На плоскости задано множество точек с целочисленными координатами, никакие две из которых не совпадают и никакие три не лежат на одной прямой. Необходимо найти количество треугольников, обладающих следующими свойствами:

1) все вершины треугольника принадлежат заданному множеству;

2) ни одна вершина не лежит на осях координат;

3) треугольник не пересекается с осью Oy, но пересекается с осью Ox.

Напишите эффективную по времени и по используемой памяти программу для решения этой задачи.

Программа считается эффективной по времени, если при увеличении количества точек в k раз время работы возрастает не более чем в k раз. Программа считается эффективной по памяти, если размер памяти для хранения всех необходимых данных не зависит от количества точек и не превышает 1 килобайта.

Перед текстом программы кратко опишите алгоритм решения и укажите язык программирования и его версию.

**Входные данные**

В первой строке задаётся N – количество точек в заданном множестве. Каждая из следующих строк содержит два целых числа x и y – координаты очередной точки. Гарантируется, что 1 ≤ N ≤ 10000, –1000 ≤ x, y ≤ 1000, никакие две точки не совпадают, никакие три не лежат на одной прямой.

*Пример входных данных*:

4

6 6

−8 8

−9 −9

−7 5

**Выходные данные**

Необходимо вывести единственное число: количество удовлетворяющих требованиям треугольников.

*Пример выходных данных для приведённого выше примера входных данных*:

1

Дан набор из *N* целых положительных чисел. Из этих чисел формируются все возможные пары (парой считаются два элемента, которые находятся на разных местах в наборе, порядок чисел в паре не учитывается), в каждой паре вычисляется сумма элементов. Необходимо определить количество пар, для которых полученная сумма делится на 7.

Напишите эффективную по времени и по памяти программу для решения этой задачи.

Программа считается эффективной по времени, если при увеличении количества исходных чисел *N* в *k* раз время работы программы увеличивается не более чем в k раз.

Программа считается эффективной по памяти, если память, необходимая для хранения всех переменных программы, не превышает 1 Кбайт и не увеличивается с ростом *N*.

Максимальная оценка за правильную (не содержащую синтаксическихошибок и дающую правильный ответ при любых допустимых входных данных) программу, эффективную по времени и по памяти, – 4 балла.

Максимальная оценка за правильную программу, эффективную только по времени или только по памяти, – 3 балла.

Максимальная оценка за правильную программу, не удовлетворяющую требованиям эффективности, – 2 балла.

Вы можете сдать одну или две программы решения задачи. Если Вы сдадите две программы, каждая из них будет оцениваться независимо от другой, итоговой станет бо́льшая из двух оценок.

Перед текстом программы кратко опишите алгоритм решения. Укажите использованный язык программирования и его версию.

Описание входных и выходных данных

В первой строке входных данных задаётся количество чисел N (1 ≤ N ≤ 1000).

В каждой из последующих N строк записано одно натуральное число, не превышающее 10 000.

*Пример входных данных:*

5

1

3

6

11

1

*Пример выходных данных для приведённого выше примера входных данных:*

3

Из 5 чисел можно составить 10 пар. В данном случае у трёх пар сумма делится на 7: 1 + 6, 1 + 6 (в наборе две единицы, поэтому пару 1 + 6 можно составить двумя способами), 3 + 11.

**Пояснение.**

Чтобы треугольник не пересекался с осью *Oy* и пересекался с осью *Ox*, его вершины должны лежать в одной полуплоскости относительно *Oy* и в разных относительно *Ox*. Получается, что вершины треугольника должны лежать

в первой и четвёртой либо во второй и третьей четвертях, причём в одной из этих четвертей должны лежать две вершины, в другой — одна.

Зная количество точек в каждой четверти, можно подсчитать количество искомых треугольников. Например, если в первой четверти лежит *n*1 точек, а в четвёртой – *n*4 точек, то количество треугольников, у которых две вершины лежат в первой четверти, а одна — в четвёртой, равно (*n*1(*n*1–1)/2) \* *n*4 = *n*1(*n*1–1)*n*4/2.

Если известны величины *n*1, *n*2, *n*3, *n*4, показывающие количество точек в каждой четверти, то общее количество треугольников равно (*n*1(*n*1-1)*n*4 + *n*4(*n*4-1)*n*1 + *n*2(*n*2-1)*n*3 + *n*3(*n*3-1)*n*2) / 2.

