

Цю сторінку замінити на сторінку з
титulloю, набраною якимись іншими
засобами

Цю сторінку замінити на сторінку з
«ВЫХОДНЫМИ сведениями», набраною
якимись іншими засобами

1 Обласна інтернет-олімпіада 2013/14 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №10).

Задача А. «Тор»

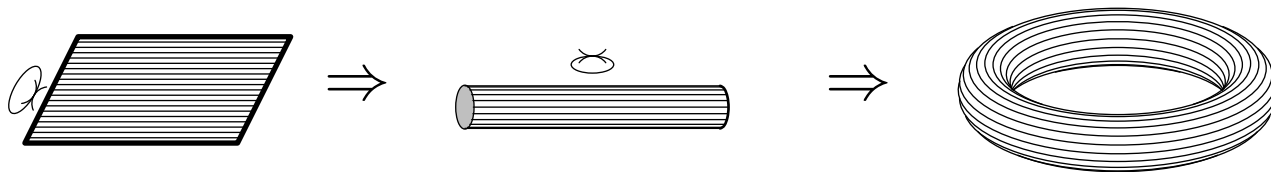
Вхідні дані: Клавiатура (stdin)

Результати: Екран (stdout)

Обмеження часу: 1 сек

Обмеження пам'яті: 64 мегабайти

Як відомо, *тор* — це поверхня бублика, яку можна отримати таким чином: узяти прямокутник розміром n клітинок по вертикалі на m клітинок по горизонталі, склеїти верхню сторону з нижньою (отримається циліндр), потім закрутити циліндр у бублик і склеїти ліву сторону з правою.



Назвемо дві клітинки *сусідніми*, якщо вони мають спільну сторону. Нехай за одну секунду можна перейти з клітинки до будь-якої сусідньої з нею. За який мінімальний час можна потрапити з клітинки $(r_1; c_1)$ у клітинку $(r_2; c_2)$? Перше число у позначенні клітинки — номер рядка початкового прямокутника, друге — номер стовпчика.

Вхідні дані Програма повинна прочитати зі стандартного входу (клавіатури) шість натуральних чисел, у порядку n, m, r_1, c_1, r_2, c_2 . Виконуються обмеження: $2 \leq n, m \leq 1\,000\,000\,000$, $1 \leq r_1, r_2 \leq n$, $1 \leq c_1, c_2 \leq m$.

Результати Програма має вивести на стандартний вихід (екран) єдине ціле число — мінімальний час.

Приклади

Вхідні дані	Результати
10 10 5 5 1 1	8
10 10 9 9 1 1	4

Розбір задачі Необхідно дістатися з точки (r_1, c_1) у точку (r_2, c_2) по «зацикленому» простору. Скориставшись відомим принципом незалежності переміщень, можна побачити, що переміщення в горизонтальному і вертикальному напрямках не залежать одне від одного. Зрозуміти це можна, розглядаючи переміщення як перетин вертикальних і горизонтальних сторін клітинки. Не залежно від траєкторії необхідно перетнути лише певну конкретну кількість вертикальних ліній (позначимо цю кількість Δr) і горизонтальних ліній (відповідно Δc). (Звісно, маючи на увазі, що ми не розглядаємо відверто не мінімальні шляхи, де одна й та ж горизонтальна чи вертикальна лінія перетинається багатократно.) Тоді відповідь на задачу є $\Delta r + \Delta c$, оскільки для зміни будь якої з координат на 1 необхідно затратити 1 секунду.

Є 2 способи переміститися з рядка r_1 у r_2 — перетинаючи край початкового (до згинів і склеювань) прямокутника і не перетинаючи. Вважаємо спочатку, що $r_1 < r_2$. Тоді не перетинаючи край витратимо $(r_2 - r_1)$ сек, а перетинаючи спочатку доберемося до рядка 1 $((r_1 - 1)$ сек), потім до рядка n (1 сек), і з нього у r_2 — $((n - r_2)$ сек). Сумарно $(r_1 - r_2 + n)$ сек. Тобто, треба вибрати мінімум зі значень $(r_2 - r_1)$ (не перетинаючи край) і $(r_1 - r_2 + n)$ (через край). Але це лише для випадку $r_1 < r_2$, а при $r_1 > r_2$ аналогічними міркуваннями отримуються схожі, але інші формули $(r_1 - r_2)$ і $(r_2 - r_1 + n)$. Щоб не задумуватися, яка з координат більша, можна використати формулу

$$\Delta r = \min((r_1 - r_2 + n) \bmod n, (r_2 - r_1 + n) \bmod n)$$

З Δc слід вчинити аналогічно, потім додати $\Delta r + \Delta c$.

