколова, Л. О. Філіпковська, Ю. К. Чернишов, О. В. Ярова

Рецензенти: д-р техн. наук, проф. Г. М. Жолткович,
канд. техн. наук, доц. В. Г. Іванов

2. НОМІНАЦІЯ "ІНЖЕНЕРНІ НАУКИ"

2.1. Загальний опис типів задач

Особливості підбору задач з номінації "Інженерні науки" полягають в тому, що мають бути враховані вимоги до практичної спрямованості, але в той же час необхідно підтримувати достатній з точки зору інформатики теоретичний рівень. Кожного року проведення олімпіади як у відбірковому, так і у вирішальному турі кількість задач становила п'ять – шість; таким чином, складність підтримувалася такою, щоб переможці мали змогу за час проведення туру розв'язати всі задачі. З іншого боку, задачі не повинні бути настільки простими, щоб всі задачі розв'язала більшість учасників. Лише за умови виконання цих вимог можна вважати, що змагання, якими і є олімпіада, проведені успішно. Практика проведення заключних турів Всеукраїнської студентської олімпіади з інформатики протягом 1998 – 2006 років показала, що інколи задачі були надто складними, щоб за відведений термін були б розв'язані всі; але загальний рівень для переможців становив чотири – п'ять розв'язаних задач.

В наступних підрозділах наведені приклади деяких задач з номінації, що розглядається. Вони далеко не вичерпують всіх тем, але є найбільш характерними. Алгоритмічна мова, якою реалізуються алгоритми, може бути будь-якою; в наведених прикладах застосовують той чи інший різновид мови Pascal. Ні в якому разі запропоновані приклади рішень не можуть вважатися оптимальними, але в умовах олімпіади дуже рідко учасникам вдається знайти дійсно оптимальне рішення. Взагалі кажучи, співвідношення простоти алгоритму і його оптимальності – дуже складне питання. При проведенні підсумкового розбору рішень серед розв'язаних задач перевага віддавалася тим, що легко читаються. Це, як правило, пов'язано зі структурованістю програми. Таким чином, в умовах олімпіади структурованість свідчить про ясне усвідомлення алгоритму.

Для розв'язання задач, які були запропоновані за цією номінацією, достатньо знати вищу математику і фізику в обсязі стандартної програми першого курсу вищого технічного навчального закладу. Однак бажано мати практику і теоретичну підготовку в галузі структур даних і загальної теорії алгоритмів [1, 2, 3].

2.2. Оптимальне розташування джерела сигналу

2004 рік

Умова

Є деяка кількість точок прийому сигналу, заданих координатами на площині. Рівень сигналу прийому зменшується з відстанню до джерела сигналу. Де слід розташовувати джерело сигналу, щоб рівень прийому для найбільш віддаленого (від джерела) приймача був максимальним із можливих ?

Відповідь складається з двох чисел: X і Y – координати джерела з десятьма вірними знаками після десяткової точки.

Координати приймача задані в текстовому файлі Z2_2t.txt.

Перш за все, в тому випадку, якщо кількість точок прийому сигналів є великою, треба залишити для розгляду тільки ті з них, які належать опуклій оболонці. Способів побудови такої оболонки існує багато. Один із них (далеко не оптимальний, але який порівняно просто реалізувати) полягає в послідовному відкиданні з масиву заданих точок тих із них, які знаходяться в трикутниках з вершинами, довільно вибраними з масиву.

Вважаємо, що заданий масив точок перетворено на масив точок, які належать опуклій оболонці. Те розташування джерела сигналу, яке треба знайти, характеризується тим, що найбільша відстань від джерела до точок масиву має найменше значення. Таким чином, одним із способів вирішення задачі є застосування того чи іншого методу двовимірної мінімізації цільової функції, що обчислюється як найбільша з відстаней від джерела сигналу до точок масиву.

Однак існує і точний метод розв'язання. Те розташування джерела сигналу, яке треба знайти, відповідає або середині відрізка, що поєднує дві найбільш віддалені точки з наявних, або центра кола найменшого радіуса, в якому або на межі якого знаходяться всі точки опуклої оболонки. Таке коло є описаним для деякого трикутника, вершини якого належать опуклій оболонці. Таким чином, треба переглянути точки опуклої оболонки по три, для кожної трійки знайти центр описаного кола і перевірити, чи не виходять за межі цього кола інші точки. Серед всіх знайдених кіл слід залишити те, радіус якого є найменшим. Його центр і є точкою, в якій слід розташувати джерело сигналу.

Приклад реалізації точного розв'язання дано мовою TP 5.5.

`Program Z2t_4a;`	Назва програми.
`uses crt;`	
`type tokr = record x,y,r : double; end;`	Тип кола.
`t_tochki = record x,y : double; end;`	Тип точки на площині.
`t_pary = record i1, i2 : integer; end;`	Тип номерів точок, які входять у пару.
`tmt = array[1..1800] of t_tochki;`	Тип масиву точок.

```
tmpar    = array[1..1800] of t_pary;
```

Тип масиву пар номерів.

```
tlin     = array[1..1800] of integer;
```

Допоміжний масив

```
Var
   n, k, nn    : integer;    i1, i2, i3  : integer;
   mt          : tmt;    xc,yc,rc,r  : double;
   xc0, yc0, rc0  : double;     delta       : double;
   namef      : string;    c          : char;
   pary       : tmpar;    npar       : integer;
   i          : integer;    n1, n2      : integer;
```

Глобальні змінні. xc, yc визначають центр описаного кола. xc0, yc0 – середина діаметра

```
procedure chtenie(namef : string; var mt : tmt;
var n : integer);
var f  : text;  i : integer; x,y      : double;
begin
assign(f,namef); reset(f); n := 0;
while not seekeof(f) do begin
inc(n); readln(f,mt[n].x,mt[n].y);
                end;    close(f);
end;
```

Процедура читання координат точок прийому сигналу і створення масиву точок

```
Procedure udalenije(var mt : tmt; var n : integer; n1,n2,n3 : integer);
Var  i, j, k, nn  : integer; x1,y1,x2,y2,x3,y3,
nx12,ny12,nx13,ny13,nx23,ny23  : double;
xc, yc, d12, d13, d23  : double;  mmt: tmt;
ind  : boolean;
begin
x1 := mt[n1].x; y1 := mt[n1].y;
x2 := mt[n2].x; y2 := mt[n2].y;
x3 := mt[n3].x; y3 := mt[n3].y;
nx12 := y2 - y1; ny12 := x1 - x2;
nx13 := y3 - y1; ny13 := x1 - x3;
nx23 := y3 - y2; ny23 := x2 - x3;
xc := (x1 + x2 + x3)/3; yc := (y1 + y2 + y3)/3;
d12 := (x1-xc) * nx12 + (y1-yc) * ny12;
        if d12 < 0 then begin
nx12 := -nx12; ny12 := -ny12; d12 := - d12;
                    end;
d13 := (x1-xc) * nx13 + (y1-yc) * ny13;
        if d13 < 0 then begin
nx13 := -nx13; ny13 := -ny13; d13 := - d13;
                    end;
d23 := (x2-xc) * nx23 + (y2-yc) * ny23;
        if d23 < 0 then begin
```

Процедура відкидання точок, які є внутрішніми у відношенні до трикутника, вершини якого мають номери n1, n2, n3. Для наявної точки визначають, чи знаходиться вона по один бік від сторін трикутника з його центром ваги. Для цього обчислюють відхилення центра й наявної точки від сторін, які задані як прямі, що проведені через дві точки (вершини)


```
nx23 := -nx23; ny23 := -ny23; d23 := - d23;
                    end;
nn := 0;
for i := 1 to n do if (i <> n1) and (i <> n2)
                 and (i <> n3) then begin
d12 := (x1-mt[i].x) * nx12 + (y1-mt[i].y) * ny12;
d13 := (x1-mt[i].x) * nx13 + (y1-mt[i].y) * ny13;
d23 := (x2-mt[i].x) * nx23 + (y2-mt[i].y) * ny23;
if (d12 < 0) or (d13 < 0) or (d23 < 0) then begin
inc(nn); mmt[nn].x := mt[i].x; mmt[nn].y :=
mt[i].y;
                        end;        end;
```