**Пример правильной и эффективной программы на языке Паскаль**

program P27;

  var

    N: **integer**; {количество **то**чек}

    x,y: **integer**; {координаты очередной **то**чки}

    n1, n2, n3, n4: **integer**;

      {количество **то**чек по четвертям}

    s: **integer**; {количество треугольников}

    i: **integer**;

begin

  readln(N);

  n1:=0; n2:=0; n3:=0; n4:=0;

  for i:=1 to N do begin

    readln(x,y);

    if (x>0) and (y>0) then n1 := n1+1;

    if (x<0) and (y>0) then n2 := n2+1;

    if (x<0) and (y<0) then n3 := n3+1;

    if (x>0) and (y<0) then n4 := n4+1;

  end;

  s := (n1\*n4\*(n1+n4-2) + n2\*n3\*(n2+n3-2)) div 2;

  writeln(s)

end.

**Пример правильной, но неэффективной программы на языке Паскаль.**

begin

readln(N);

s := 0;

for i := 1 to N do read(a[i, 1], a[i, 2]);

for i := 1 to N do

for j := i+1 to N do

for k := j+1 to N do

if (a[k, 1]<0) and (a[j, 1]<0) and (a[i, 1]<0) then begin

if ((a[k, 2]<0) and (a[j, 2]>0)) or ((a[k, 2]<0) and (a[i, 2]>0)) or ((a[k, 2]<0) and (a[j, 2]>0)) then

s := s + 1;

if ((a[k, 2]>0) and (a[j, 2]<0)) or ((a[k, 2]>0) and (a[i, 2]<0)) or ((a[k, 2]>0) and (a[j, 2]<0)) then

s := s + 1;

end;

if (a[k, 1]>0) and (a[j, 1]>0) and (a[i, 1]>0) then begin

if ((a[k, 2]<0) and (a[j, 2]>0)) or ((a[k, 2]<0) and (a[i, 2]>0)) or ((a[k, 2]<0) and (a[j, 2]>0)) then

s := s + 1;

if ((a[k, 2]>0) and (a[j, 2]<0)) or ((a[k, 2]>0) and (a[i, 2]<0)) or ((a[k, 2]>0) and (a[j, 2]<0)) then

s := s + 1;

end;

writeln(s);

end.

**Пояснение.**

Разобьём все числа исходного набора на 7 групп по значению остатка от деления на 7 и подсчитаем количество чисел в каждой группе. Сами числа можно не хранить, достаточно при вводе определить остаток от деления очередного числа на 7 и увеличить соответствующий счётчик. Таким образом, независимо от количества чисел в исходном наборе, после чтения исходных данных для хранения необходимой информации хватит массива из 7 элементов и программа получится эффективной по памяти.

Чтобы сумма двух чисел делилась на 7, они оба должны делиться на 7 либо сумма их остатков от деления на 7 должна быть равна 7. Зная количество элементов для каждого остатка, можно определить количество пар.

Ниже приведена реализующая описанный выше алгоритм программа на языке Паскаль (использована версия PascalABC)

**Пример правильной и эффективной программы на языке Паскаль**

var

    N: integer; {количество чисел}

    a: integer; {очередное число}

    d: array [0..6] of integer; {группы по остаткам}

    s: integer; {количество пар}

    i: integer;

begin

    for i:=0 to 6 do d[i]:=0;

    readln(N);

    for i:=1 to N do begin

        readln(a);

        inc(d[a mod 7])

    end;

    s := d[0]\*(d[0]-1) div 2;

    for i:=1 to 3 do s := s + d[i]\*d[7-i];

    writeln(s)

end.

Возможно также «лобовое» решение: запишем все исходные числа в массив, переберём все возможные пары и подсчитаем количество подходящих. Такое решение не является эффективным ни по памяти (требуемая память зависит от размера исходных данных), ни по времени (количество возможных пар, а значит, количество действий и время счёта с ростом количества исходных элементов растёт квадратично). Такая программа оценивается не выше 2 баллов.

Ниже приведена реализующая описанный выше алгоритм программа на языке Паскаль (использована версия PascalABC)

**Пример правильной, но неэффективной программы на языке Паскаль**

var

    N: integer; {количество чисел}

    a: array [1..1000] of integer; {исходные данные}

    s: integer; {количество пар}

    i,j: integer;

begin

    readln(N);

    for i:=1 to N do readln(a[i]);

    s :=0;

    for i := 1 to N-1 do begin

        for j := i+1 to N do begin

            if (a[i]+a[j]) mod 7 = 0

                then s := s + 1

        end

    end;

    writeln(s)

end.