Задача В. «Паркет–1»

Вхідні дані:	Клавіатура (stdin)
Результати:	Екран (stdout)
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Щоб зобразити за допомогою паркету Супер-Креативний Візерунок, треба

N_1 дощечок розмірами 1×1 , N_2 дощечок розмірами 2×1 , N_3 розмірами 3×1 , N_4 розмірами 4×1 та N_5 дощечок розмірами 5×1 . Купити можна лише дощечки розмірами 5×1 . Дощечки можна різати, але не можна склеювати. Наприклад, коли потрібні п'ять дощечок 2×1 , їх не можна зробити з двох дощечок 5×1 , але можна з трьох. Для цього дві з них розріжемо на три частини 2×1 , 2×1 та 1×1 кожну, а третю — на дві частини 2×1 та 3×1 . Отримаємо потрібні п'ять дощечок 2×1 , а дві дощечки 1×1 та одна 3×1 підуть у відходи.

Напишіть програму, яка, прочитавши кількості дощечок N_1 , N_2 , N_3 , N_4 та N_5 , знайде, яку мінімальну кількість дощечок 5×1 необхідно купити.

Вхідні дані слід прочитати зі стандартного входу (клавіатури). Це будуть п'ять чисел N_1 , N_2 , N_3 , N_4 та N_5 (саме в такому порядку), розділені пропусками (пробілами).

Результати Єдине число (скільки дощечок треба купити) виведіть на стандартний вихід (екран).

Приклади

Вхідні дані	Результати
0 5 0 0 0	3
1 1 1 1 1	3

Оцінювання Усі кількості невід'ємні; 90 балів (з 250) припадатиме на тести, в яких сумарна кількість $N_1 + N_2 + N_3 + N_4 + N_5$ перебуває в межах від 0 до 20, ще 80 балів — від 100 до 10 000, решта 80 балів — від 500 000 000 до 2 000 000 000.

Здати потрібно одну програму, а не для кожного випадку окремо; різні обмеження вводяться виключно для того, щоб дати приблизне уявлення, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі Задача досить складна тим, що треба дуже акуратно розглянути всі випадки, і разом з тим дуже проста тим, що для остаточної реалізації програми не потрібні ніякі засоби, крім розгалужень та присвоєнь.

Нехай змінні n_1 , n_2 , n_3 , n_4 та n_5 містять потрібні кількості дощечок розмірами 1×1 , 2×1 , 3×1 , 4×1 та 5×1 відповідно, у змінній **res** будемо будувати

відповідь задачі. Протягом роботи алгоритму значення деяких зі змінних $n1$, $n2$, $n3$, $n4$ або $n5$ можуть зменшуватися — по мірі того, як враховуємо відповідні кількості у змінній res , яка наприкінці міститиме остаточну відповідь.

Фрагмент коду	Коментар
$res = n5$	На кожну дощечку розмірами 5×1 неминуче потрібна окрема ціла дощечка.
$res += n4$	На кожну дощечку розмірами 4×1 теж неминуче потрібна окрема дощечка. . .
$n1 -= \min(n1, n4)$. . . але обрізки після відрізання дощечок розмірами 4×1 можна використати як дощечки розмірами 1×1 . Тому надалі можна вважати, що потреба в дощечках 1×1 складає вже не $n1$, а або $n1 - n4$, або 0.
$res += n3$	На кожну дощечку розмірами 3×1 теж неминуче потрібна окрема дощечка. . .
$\text{if } (n2 > n3) :$ $\quad n2 -= n3$. . . причому, при $N_2 > N_3$ просто використовуємо всі обрізки як дощечки розмірами 2×1 . . .
else: $\quad n3 -= n2$ $\quad n2 = 0$ $\quad n1 -= \min(n3 * 2, n1)$. . . а при $N_2 \leq N_3$ спочатку формуємо з цих обрізків абсолютно всі дощечки 2×1 , а із решти — ще й $\min(n3 * 2, n1)$ дощечок 1×1 .
$res += n2 / 2$ $\quad n1 -= \min(n1, n2 / 2)$ $\quad n2 \% = 2$	Оскільки всі дощечки розміром 3×1 раніше вже сформовані, то тепер на кожну пару дощечок розміром 2×1 неминуче потрібна окрема дощечка, причому обрізок можна використати як дощечку 1×1 . <i>Ділення тут цілочисельне (div).</i>