Переглядають всі точки масиву.
Номери точок, зовнішніх у відношенні до трикутника, записують у проміжний масив.
Обчислюють також їх кількість

```
mmt[nn+1] := mt[n1]; mmt[nn+2] := mt[n2];
mmt[nn+3] := mt[n3];
nn := nn+3; mt:= mmt; n := nn;
end;
```

Новий масив приймають таким, що дорівнює отриманому проміжному

```
procedure razrejenie(var mt : tmt; var n : integer; kol_wo : integer);
var i,j, n1,n2, n3 : integer;
begin
for i := 1 to 100  do j := random(100);
for i := 1 to kol_wo do begin
n1 := 1 + random(n); n2 := 1 + random(n);
n3 := 1 + random(n);
if (n1<>n2) and (n1<>n3) and (n2<>n3) then
udalenije(mt,n,n1,n2,n3);
end;
end;
```

Процедура розрідження початкового масиву.
Задану кількість разів (kol_wo) повторюється випадковий вибір номерів трьох точок з наявного масиву та відкидання внутрішніх

```
procedure naibolee_udalennye(mt : tmt;
n : integer; var n1, n2 : integer);
var i, j : integer; d, dmax : double;
begin
dmax := -1;
for i := 1 to n-1 do for j := i+1 to n do begin
d    :=    sqrt(sqr(mt[i].x-mt[j].x)+sqr(mt[i].y-
mt[j].y));
                        if d > dmax then begin
dmax := d; n1 := i; n2 := j;
                                        end;
                    end;
end;
```

Процедура пошуку номерів двох найбільш віддалених точок у масиві

```
function wne_okr(mt : tmt; n : integer; xc, yc,
rc, delta : double): word;
var
   i, kwne     : integer;
```

Підрахування кількості точок, які знаходяться

```
begin
kwne := 0;
for i := 1 to n do
if sqrt(sqr(mt[i].x-xc)+sqr(mt[i].y-yc)) >=
 rc + delta      then inc(kwne);
wne_okr := kwne;
end;
```

зовні від кола з центром у точці (xc, yc), яке має радіус rc. Перевірку роблять із заданою похибкою delta

```
procedure centr(mt: tmt; n : integer;
i1,i2,i3 : integer;  var xc, yc, rc  : double);
var a11, a12, a21, a22, b1, b2, d, dx, dy :
double;
begin
a11 := mt[i2].x-mt[i1].x; a12 := mt[i2].y-
mt[i1].y;
a21 := mt[i3].x-mt[i1].x; a22 := mt[i3].y-
mt[i1].y;
b1 := 0.5*(sqr(mt[i2].x)-sqr(mt[i1].x) +
sqr(mt[i2].y)-sqr(mt[i1].y));
b2 := 0.5*(sqr(mt[i3].x)-sqr(mt[i1].x) +
sqr(mt[i3].y)-sqr(mt[i1].y));

d := a11*a22 - a12*a21;
dx := b1*a22-b2*a12; dy := a11*b2-a21*b1;
xc := dx/d; yc := dy/d;
rc := sqrt(sqr(mt[i1].x-xc)+sqr(mt[i1].y-yc));
end;
```

Процедура для знаходження центра кола, описаного відносно трикутника, вершинами якого є точки з номерами i1, i2, i3.
Центр описаного кола знаходиться на перетині серединних перпендикулярів до двох сторін трикутника.
Одночасно обчислюють радіус описаного кола

```
procedure wybor(mt : tmt; n : integer; var
i1,i2,i3 : integer;
var xc, yc, rc : double);
var i, j, k, kwne : integer; rm : double;
begin
rm := 1e30;
for i := 1 to n-2 do
   for j := i+1 to n-1 do
      for k := j+1 to n do
                    begin
centr(mt,n,i,j,k,xc,yc,rc);
kwne := wne_okr(mt,n,xc,yc,rc,delta);
if (kwne = 0) and (rc < rm) then begin
i1 := i; i2 := j; i3 := k; rm := rc;
                         end;
                    end;
```

В даній процедурі перевіряють трикутники шляхом перегляду по три точки з даного масиву, чи є точки, які знаходяться зовні від описаного кола відповідного трикутника. Якщо кількість таких точок дорівнює нулю, то для цього кола обчислюють координати центра та радіус.
Результатом є центр кола з найменшим радіусом і величина


```
centr(mt,n,i1,i2,i3,xc,yc,rc);
end;

Procedure wypucl_ob(mt : tmt; n : integer;
              var pary : tmpar; var npar : inte-
ger);
label metka;
var i, j, k : integer;  x, y, x1, x2, y1, y2  : dou-
ble;
xc, yc : double;  nx, ny, c  : double;
d, dc, ind              : boolean;
begin
xc := 0; yc := 0;
for i := 1 to n do begin
xc := xc + mt[i].x; yc := yc + mt[i].y;
                  end;
xc := xc/n; yc := yc/n;
npar := 0;
for i := 1 to n-1 do begin
  x1 := mt[i].x; y1 := mt[i].y;
     for j := i+1 to n do begin
  x2 := mt[j].x; y2 := mt[j].y;
  nx := y2-y1; ny := x2-x1;
  c := x1*nx - y1*ny;
  dc := xc*nx - yc*ny - c > 0;
ind := true;
for k := 1 to n do if (k<>i) and (k<>j) then be-
gin
x := mt[k].x; y := mt[k].y;
d := x*nx - y*ny - c > 0;
ind := (d and dc) or ((not d) and (not dc));
if not ind then goto metka;
                                              end;

       metka : if ind then begin
inc(npar); pary[npar].i1 := i; pary[npar].i2 := j;
                                     end;
                          end;
                   end;
end;
```

цього радіуса.

Процедура пошуку масиву номерів точок, які утворюють опуклу оболонку. Для цього здійснюють перегляд точок по дві; для прямої, проведеної через ці точки, проводять пряму, після чого перевіряють, чи по один бік від неї знаходяться інші точки. Якщо це не так, то пара точок,що розглядається, не входить в опуклу оболонку. В противному разі в масив **pary** додають номери точок, які входять у знайдену пару.

```
procedure wybrasywanie_serediny(var mt :
tmt; var n : integer; pary : tmpar; npar : inte-
ger);
var i, j          : integer; mtm          : tmt;
m : integer; lin : array[1..1300] of byte;
begin
```

Процедура виділення масиву номерів точок, що входять хоча б раз в якийсь елемент масиву pary.
Інші точки початкового

```
for i := 1 to n do lin[i] := 0;
for i := 1 to npar do begin
j := pary[i].i1; lin[j] := 1;
j := pary[i].i2; lin[j] := 1;
                    end;
nn := 0;
for i := 1 to n do
if lin[i] = 1 then begin
 inc(nn); mtm[nn].x := mt[i].x; mtm[nn].y :=
mt[i].y;
                end;
mt := mtm; n := nn;
end;
```

масиву для розв'язання задачі є зайвими, і тому провадиться зтиск початкового масиву до такого, який містить тільки точки опуклої лінійної оболонки

```
BEGIN
clrscr; delta := 1e-12;
chtenie('z_2t.4b',mt,n);
razrejenie(mt, n, 150);
wypucl_ob(mt,n,pary,npar);
wybrasywanie_serediny(mt,n,pary,npar);
```

Основна програма.
Після читання початкових даних провадиться розрідження шляхом вибору довільних трикутників (150 разів), побудова опуклої лінійної оболонки і зтиск початкового масиву

```
wybor(mt,n,i1,i2,i3,xc,yc,rc);
```

Побудова трикутника, описаного відносно опуклої лінійної оболонки в цілому

```
naibolee_udalennye(mt, n, n1, n2);
```

Знаходження номерів двох найбільш віддалених одна від одної точок

```
xc0 := (mt[n1].x+mt[n2].x)*0.5;
yc0 := (mt[n1].y+mt[n2].y)*0.5;
rc0 := sqrt(sqr(mt[n1].x-mt[n2].x)+
                  sqr(mt[n1].y-mt[n2].y))*0.5;
```

Знаходження центра (xc0, yc0) відрізку, який поєднує дві найбільш віддалені точки, і половини його довжини rc0

```
nn := wne_okr(mt,n,xc0,yc0,rc0,delta);
```

Знаходження кількості (nn) точок, які лежать зовні від кола з центром в точці (xc0, yc0) і радіусом rc0

```
writeln(' xc, yc, radius :');
writeln(' nn = ', nn,' ',
    xc0:12:9,' ',yc0:12:9,' ',rc0:12:9);
```

Якщо nn дорівнює нулю, то потрібне розташування джерела сиг-


```
writeln(xc:12:9,' ',yc:12:9,' ',rc:12:9);
readln;
END
```

налу задається точкою (xc0, yc0), у противному разі – точкою (xc, yc).