Фрагмент коду	Коментар
<pre>if n2==1: res += 1 n1 -= min(n1,3) n2 = 0</pre>	Якщо при виконанні попереднього етапу кількість дощечок розміром 2×1 була непарна, то зараз треба сформувати останню дощечку розміром 2×1 , причому обрізок можна використати для формування від нуля до трьох дощечок розмірами 1×1 .
<pre>res += n1/5 n1 %= 5</pre>	Якщо, незважаючи на усі попередні кроки, досі є потреба в дощечках розміром 1×1 , формуємо їх, розрізаючи кожну дощечку на 5 частин. <i>Ділення тут цілочисельне (div).</i>
<pre>if n1 > 0: res += 1</pre>	Якщо після попереднього кроку все ще залишилася потреба у дощечках розміром 1×1 (від 1 до 4 штук), для цього достатньо ще <i>однієї</i> дощечки 5×1 .

(Мова коду — Python; “=” (одинарне) — присвоєння; “==” (подвійне) — перевірити, чи дорівнює; “%” — залишок від ділення (mod); “a+=b” — те саме, що a=a+b, тобто до старого значення a додати b і покласти результат у ту саму змінну a; аналогічно “a-=b”, “a%=b”).)

2 II (районний/міський) етап 2013/14 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №14).

Задача А. «Електричка»

Вхідні дані: Клавіатура (stdin)

Результати: Екран (stdout)

Обмеження часу: 1 сек

Обмеження пам'яті: 64 мегабайти

На кожному вагоні електрички є табличка, на якій фарбою написано його номер. Вагони занумеровані натуральними числами $1, 2, \dots, N$ (крайній вагон має номер 1, сусідній з ним — номер 2, і т. д., до крайнього з протилежного боку вагону, який має номер N). Електричка має кабіни з обох боків, і може поїхати хоч 1-им вагоном уперед, хоч N -им.

Run not latex but pdflatex to insert picture

Під час прибуття електрички на платформу, Вітя помітив, що $(i-1)$ штук вагонів електрички проїхали мимо нього, а i -й по порядку зупинився якраз навпроти. Ще він помітив, що на табличці цього вагона написаний номер j . Ще він точно знає (і ці знання відповідають дійсності), що електрички ніколи не бувають ні коротшими 4 вагонів, ні довшими 12 вагонів. Вітя хоче визначити, скільки всього вагонів у електричці. Напишіть програму, яка або знаходитиме цю кількість, або повідомлятиме, що без додаткової інформації це зробити неможливо.

Вхідні дані Програма має прочитати зі стандартного входу (клавіатури) два цілі числа i та j , розділені пропуском. $2 \leq i \leq 12$, $2 \leq j \leq 12$, числа гарантовано задовольняють всі вищезгадані обмеження.

Результати Виведіть на стандартний вихід (екран) одне число — кількість вагонів у електричці. Якщо однозначно визначити кількість вагонів неможливо, виведіть замість кількості число 0.

Приклад

Вхідні дані	Результати
4 2	5

Розбір задачі Розглянемо два випадки:

- електричка їде 1-им вагоном вперед: тоді 1-ий по порядку слідування вагон має напис «№ 1», 2-ий по порядку слідування — напис «№ 2», і т. д.

Тобто, $i=j$, причому незалежно від кількості вагонів електрички.

- електричка їде N -им вагоном вперед: тоді 1-ий по порядку слідування вагон має напис «№ N », 2-ий по порядку слідування — напис «№ $(N-1)$ », і т. д. Тобто, $i+j = N+1$

Звідси можна зробити висновок, що при $i \neq j$ точно має місце 2-й випадок, для якого $N = i+j-1$. Може здатися, ніби при $i=j$ слід виводити 0 («без додаткової інформації визначити кількість вагонів неможливо»); але тут є виключення: при $i=j=12$, відповідь 12 (електрички не бувають довшими 12 вагонів). А при $(i=j)$ and $(j < 12)$, таки виводити 0, бо така ситуація можлива при будь-якому N у межах $j \leq N \leq 12$. Приклад реалізації: ideone.com/vIUkUt

(Повний вміст даного посилання ideone.com/vIUkUt такий:

```
var i, j : integer;  
BEGIN  
  readln(i, j);  
  if i=j then  
    if i=12 then  
      writeln('12')  
    else  
      writeln('0')  
  else  
    writeln(i+j-1)  
END.
```

— кінець цитати посилання ideone.com/vIUkUt)

Програма потребує виконання всього кількох арифметичних дій, тому має асимптотичну складність $\theta(1)$ і виконується практично миттєво.