2.3. Залишок від степеня

2004 рік

Умова

Тур перший

Знайти залишок від ділення числа a^b на число d, причому a, b і d знаходиться в межах від 100 до 1500.

Тест : a = 234, b = 1243, d = =1427.

Розв'язання основується на формулі, яку легко вивести:

$$(a \cdot b) \bmod d = ((a \bmod d) \cdot b) \bmod d \ . \tag{2.1}$$

Наприклад, залишок від ділення степеня a^c , де c=5, на число d можна обчислити так:

$(a \cdot a \cdot a \cdot a \cdot a) \bmod d = (((1 \cdot a) \bmod d) \cdot a \cdot a \cdot a \cdot a) \bmod d =$
$=(a_1 \cdot a \cdot a \cdot a \cdot a) \bmod d = (((a_1 \cdot a) \bmod d) \cdot a \cdot a \cdot a) \bmod d =$
$=(a_2 \cdot a \cdot a \cdot a) \bmod d =(((a_2 \cdot a) \bmod d) \cdot a \cdot a) \bmod d =$
$=(a_3 \cdot a \cdot a) \bmod d =(((a_3 \cdot a) \bmod d) \cdot a) \bmod d =$
$=(a_4 \cdot a) \bmod d = a_5$.

Тут

$a_1=(1 \cdot a) \bmod d$, $a_2=(a_1 \cdot a) \bmod d$, $a_3=(a_2 \cdot a) \bmod d$,
$a_4=(a_3 \cdot a) \bmod d$, $a_5=(a_4 \cdot a) \bmod d$.

```
function mod_step(a,c,d : longint) : longint;
var i, b : longint;
begin
b := a; a := 1;
for i := 1 to c do a := (a*b) mod d;
mod_step := a;
end;
```

Програмна реалізація обчислення залишку від ділення степеня на задане число

Тур другий

Знайти залишок від ділення числа a^b , де b $=c^d$, на число e, причому a, c, d і e знаходяться в межах від 100 до 1500.

Для тестового розрахунку прийняти a = 632, b = 924, c = 1389, d = 912, e = 1426.

Розв'язання. З формули (2.1) випливає, що залишок від діле
добутку на деяке число дорівнює залишку від ділення добутку зал
ків:

$$(a \cdot b) \bmod d = ((a \bmod d) \cdot b) \bmod d = ((a \bmod d) \cdot (b \bmod d)) \bmod d\ . \quad (2.$$

Звідси випливає таке співвідношення:

$$(a^n) \bmod d = ((a \bmod d)^n) \bmod d\ . \quad (2.$$

На додаток до функції mod_step можна використати функцію resolution(a,c,d,e : longint) : longint, яка реалізує повторення операц обчислення залишків. Наприклад,

$$\left(a^{c^5}\right) \bmod e = \left(a^{c \cdot c \cdot c \cdot c \cdot c}\right) \bmod e = \left(\left(a^c\right)^{c \cdot c \cdot c \cdot c}\right) \bmod e =$$

$$= \left(\left(\left(a^c\right) \bmod e\right)^{c \cdot c \cdot c \cdot c}\right) \bmod e = \left(a_1^{c \cdot c \cdot c \cdot c}\right) \bmod e = \ldots$$

function resolution(a,c,d,e : longint) :
longint;
var i : longint;
begin
for i := 1 to d do a := mod_step(a,c,e);
resolution := a;
end;
	Функція, яка реалізує кратне повторення обчи лення залишку від ділен основи степеня a^c на чи ло e.

2.4. Об'єднання прямокутників

2006 рік

Умова

N прямокутників зі сторонами, паралельними осям коордик задані координатами лівого верхнього та правого нижнього кутів :

(11; 56.56), (35.35; 13)	(28; 67.67), (74.74; 18)	(43; 81.81), (92.9 46)
(15; 127.12), (51.51; 48)	(52; 97.97), (76.76; 26)	(7; 77.77), (39.3 33)
(25; 53.53), (55.55; 4)	(31; 30.3), (75.75; 2)	(27; 63.63), (33.3 43)

Знайти площу їх об'єднання.

Один із методів розв'язання подібних задач полягає в нанесе прямокутної сітки спеціального вигляду в прямокутній області, яка о ймає всі задані прямокутники. Для цього будується масив 2•N сел, кожне з яких збігається зі значенням абсциси або лівого, або п вого з ребер того чи іншого із заданих прямокутників; після цього м сив впорядковується за зростанням. Аналогічно будується впоряд ваний масив ординат ребер заданих прямокутників. Ці масиви і виз чають сітку; її можна уявити у вигляді набору $(2•N)^2$ прямокутник Кожний з них або повністю належить деякій сукупності початков прямокутників, або не належить жодному. Для знаходження пло об'єднання заданих прямокутників залишається переглянути всі пр

кутники, які належать сітці, і для якоїсь точки кожного з них (наприклад, точки перетину діагоналей) визначити, чи належить ця точка хоча б одному із початкових прямокутників. Це дає можливість виділити прямокутники сітки, які належать об'єднанню початкових; якщо знати їх площі, стає можливим знайти площу об'єднання.

Розглянемо фрагменти однієї з можливих реалізацій цього алгоритму на мові Delphi (6 або 7).

Введемо набір необхідних типів:

type t_pr = record xl, yw, xr, yn : double; end;	Тип, який описує окремо взятий заданий прямокутник координатами лівої, верхньої, правої, нижньої вершин
t_m_pr = array[1..100] of t_pr;	Тип масиву заданих прямокутників. Оскільки кількість прямокутників є невеликою, масив для спрощення задається із залишком
t_mass_1 = array[0..1000] of double;	Тип масивів, які будуть задавати сітку

Глобальні змінні, які використовують у даній задачі:

var n : byte;	Кількість прямокутників.
p : t_m_pr;	Масив заданих прямокутників.
xmin, xmax, ymin, ymax : double;	Параметри, які задають область, в якій знаходяться всі прямокутники.
s : double;	Площа.
mass_x, mass_y : t_mass_1;	Масиви абсцис і ординат вузлів сітки.

Необхідні такі найпростіші процедури:

function yes_in(x, y : real; nomer : byte) : boolean;
begin
with p[nomer] do
yes_in := (x > xl) and (x < xr) and (y > yn) and (y < yw);
end;
	Функція, яка набуває значення істини в тому випадку, якщо точка із заданими координатами знаходиться в прямокутнику із заданим номером

procedure porjadok(var mass : t_mass_1; m : integer); var i, j : integer; c : double;	Найпростіший алгоритм впорядкування масиву за збільшенням.

```
begin
for i := 1 to m-1 do for j := i+1 to m
do
  if mass[i] > mass[j] then begin
c := mass[i]; mass[i] := mass[j];
mass[j] := c;
                    end;
end;

procedure reshenije;
var ind : boolean; i, j, k : integer;
x, y : real;
stroka : string;
begin
s := 0;
for i := 1 to 2*n-1 do begin

x := (mass_x[i] + mass_x[i+1])*0.5;

                 for j := 1 to 2*n-1 do
begin
y := (mass_y[j] + mass_y[j+1])*0.5;

ind := false;
for k := 1 to n do ind := ind or
yes_in(x, y, k);


if ind then s := s + (mass_x[i+1]-
mass_x[i])*(mass_y[j+1]-mass_y[j]);
                              end;
                 end;
str(s : 15:4,stroka);
Form1.Label5.caption := stroka;
end;
```

Якщо задана кількість прямок ників є великою, то необхід застосовувати більш ефектив але одночасно і більш склад алгоритми

Суто розв'язання задачі

Масиви абсцис і ординат вузлі допоміжної сітки вважають вже впорядкованими

Переглядають всі інтервали сіт ки за віссю ОХ; для кожного з них обчислюють середину

Переглядають всі інтервали сіт ки за віссю ОУ; для кожного з них обчислюють середину

Знайдену середину наявної комірки допоміжної сітки перевіряють на належність якому-небудь із заданих прямокутник Якщо знайдена точка належи об'єднанню, обчислюють площ комірки і додають до загальн площі об'єднання.

Виведення результату на форм

Процес розв'язання задачі полягає в такому:

```
procedure TForm1.Button1Click(Sender: Tobject);
var f : textfile; i : byte; name : string;

begin
name := 'Prjamoug.txt';
assignfile(f,name);

                  reset(f);
```



```
read(f,n);
for i := 1 to n do
read(f,p[i].xl, p[i].yw, p[i].xr, p[i].yn);
                 closefile(f);

xmin := 1000; xmax := -xmin;
ymin := 1000; ymax := -ymin;

for i := 2 to n do
          with p[i] do begin
if xl < xmin then xmin := xl;
if yn < ymin then ymin := yn;

if xr > xmax then xmax := xr;
if yw > ymax then ymax := yw;
                     end;

for i := 1 to n do with p[i] do begin
mass_x[2*i-1] := xl; mass_x[2*i] := xr;
mass_y[2*i-1] := yn; mass_y[2*i] := yw;
                          end;
porjadok(mass_x, 2*n); porjadok(mass_y, 2*n);

reshenije;
end;
```

Зчитування параметрів заданих прямокутників

Введення параметрів робочої області

Побудова масиву абсцис і ординат вузлів допоміжної сітки

Впорядкування вузлів

Суто розв'язання.

2.5. Логістичне рівняння

2004 рік

Умова

Розглядають послідовність : $x_{n+1} = 0.0038 \cdot x_n \cdot (1000.0 - x_n)$, де x_1 = 999.999. З неї утворюють масив { $x_1, x_2, .., x_n$ }. Знайти такі номери і m елементів цього масиву, для яких абсолютне значення різниці $x_i - x_m$ було б найменшим. Прийняти n = 100000.

Розв'язання. Послідовність, про яку йде мова в задачі, породжена логістичним рівнянням. При вибраному значенні множника 0.0038 вона є хаотичною, тобто не має ані періоду, ані квазіперіоду [4]. Після утворення масиву його слід впорядкувати за збільшенням, після чого задача стає простою. Впорядкування можна провадити різними способами. Особливість даної задачі полягає у великій кількості елементів масиву. Це означає, що треба використати який-небудь швидкий метод сортування. Нижче наведено варіант так званого «швидкого сортування» Хоара [1 – 3]. Програму написано у вигляді консольної програми в середовищі Delphi7.

```
program Logist;
{$APPTYPE CONSOLE}
uses  SysUtils;
```

Автоматичне завдання вид[...] і ресурсів програми

```
Type
  el_index  = record
   i      : longint;
   x      : double;
          end;
```

Тип окремого елемента ма[...]сиву. Містить у собі номе[...] відповідного елемента по[...]слідовності і його значення

```
  t_index = array of el_index;
```

Тип масиву в цілому. Маси[...] задається як динамічний

```
Procedure part(var a : t_index;
 left, right : longint; var c : longint);
var aa : double;
        x : el_index; i,j : longint;
begin
aa := a[left].x; i := left; j := right;
while i < j do begin
  while (i < j)  and (a[j].x >= aa) do dec(j);
  while (i < j)  and (a[i].x <  aa) do inc(i);
       if i < j then begin
  x := a[i]; a[i] := a[j]; a[j] := x;
                 end;
           end;
  c := j;
end;
```

Складова частина алгорит[...]му двійкового сортування [...] розподілення частини маси[...]ву на інтервалі номерів ві[...] left до right на два підмаси[...]ви; у лівому з них збирают[...] всі елементи, менші, ні[...] критичне число aa, а в пра[...]вому – не менші. Результа[...]том є також номер розділь[...]ника с
Як критичне число в даном[...] варіанті вибрано перши[...] елемент (тобто лівий) час[...]тини масиву, що розгляда[...]ється

```
Procedure quick_s(var a : t_index;
                  left, right : longint);
var c : longint;
begin
if left < right then begin
```

Алгоритм швидкого сорту[...]вання

```
part(a,left, right, c);
```

Розподілення по фракціях

```
quick_s(a,left,c);
quick_s(a,c+1,right);
              end;end;
```

Рекурсивний виклик проце[...]дури сортування для впоря[...]дкування окремо лівої і пра[...]вої фракцій

```
Begin
n := 100000; x := 999.999; c := 0.0038;
setlength(a,n+1);
```

Початок основної програми[...] Умови задачі та виділенн[...] пам'яті для динамічного ма[...]сиву


```
[...]r i := 1 to n do begin
[...]a[i].i := i; a[i].x := x;
[...] := c*x*(1000.0-x);
[...]nd;
[...]riteln;
```

Оформлення послідовності, отриманої з логістичного рівняння, у вигляді масиву

```
[...]uick_s(a,1,n);
```

Впорядкування одержаного масиву

```
[...] min := a[n].x - a[1].x;
[...] := a[1].i; m := a[n].i;
[...]r i := 1 to n-1 do begin
[...] := a[i+1].x - a[i].x;
[...]f x < d_min then begin
[...] min := x; k := a[i].i; m := a[i+1].i;
              end;
                end;
```

Вирішення задачі пошуку інтервалу найменшої довжини

```
[...]riteln(k:8,' ',m:8,' ',d_min);
```

Одержання відповіді

```
[...]eadln;

{ TODO -oUser -cConsole Main : Insert
[...]ode here }
[...]nd.
```

2.6. Кільця та пружини

2004 рік

Умова

На кожному боці прямокутного трикутника з катетами довжини [...] розташоване невагоме кільце. Кожне з них може вільно пересувати[...]ся по сторонах. Кільця сполучені пружинами, сили пружності яких [...]дорівнюють $k_i \cdot d_i$, де d_i – відстань між кільцями. Знайти площу трикут[...]ника, утвореного пружинами, в стані рівноваги. Для тестового розра[...]хунку $k_1 = 3$, $k_2 = 4$, $k_3 = 5$; k_1 – для пружини, яка сполучає кільця на ка[...]тетах.

Задачу можна звести до мінімізації сумарної потенціальної енер[...]гії, накопиченої в пружинах. Методів мінімізації існує дуже багато; найпростіший з них – метод покоординатного спуску в багатовимір[...]ному випадку і методи типу методу золотого поділу в одновимірному.

Розглянемо трикутник, про який йдеться в задачі, так, що катети виходять з початку координат по осях OX і OY. Перше з кілець нехай розташовується на гіпотенузі і має координати (x1, y1): x1 + y1 = 1.

Друге кільце має координати (0, y2), а третє – (x3, 0). Тоді відстань від першого кільця до другого обчислюється так: $\sqrt{(x1)^2+(y1-y2)}$ d2 і d3 – як відстані від першого кільця до третього і від другого третього – за такими формулами: $d2=\sqrt{(x1-x3)^2+(y1)^2}$, $d3 =\sqrt{(y2)^2+(x3)^2}$. Таким чином, набір відстаней між кільцями параме ризується шляхом введення параметрів: x ≡ x3, y ≡ y2, z ≡ x1. Відмі мо також, що y1 = 1 – z.

Згідно із законом Гука, сила пружності пропорційна подовжен пружини від стану спокою. В даному випадку подовження будемо вв жати таким, що дорівнює довжині пружини. Потенціальну енергію, я накопичено в розтягнутій пружині, обчислюють як одну другу квадра сили пружності, помноженої на коефіцієнт пружності. Повна потенці льна енергія дорівнює сумі енергій, накопичених у трьох пружинах.