Розв'язки, які виводили правильну відповідь при $i \neq j$, а при $i=j$ — *завжди* 0, оцінювалися на 180 балів з 200.

Задача В. «Цифрові ріки»

Вхідні дані:	Клавіатура (stdin)
Результати:	Екран (stdout)
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Цифрова ріка — це послідовність чисел, де число, що слідує за числом n , це n плюс сума його цифр. Наприклад, якщо число $n=12345$, то за ним буде йти $12345 + (1+2+3+4+5) = 12360$ і т. д. Якщо перше число цифрової річки N , ми будемо називати її «**річка N** ».

Для прикладу, **річка 480** — це послідовність чисел, яка починається з чисел 480, 492, 507, 519, ..., а **річка 483** — послідовність, що починається з 483, 498, 519, ...

Напишіть програму, яка приймає на вхід два цілих значення k ($1 \leq k \leq 16384$) та N ($1 \leq N \leq 10000$), і виводить k -те число річки N .

Приклад	Вхідні дані	Результати
	4 480	519

Розбір задачі Задача робиться «в лоб», тобто без усяких придумок $k-1$ раз («мінус один», бо треба отримати k -те число з 1-го, а не 0-го) застосовується дія «порахувати й додати суму цифр поточного числа».

Найпоширеніший спосіб рахувати суму цифр числа — у циклі розглядати останню цифру числа (Pascal: $n \bmod 10$, C/C++: $n\%10$), а потім «відсікати» її (Pascal: $n:=n \div 10$, C/C++: $n/=10$). Ці операції треба робити з «копією» (а не самим поточним числом n , яке від «відсікань» втрачається); це може бути забезпечено або присвоєнням у додаткову змінну, або передачею у функцію параметром-значенням (Pascal — без модифікатора `var`). Приклад такого розв'язання — <http://ideone.com/YWcv5B>

Іншим способом є перетворення числа у рядкову величину й подальші звернення до окремих символів-цифр. Деталі сильно відрізняються від конкретної мови програмування, навіть конкретних бібліотек, тому їх важко пояснити теоретично. Дивіться конкретні розв'язки: ideone.com/5poAvj

(Повний вміст даного посилання ideone.com/5poAvj такий:

```
#include <iostream>
#include <cstdio>

using namespace std;

int main(int argc, char* argv[])
```

```
{
    int N, k;
    cin >> k >> N;
    for(int i=1; i<k; i++) {
        char buff[8];
        sprintf(buff, "%d", N);
        for(int j=0; buff[j]!='\0'; j++)
            N += buff[j] - '0';
    }
    cout << N << endl;
    return 0;
}
```

— кінець цитати посилання ideone.com/5poAvj)

(мовою C++) та ideone.com/d2F0rb

(Повний вміст даного посилання ideone.com/d2F0rb такий:

```
{ $mode delphi }

uses SysUtils;

var k, N, i, j : integer; //32-bit due to { $mode delphi }
    s : string;

BEGIN
    Readln(k, N);
    for i:=2 to k do begin
        s := IntToStr(N);
        for j:= 1 to Length(s) do
            N := N + StrToInt(s[j]);
        end;
        writeln(N);
    END.
```

— кінець цитати посилання ideone.com/d2F0rb)

(мовою Pascal із функціями `IntToStr` та `StrToInt` — найбільш лаконічний з усіх наведених).

Протягом перетворень значення поточного числа може стати більшим 32767, тому треба забезпечити, щоб цілий тип був не 16-бітовим, а 32-бітовим. У Pascal: або писати тип `longint`, або здавати не під «Free Pascal», а під «Free Pascal in Delphi mode» (останній варіант, який можна забезпечити також шляхом написання на початку програми рядка `{ $mode delphi }`, робить 32-бітовим тип `integer`). У C/C++ це не повинно бути проблемою, бо просто `int` вже 32-бітовий.

Задача С. «Логічний куб»

Вхідні дані: Клавiатура (stdin)

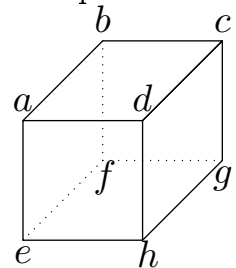
Результати: Екран (stdout)

Обмеження часу: 1 сек

Обмеження пам'яті: 64 мегабайти

Логічний куб — це куб, у вершинах якого знаходяться значення 0 (**false**) або 1 (**true**). Потрібно знайти шлях від однієї заданої вершини до іншої, якщо такого шляху не існує то вивести відповідне повідомлення. В кубі можна проходити через усі ребра, а також через вершини значення яких рівне 1.