Приклад програми з наведеною параметризацією подається вигляді консольної програми в середовищі Delphi 7.

```
Program Prujiny;
{$APPTYPE CONSOLE}
uses SysUtils;
```

Назва програми і модуль,що використовується

```
type t_f = function(x,y,z,d0,k1,k2,k3 : double): double;
var x, y, z, d0, k1, k2, k3, epsilon : double;
```

Тип функції для обчислення довжин пружин і глобальні змінні

```
Function fd1(d0,x,y,z : double): double;
var d : double;
begin
d := sqrt(sqr(z) + sqr(1-z-y)); fd1 := sqr(d - d0);
end;
```

Функція, з допомогою якої обчислюють довжину першої пружини

```
function fd2(d0,x,y,z : double):double;
var d : double;
begin
d := sqrt(sqr(x-z) + sqr(1-z)); fd2 := sqr(d - d0);
end;
```

Функція, з допомогою якої обчислюють довжину другої

```
function fd3(d0,x,y,z : double):double;
var d : double;
begin
d := sqrt(sqr(x) + sqr(y)); fd3 := sqr(d - d0);
end;
```

Функція, з допомогою якої обчислюють загальну енергію пружин

```
function energija(d0,x,y,z,k1,k2,k3: double):
double;
begin
ergija := 0.5*(k1*fd1(d0,x,y,z)+
  k2*fd2(d0,x,y,z)+k3*fd3(d0,x,y,z));
d;
```



```
ction min_x(f : t_f; d0,k1,k2,k3,x,y,z, epsi-
n : double): double;
 f0, f1, x0, x1, h : double;
gin
= 0.01;
 = x; f0 := f(d0,x0,y,z,k1,k2,k3);
  repeat
 := x0 + h; f1 := f(d0,x1,y,z,k1,k2,k3);
 >= f0 then h := -h*0.6;
 := x1; f0 := f1;
  until abs(h) < epsilon;
n x := x0;
  end;

ction min_y(f : t_f; d0,k1,k2,k3,x,y,z, epsi-
n : double): double;
 f0, f1, y0, y1, h : double;
gin
= 0.01;
 := y; f0 := f(d0,x,y0,z,k1,k2,k3);
  repeat
 := y0 + h; f1 := f(d0,x,y1,z,k1,k2,k3);
 >= f0 then h := -h*0.6;
 := y1; f0 := f1;
  until abs(h) < epsilon;
n y := y0;
  end;

ction min_z(f : t_f; d0,k1,k2,k3,x,y,z, epsi-
n : double): double;
 f0, f1, z0, z1, h : double;
gin
= 0.01;
 := z; f0 := f(d0,x,y,z0,k1,k2,k3);
  repeat
 := z0 + h; f1 := f(d0,x,y,z1,k1,k2,k3);
 >= f0 then h := -h*0.6;
 := z1; f0 := f1;
  until abs(h) < epsilon;
n z := z0;
  end;
```

Функції, з допомогою яких реалізують пошук мінімуму загальної енергії за трьома параметрами окремо. Як один із вхідних параметрів прийнято похибку. Використовують метод класу методів золотого поділу. Якщо крок за параметром призводить до зменшення цільової функції, то довжина і напрямок кроку не змінюється. У противному разі довжину кроку домножують на 0.6, а знак змінюють на протилежний

```
procedure step_optim(f : t_f; d0,k1,k2,k3,
epsilon : double;    var x, y, z : double);
begin
        x := min_x(f,d0,k1,k2,k3,x,y,z,epsilon);
        y := min_y(f,d0,k1,k2,k3,x,y,z,epsilon);
        z := min_z(f,d0,k1,k2,k3,x,y,z,epsilon);
end;
```

Процедура послідовної оптимізації за параметрами x, y, z

```
procedure optim(f : t_f; d0,k1,k2,k3,epsilon :
double; var x, y, z : double);
var d, x0, y0, z0 : double;
begin
        repeat
x0 := x; y0 := y; z0 := z;
step_optim(f, d0,k1,k2,k3,epsilon,x, y, z);
d := abs(x-x0)+abs(y-y0)+abs(z-z0);
        until d < 3*epsilon;
end;
```

Після мінімізації по черзі за параметрами x, y, z у тому випадку, якщо нова точка в просторі (x, y, z) віддалена від попередньої більш ніж на 3•epsilon, то знову повторюється мінімізація по всіх трьох параметрах по черзі. У противному разі вважають, що точку мінімуму знайдено

```
Function plosh(x,y,z : double):double;
Begin
        plosh := 0.5*(y*(z-x) + x*(1-z));
end;
```

Функція, з допомогою якої обчислюють площу трикутника, вершини якого збігаються з кінцями

```
BEGIN
k1 := 3; k2 := 4; k3 := 5; epsilon := 1e-10;
d0 := 0.0;
        x := 0.5; y := 0.5; z := 0.5;
optim(energija,d0,k1,k2,k3,epsilon, x, y, z);
writeln(' x=',x:16:9,' y=',y:16:9,' z=',z:16:9);
writeln(' S=',plosh(x,y,z):16:9);
        readln;
END.
```

Основна програма.

Завдання умов задачі, похибки і стартові значення параметрів мінімізації

Виведення значень знайдених величин

2.7. Задача про три сфери

2004 рік

Умова

В нерухомій сфері радіуса 20 м з центром у початку системи координат вільно переміщуються дві сфери радіусів 0.5 і 0.8 м. У момент зіткнення з більшою сферою виникає пружне відштовхування. Поч...

...ні положення задаються відповідно координатами (5, 0, 6 м) і (17, 0, ... м), а початкові швидкості векторами (2, 10, 0 м/с), (5, 5, 0 м/с). Через скільки секунд зіштовхнуться малі сфери? Відповідь дати з точністю до сотої секунди.

Спосіб розв'язання 1. Полягає в спостеріганні за послідовністю переміщень рухомих сфер із заданим малим кроком за часом Δt з даними швидкостями. Можливі три види подій: подія 1 – внутрішнє зіткнення першої рухомої сфери із зовнішньою (нерухомою) сферою, подія 2 – внутрішнє зіткнення другої рухомої сфери із зовнішньою і подія 0 – зовнішнє зіткнення рухомих сфер. Події 1 або 2 означають те, що відстань між центром рухомої сфери і центром зовнішньої сфери стає більшим модуля різниці радіусів сфер, які беруть участь у події. Подія 0 настає в тому випадку, якщо відстань між центрами рухомих сфер стає меншою суми їх радіусів. Якщо відбувається подія 0, то розрахунки припиняються і на екран виводиться наявний час як відповідь. Якщо відбувається подія 1, то швидкість першої сфери змінюється за законом пружного відбиття від нерухомої сфери. Це означає, що знак тієї компоненти швидкості, яка є паралельною радіусу, який проведено з центра нерухомої сфери в центр першої сфери, змінюється на протилежний. Аналогічні перетворення відбуваються і під час події 2.

Швидкість розрахунків при використанні наведеного алгоритму залежить від величини кроку за часом. Чим він менший, тим обчислення точніші й, головне, достовірніші. Але зменшення Δt призводить до збільшення часу розрахунків.

Рішення, отримане за цим способом, може бути прийнятим в умовах олімпіади (тобто браку часу на обмірковування алгоритму), якщо за розумний час одержана достатня точність, що дуже мало – ймовірно.

Спосіб 2. Відмітимо, що більшу частину часу сфери переміщуються вільно без зміни швидкості. Швидкості змінюються тільки в момент, коли відбувається зіткнення. Розглянемо першу з рухомих сфер. Нехай в деякий момент часу dt=0 положення її центра задано вектором $\vec{r}_{10}$. Зміна положення центра при вільному пересуванні відбувається по прямій лінії з напрямним вектором $\vec{v}_1$. Параметричне рівняння руху центру цієї сфери має такий вигляд:

$$\vec{r}_1(dt) = \vec{r}_{10} + \vec{v}_1 \cdot dt. \qquad (2.4)$$

Подія 1 відбувається тоді, коли $\vec{r}_1^{\,2} = (R_0 - R_1)^2$. Для знаходження моменту часу dt_1 виникнення події 1 треба розв'язати відносно величини dt таке квадратне рівняння:

$$\vec{v}_1^{\,2} \cdot (dt)^2 + 2 \cdot (\vec{r}_{10}, \vec{v}_1) \cdot dt + (\vec{r}_{10}^{\,2} - (R_0 - R_1)^2) = 0. \qquad (2.5)$$

Серед двох коренів події 1 відповідає менший з невід'ємних. У момент внутрішнього зіткнення після перетворення швидкостей скалярний добуток $(\vec{r}_{10},\vec{v}_1)$ є від'ємним, і тому треба використати та формулу:

$$dt_1=\frac{-(\vec{r}_{10},\vec{v}_1)+\sqrt{(\vec{r}_{10},\vec{v}_1)^2-\vec{v}_1^2\cdot(\vec{r}_{10}^2-(R_0-R_1)^2)}}{\vec{v}_1^2}. \tag{2.6}$$

Якщо вважати, що обчислення моменту часу dt відповідає тіль моменту внутрішнього відбиття, коли $|\vec{r}_{10}|=R_0-R_1$, то формула (2 спрощується так:

$$dt_1=-2\cdot\frac{(\vec{r}_{10},\vec{v}_1)}{\vec{v}_1^2}. \tag{2.7}$$

Аналогічно обчислюють момент dt_2 зіткнення другої сфери з н рухомою.

Подія 0 відбувається, якщо $(\vec{r}_2-\vec{r}_1)^2=(R_2+R_1)^2$. Для знаходже ня моменту часу dt_0 для події 0 необхідно розв'язати відносно велич ни dt таке квадратне рівняння:

$$(\vec{v}_2-\vec{v}_1)^2\cdot(dt)^2+2\cdot(\vec{r}_{20}-\vec{r}_{10},\vec{v}_2-\vec{v}_1)\cdot dt+ \\ +((\vec{r}_{20}-\vec{r}_{10})^2-(R_2+R_1)^2)=0. \tag{2.8}$$

Якщо підкореневий вираз від'ємний, то рухомі сфери для зад них початкових положень і швидкостей не зіштовхуються; це мож врахувати, поклавши значення dt_0 таким, що дорівнює дуже великом додатному числу. У противному разі події 0 відповідає менший з н від'ємних коренів рівняння (2.8); якщо обидва корені рівняння (2.8) і нують і від'ємні, то значення dt_0 вважають таким, що дорівнює ду великому додатному числу.

Таким чином, у будь-який наявний момент часу є можливість о числити інтервали часу для появи подій 0, 1 і 2. Насправді ж відб деться лише та подія, яка є найближчою за часом. Розглянемо інте вал dt_{min} = min (dt_0 , dt_1 , dt_2). Він визначить ту подію, яка неминуч відбудеться. Пересунемо обидві рухомі сфери відповідно до знайде ного інтервалу часу dt_{min} . Якщо найближча за часом подія має номе 1 або 2, то виконаємо відповідне перетворення швидкості сфери, як бере участь в зіткненні, і знову обчислимо інтервал часу dt_{min}. Якщ найближчою за часом є подія 0, то розв'язання закінчується.

Нижче наведено варіант обчислень за другим способом алгори тмічною мовою TurboPascal 5.5.

```
Program dwe_molekuly;
const rad1 = 0.5; rad2 = 0.8;
rad0 = 20; dtw0 = 1.0;
```

Радіуси сфер

```
Type   t_vector = array[1..3] of real;
```

Тип вектора

```
Const
v0 : t_vector = (0, 0, 0);
r0 : t_vector = (0,0,0);
```

Задані в умові вектори швидкостей і початкових положень


```
v1 : t_vector = (2, 10, 0);
r1 : t_vector = ( 5, 0, 0);
v2 : t_vector = (5, 5, 0);
r2 : t_vector = ( 17, 0, 3);
Var
  i                          : byte;
  t, dt, dt1, dt2, dt0, dtw  : real;
  k1, k2                     : integer;
  lamnda                     : real;
  indikator                  : byte;
```

Глобальні змінні, які використовують при розв'язанні

```
function sk_p(a,b: t_vector):real;
begin
sk_p := a[1]*b[1]+a[2]*b[2]+a[3]*b[3];
end;
```

Функція для обчислення скалярного добутку двох векторів

```
function t_in(r1,v1,r2,v2 : t_vector; rad1,
rad2 : real): real;
var a, b, c, d, x : real;
begin
a := 0; for i := 1 to 3 do
            a := a + sqr(v2[i]-v1[i]);
b := 0; for i := 1 to 3 do
            b := b + (v2[i]-v1[i])*(r2[i]-r1[i]);
c := 0; for i := 1 to 3 do
            c := c + sqr(r2[i]-r1[i]);
c := c - sqr(rad1 - rad2);
d := b*b-a*c; x := (-b+sqrt(d))/a; t_in := x;
end;
```

Функція для обчислення інтервалу часу до зіткнення рухомої сфери з нерухомою відповідно з формулою (3)

```
function t_out(r1,v1,r2,v2 : t_vector; rad1,
rad2 : real): real;
var a, b, c, d, x1, x2, t : real;
begin
a := 0; for i := 1 to 3 do
            a := a + sqr(v2[i]-v1[i]);
b := 0; for i := 1 to 3 do
            b := b + (v2[i]-v1[i])*(r2[i]-r1[i]);
c := 0; for i := 1 to 3 do
            c := c + sqr(r2[i]-r1[i]);
c := c - sqr(rad1 + rad2);
d := b*b-a*c; if d <= 0 then t := 1e30
                        else begin
x1 := (-b+sqrt(d))/a; x2 := (-b-sqrt(d))/a;
t := x1; if x2 < x1 then t := x2;
if t < 0 then t :=1e30;
```

Функція для обчислення інтервалу часу до зіткнення рухомих сфер за рівнянням (5)

Якщо дискримінант є від'ємним або обидва корені від'ємні, то обчислений інтервал часу приймають таким, що дорівнює великому числу (1e30)

`end;`	
`t_out := t;` `end;`	
`BEGIN`	Основна програма
`t := 0; k1 := 0; k2 := 0;`	Стартові значення часу і кількості зіткнень
`repeat`	Починається цикл

`dt1 := t_in(r1,v1,r0,v0, rad1, rad0);`
`dt2 := t_in(r2,v2,r0,v0, rad2, rad0);`
`dt0 := t_out(r1,v1,r2,v2, rad1, rad2);`
	Обчислення інтервалів часу до моментів подій 0, 1 і 2

`if (dt0 <= dt1) and (dt0 <= dt2) then begin`
`indikator := 0; dt := dt0; end;`
`if (dt1 < dt2) and (dt1 < dt0) then begin`
`indikator := 1; dt := dt1; end;`
`if (dt2 <= dt1) and (dt2 < dt0) then begin`
`indikator := 2; dt := dt2; end;`
`if (dtw < dt) then begin`
`indikator := 3; dt := dtw; dtw := dtw0;`
`end`
`else dtw := dtw - dt;`
	З'ясування вигляду події, найближчої за часом
	
	Подія 3 – допоміжна; вводиться для зменшення обчислювальних помилок

`t := t+dt;`	Нарощування глобального часу
`for i := 1 to 3 do r1[i] := r1[i] + dt*v1[i];` `for i := 1 to 3 do r2[i] := r2[i] + dt*v2[i];`	Переміщення обох сфер згідно з рівнянням (1)

`case indikator of`
`1: begin`
`lambda := 2*sk_p(v1,r1)/sk_p(r1,r1);`
`for i := 1 to 3 do v1[i] := v1[i] - lambda*r1[i];`
`k1 := k1 + 1;`
`end;`
`2: begin`
`lambda := 2*sk_p(v2,r2)/sk_p(r2,r2);`
`for i := 1 to 3 do v2[i] := v2[i] - lambda*r2[i];`
`k2 := k2 + 1;`
`end;`
`end;`
	Відбиття тієї чи іншої рухомої сфери від нерухомої. Супроводжується підрахунком кількості зіткнень

`writeln(dt1:10:6,' ',dt2:10:6,' ',dt:10:6);`	Проміжне виведення на

`writeln(t:17:12,' ',k1,' ',k2);`	екран (не є необхідним)
`until indikator = 0;`	Скінчення циклу; визначається тим, що найближчою за часом є подія 0.
`writeln(' Ћвўґв : ',t:10:5,' ',k1,' ',k2);`	Виведення рішення на екран. Додатково виводяться кількості зіткнень кожної з рухомих сфер з нерухомою.
`readln;`	Закінчення програми
`END.`	

В іншому варіанті умови пропонується підрахувати кількість зіткнень рухомих сфер з нерухомою до моменту зіткнення рухомих сфер між собою. Відповідь за заданих умов виглядає таким чином:

$t = 1862.15981355\ C^0$, $k1 = 531$, $k2 = 449$.

Запропонована задача є найпростішою з кола задач, пов'язаних з поведінкою великої кількості пружних рухомих сфер. Оптимальні алгоритми розглянуті, наприклад, у посібнику [5].

2.8. Рекурентна послідовність

2006 рік

Умова

Члени послідовності пов'язані співвідношенням

$$a_n = \frac{(2+\sqrt{3})\cdot(a_{n-1}+a_{n+1})-(a_{n-2}+a_{n+2})}{2\cdot(1+\sqrt{3})},$$

причому $a_0 = 2,\ a_1 = 1+\sqrt{3},\ a_2 = 3,\ a_3 = 3$. Знайти a_N.

Тест : N=131313137.

Число N вводять з клавіатури (N <= 2·10⁹).

Розв'язання можна здійснити кількома способами, але в кожному випадку початкове співвідношення треба привести або до вигляду оберненого рівняння

$$a_{n+4}-(2+\sqrt{3})\cdot a_{n+3}+2\cdot(1+\sqrt{3})\cdot a_{n+2}-(2+\sqrt{3})\cdot a_{n+1}+a_n=0, \qquad (2.9)$$

або до вигляду рекурентної послідовності

$$a_{n+4}=(2+\sqrt{3})\cdot a_{n+3}-2\cdot(1+\sqrt{3})\cdot a_{n+2}+(2+\sqrt{3})\cdot a_{n+1}-a_n. \qquad (2.10)$$

<u>Спосіб 1</u> – суто олімпіадний, потребує лише ініціативи. Використавши співідношення (2.10), пропонується передивитись по-

слідовність, що виникає, для достатньо великої кількості членів і побачити, що послідовності мають ніби період:

$a_0 = 2$, $a_1 = 2.7320508076$, $a_2 = 3$, $a_3 = 3$, $a_4 = 3$, $a_5 = 3.2679491924$,

$a_6 = 4$, $a_7 = 5.2679491924$, $a_8 = 7$, $a_9 = 9$, $a_{10} = 11$, $a_{11} = 12.7320508076$,

$a_{12} = 14$, $a_{13} = 14.7320508076$, $a_{14} = 15$, $a_{15} = 15$, $a_{16} = 15$,

$a_{17} = 15.2679491924, \ldots$

Виникає гіпотеза :

$$a_n = n - k + a_k, \tag{2.11}$$

де k – залишок від ділення n на 12.

Її треба перевірити для невеликих значень номера n. Переконавшись у тому, що обчислення з допомогою рекурентної послідовності (2.10) і за формулою (2.11) збігаються, з великою часткою впевненості можна вважати, що формула (2.11) дає вірний результат. Для інженерних цілей ці міркування є цілком достатніми. Треба також відмітити, що застосовувати співвідношення (2.10) для заданого великого тестового значення n не можна, оскільки похибки, що накопичуються, призводять до абсолютно невірного результату.

Спосіб 2. Основується на властивостях обернених рівнянь. Виходячи з рівняння (2.9), складаємо таке алгебричне рівняння:

$$\lambda^4 - (2+\sqrt{3})\cdot\lambda^3 + 2\cdot(1+\sqrt{3})\cdot\lambda^2 - (2+\sqrt{3})\cdot\lambda + 1 = 0. \tag{2.12}$$

Розділивши рівняння (2.12) на λ^2 і зробивши заміну $z = \lambda + \frac{1}{\lambda}$, приводимо рівняння (2.12) до вигляду квадратного:

$$z^2 - (2+\sqrt{3})\cdot z + 2\cdot\sqrt{3} = 0. \tag{2.13}$$

Згідно з теоремою Вієта, $z_1 = 2$, $z_2 = \sqrt{3}$.

Послідовно розв'яжемо два квадратних рівняння:

$$\lambda + \frac{1}{\lambda} = 2 \Rightarrow \lambda_1 = \lambda_2 = 1;$$

$$\lambda + \frac{1}{\lambda} = \sqrt{3} \Rightarrow \lambda_3 = \frac{\sqrt{3}}{2} + i\cdot\frac{1}{2}, \ \lambda_4 = \frac{\sqrt{3}}{2} - i\cdot\frac{1}{2}, \text{ де } i = \sqrt{-1}.$$

Тому загальне рішення оберненого рівняння (2.9) є можливим уявити в такому вигляді:

$$a_n = C_1\cdot 1 + C_2\cdot n\cdot 1 + C_3\cdot\cos\frac{n\cdot\pi}{6} + C_4\cdot\sin\frac{n\cdot\pi}{6}. \tag{2.14}$$

Для знаходження констант C_1, C_2, C_3, C_4 скористаємося даними умовами і складемо систему лінійних рівнянь

$$\begin{cases} C_1 + C_3 = 2, \\ C_1 + C_2 + C_3\cdot\frac{\sqrt{3}}{2} + C_4\cdot\frac{1}{2} = 1+\sqrt{3}, \\ C_1 + 2\cdot C_2 + C_3\cdot\frac{1}{2} + C_4\cdot\frac{\sqrt{3}}{2} = 3, \\ C_1 + 3\cdot C_2 + C_4 = 3. \end{cases} \tag{2.15}$$

Легко знайти, что система (2.15) має таке розв'язання:

$C_1 = C_4 = 0$, $C_2 = 1$, $C_3 = 2$.

Часткове розв'язання рівняння (2.9), яке задовольняє задані початкові умови, з урахуванням періодичності косинуса має вигляд

$$a_n = n + 2\cdot\cos\frac{n\cdot\pi}{6} = n + 2\cdot\cos\frac{k\cdot\pi}{6}, \tag{2.16}$$

де k = n mod 12.

Програмування за формулою (2.16) труднощів не викликає; розв'язок при цьому є точним і обгрунтованим.

2.9. Найменший периметр

2006 рік

Умова

У текстовому файлі **Z3_2t.txt** міститься така інформація: в першому рядку – кількість точок і довжина s боку квадрата. У другому рядку – координата X і координата Y першої з випадкових точок, в третьому – координати третьої з точок, і т.д. Точки заповнюють квадрат з вершинами **(0,0), (0,s), (s,0) і (s,s).** Кожні три точки є вершинами деякого трикутника. Знайти номери вершин трикутника з найменшим периметром, а також величину цього периметра.

Найпростіший спосіб розв'язання полягає в простому переліку заданих точок, узятих по три. Якщо кількість точок невелика (наприклад, є меншою ста – двохсот), то на сучасних ЕОМ цілком можна розв'язати цю задачу. Однак у файлі **Z3_2t.txt** кількість точок має порядок 100000 ... 200000. Тому необхідний спеціальний метод, пов'язаний з введенням підсистем. Припустимо, що кількістю точок є точний квадрат деякого числа m. Розіб'ємо сторону квадрата, яка виходить з початку координат по осі OX, точками з координатами $x_0=0\cdot h$, $x_1=1\cdot h, \ldots, x_i=i\cdot h, \ldots, x_{m-1}=(m-1)\cdot h$, де h = s / (m-1). З кожної з цих точок побудуємо перпендикуляр до осі OX; кожний з перпендикулярів аналогічним чином розіб'ємо точками в кількості m. Отримані m^2 точок є вузлами прямокутної сітки. Найменша відстань між двома довільними точками за побудовою дорівнює h. Якщо змінити дозволеним чином (не виходячи за межі заданого квадрата) положення якої-небудь точки, то найменша відстань може бути лише меншою, ніж величина h. Таким чином, при довільному заповненні квадрата n точками найменша відстань не перевищує кроку h. Інтуїтивно очевидно, що найменший периметр, про який йдеться в задачі, має порядок трьох мінімальних відстаней. Більш детальний розгляд показує, що ймовірність того, що найменший периметр при рівномірному розподіленні точок перевищує h, при достатньо великій кількості n практично дорівнює нулю.

У загальному випадку оберемо число m таким, що дорівнює цілій частині кореня квадратного з кількості точок n і побудуємо сітку, як було показано вище. Для кожної квадратної комірки сітки знайдемо номери точок, які їй належать. Кожна комірка утворює підсистему початкової системи заданих точок; підсистема з мультиіндексом (i,j) містить у собі список номерів точок, які їй належать. У середньому довжина списку дорівнює одиниці. Організація підсистем потребує кількості операцій, пропорційній кількості підсистем m^2, тобто має порядок O(n).

Наступний крок полягає в переліку всіх тих підсистем, для яких жодна із сторін не лежить на стороні початкового квадрата. Для кожної з них утворюється масив номерів точок, які входять у підсистему і сусідні з нею; їх загальна кількість дорівнює шести. Розмір цього робочого масиву має порядок 9, однак для надійності цей розмір бажано вибирати більшим (наприклад, сто), що практично не відобразиться на потрібній пам'яті.

Для робочого масиву точок звичайним перебором по три обчислюють номери вершин, які входять у масив, і для кожної трійки знаходять периметр відповідного трикутника, який порівнюють із мінімальним на наявний момент. У середньому кількість таких трикутників дорівнює 84.

Таким чином, загальна кількість операцій є пропорційною величині K•O(n), де множник K має порядок 100. Головне – це те, що згідно із системним підходом загальна кількість операцій зростає лінійно.

Множник K можна дещо зменшити шляхом оптимізації перегляду трійок вершин. Наприклад, можна розглядати не всі дев'ять сусідніх комірок, а лише чотири, три з яких знаходяться справа або зверху у відношенні до підсистеми з мультиіндексом (i,j). Можна також серед трійок вершин залишати для розгляду лише ті, серед яких хоча б одна належить підсистемі, яка розглядається; тоді множник K зменшується до величини порядку 3. Однак це пов'язане з деяким ускладненням програмування (тобто з витратами часу, дуже цінного в умовах олімпіади) і кінець кінцем несуттєве.

Далі наведені необхідні фрагменти програми на мові Delphi (6 або 7).

`type`	Необхідні типи:

`t_point_one = record`
`x, y : double;`
`end;`
	Тип однієї точки, яку задають її координатами

`t_points = array of t_point_one;`	Тип динамічного масиву точок
`t_pointer = ^t_c;`	Тип покажчика на список номерів точок, які належать окремій підсистемі

`t_c = record`
`nomer : longint;`
`adres : t_pointer;`
`end;`
	Тип окремого елемента списку

`t_cells = array of array of t_pointer;`	Тип двовимірного динамічного масиву підсистем

Глобальні змінні:

`n : longint;`	Кількість заданих точок.
`s : double;`	Довжина сторони квадрат.
`points : t_points;`	Динамічний масив заданих точок.
`ii, jj, kk : longint;`	Індекси для перебору по три.
`pmin : double = 1e10;`	Стартове значення мінімального периметра.
`m : longint;`	Кількість розбиттів однієї сторони квадрата.
`d_rebra : double;`	Довжина ребра комірки.
`cells : t_cells;`	Масив комірок (підсистем).
`arr_temp : array[1..10000] of longint;`	Робочий масив номерів точок, серед яких проводиться пошук по три

Найпростіші функції та процедури, які використовують при розв'язанні:

```
Function dlina(i, j : longint) : double;
var x1, y1, x2, y2 : double;
begin
with points[i] do begin
x1 := x; y1 := y; end;

with points[j] do begin
x2 := x; y2 := y; end;

dlina := sqrt(sqr(x2-x1)+sqr(y2-y1));

end;
```

Процедура-функція для обчислення відстані між точками з номерами i та j

```
function perimetr(i, j, k : longint) :
double;
```

Процедура-функція для обчислення периметра трикутника,

```
begin
perimetr := dlina(i,j) + dlina(j,k) +
dlina(k,i);
end;
```

вершини якого мають номери i, j та k

```
procedure zapoln_cells;
var i, j, k, kk : longint; u, p : t_pointer;
begin

for i := 0 to m do for j := 0 to m do
cells[i,j] := nil;
```

Процедура побудови динамічного масиву списків номерів точок, які належать окремим коміркам (підсистемам). Спочатку елементи масиву покладають такими, що дорівнюють Nil.

```
for k := 1 to n do begin
i := trunc(m * points[k].x / s);
j := trunc(m * points[k].y / s);
u := cells[i,j];
new(p);
p.nomer := k;
p.adres := u;
cells[i,j]:= p;
            end;

end;
```

Переглядають всі точки; для кожної з них визначають мультиіндекс відповідної підсистеми; для знайденої підсистеми дописують у список номер даної точки (в даному прикладі вибрано варіант побудови стеку, що не є принциповим)

```
procedure prowerka_ij(i, j : word;
var nn : longint);
var
i1, j1 : word;
p     : t_pointer;
begin
nn := 0;
for i1 := i - 1 to i + 1 do
  for j1 := j - 1 to j + 1 do begin
p := cells[i1, j1];
               while p <> nil do begin
inc(nn); arr_temp[nn] := p.nomer;
p := p.adres;
                          end;
                    end;
end;
```

Процедура перевірки непустоти робочого масиву і побудови робочого масиву arr_temp

Для мультиіндексу (i, j) розглядається комірка із цим мультиіндексом і всі сусідні. Збирають у загальний масив arr_temp номери точок, які належать цим підсистемам. Одночасно знаходять розмір nn робочого масиву

Підготовчий етап розв'язання:

```
procedure TForm1.FormCreate(Sender:
TObject);
```

Побудова динамічного масиву підсистем списків номерів точок


```
var f : textfile; i, j, k : longint; stroka :
string;
begin
assignfile(f,'points.txt'); reset(f);
read(f,n, s); setlength(points, n+2);
```

Встановлення довжини динамічного масиву

```
for i := 1 to n do with points[i] do
read(f,x,y);
                    closefile(f);
```

Побудова масиву точок, заданих координатами

```
{-------------------------------------------------}
m := trunc(sqrt(n)); d_rebra := s / m;
```

Обчислення параметрів системи комірок сітки

```
{-------------------------------------------------}
Label1.caption := ' n ='+inttostr(n)+
'  m= '+inttostr(m);
```

Виведення на екран

```
setlength(cells, m+2);
for i := 0 to m+1 do setlength(cells[i],
m+2);
```

Встановлення довжини двовимірного динамічного масиву комірок

```
zapoln_cells;
end;
```

Заповнення масиву підсистем

Вирішення задачі:

```
procedure TForm1.Button2Click(Sender:
TObject);
var i, j : word; ip, jp, kp, k1, k2, k3 : Longint;
p : double;
begin
for i := 1 to m - 1 do for j := 1 to m-1 do begin

prowerka_ij(i, j, n_temp);
```

Перевіряють комірки, які не прилягають до сторін початкового квадрата. Будують робочий масив arr_temp

```
      for k1 := 1 to n_temp - 2 do
          for k2 := k1 + 1 to n_temp - 1 do
                for k3 := k2 + 1 to n_temp do
begin
ip := arr_temp[k1];
jp := arr_temp[k2];
kp := arr_temp[k3];
```

Перевіряють трійки елементів робочого масиву

```
p := perimetr(ip, jp, kp);

                               if p < pmin then
begin
```

Для трійки вершин, що розглядають, обчислюють периметр трикутника із цими вершинами і порівнюють із знайденим

```
pmin := p; ii := ip; jj := jp; kk := kp;        | мінімальним.
end;                                            |
end;                                            |
                    end;                        |
```

Після того, як виконано вказані дії, слід вивести або на екран, або в файл величини, які шукають.

Загальний час роботи програми на ЕОМ із тактовою частотою 660 МГц при використанні цього алгоритму дорівнює 12 секундам, якщо кількість точок – 200 000.

Бібліографічний список

1. Вирт Н. Алгоритмы и структуры данных. – СПб, 2001. – 352 с.
2. Применение ЭВМ для решения задач дискретной математики / А.Ю. Соколов, М.Л. Угрюмов, В.А. Халтурин, Ю.К. Чернышев. – Х.: ХАИ, 2003 . – 78 с.
3. Дискретная математика. Применение ЭВМ при выполнении лабораторных работ: Учеб. пособие / Ю.К. Чернышев, М.Л. Угрюмов, В.А. Халтурин, О.В. Яровая – Х.: ХАИ, 2004. – 60 с.
4. Чернышев Ю.К. Решение задач имитационного моделирования поведения большого количества модельных частиц. – Х.: ХАИ, 2006. – 60 с.
5. Чернышев Ю.К., Левин С.С. Основные положения нелинейной динамики. Примеры решения задач на ЭВМ. – Х.: ХАИ, 2006. – 52 с.