Напишіть програму яка знаходить шлях між двома вершинами (якщо він існує), та виводить його у вигляді послідовності вершин куба. Гарантовано, що ці дві задані вершини мають значення 1.



Вхідні дані В першому рядку задаються через пробіл дві вершини куба, це можуть бути дві з наступних маленьких латинських літер: *a, b, c, d, e, f, g, h*. В наступному рядку, послідовно записуються значення кожної з вершин куба (0 або 1). Значення у вершинах перелічені в алфавітному порядку.

Результати Якщо шлях існує, то вивести (без пробілів) послідовність маленьких латинських літер мінімальної довжини, які визначають шуканий шлях. Якщо такого шляху не існує, то вивести рядок "NO".

Приклади

Вхідні дані	Результати
e d 10011011	ead
e d 00011110	NO

Розбір задачі Задачу можна розв'язати шляхом ручного аналізу випадків. Але це страшнуватий спосіб, вартий уваги лише коли нема кращих ідей. За посиланням <http://ideone.com/jjq2Pq> можна бачити трохи модифікований варіант програми, яку здав один з учасників. Цей модифікований варіант набирає 270 балів з 300, що з одного боку немало, з іншого — на ручний розгляд великої кількості випадків явно пішло багато часу, розв'язок все одно

виявився не повним, а шукати помилку в *такому* нагромадженні — украй марудна й невдячна справа. У цьому смислі краще писати осмислений алгоритм, щоб різні вхідні дані оброблялися більш-менш однотипно.

Коли мова йде про мінімальні (за кількістю переходів по ребрам) шляхи, природним є алгоритм *пошуку вшир* (він же *пошук у ширину*); рос. *поиск в ширину*, англ. *breadth first search (BFS)*. Застосовувати BFS треба до неорієнтованого графа, вершини якого — ті і тільки ті вершини куба, значення яких 1 (по яким можна проходити), а ребра — ті й тільки ті ребра куба, що поєднують дві вершини зі значеннями 1. Причому, раз питають не про відстань (як число), а про шляхи (як послідовності вершин), то потрібен варіант BFS, у якому запам'ятовуються батьківські вершини (вони ж попередники), а потім відбувається відновлення шляху зворотнім ходом. Деталі можна знайти в Інтернеті або літературі.

Вищесказане — не таке просте, а куб має лише 8 вершин (що дуже мало). Тому може бути доцільним і повний перебір (наприклад, рекурсивний) усіх можливих шляхів, що не містять повторень вершин. Приклад такого розв'язку — ideone.com/pihZzm

(Повний вміст даного посилання ideone.com/pihZzm такий:

```
program logcube;

{$mode delphi}

uses
  SysUtils;

type SetOfChars = set of 'a'..'h';

var vertex_mark : array['a'..'h'] of char;
    edges_to : array['a'..'h'] of SetOfChars;
    start, c, finish : char;
    min_way : string;

procedure look_from(curr_way : string; can_use : SetOfChars);
var curr_pos, next_pos : Char;
begin
  curr_pos := curr_way[Length(curr_way)];
  if curr_pos = finish then begin
    if Length(curr_way) < Length(min_way) then
      min_way := curr_way;
```

```

end else begin
  for next_pos := 'a' to 'h' do
    if (vertex_mark[next_pos]='1') and (next_pos in can_use) and (next_pos in edges_to)
      look_from(curr_way + next_pos , can_use - [next_pos]);
    end
  end;
end;

begin
  edges_to['a'] := ['b','d','e'];
  edges_to['b'] := ['a','c','f'];
  edges_to['c'] := ['b','d','g'];
  edges_to['d'] := ['a','c','h'];
  edges_to['e'] := ['f','h','a'];
  edges_to['f'] := ['e','g','b'];
  edges_to['g'] := ['f','h','c'];
  edges_to['h'] := ['e','g','d'];
  read(start, c, finish);
  Readln;
  for c := 'a' to 'h' do
    read(vertex_mark[c]);
  min_way := 'no,no,its_impossible';

  look_from(start, ['a'..'h'] - [start]);

  if Length(min_way) > 8 then
    min_way := 'NO';
  Writeln(min_way);
end.

```

— кінець цитати посилання ideone.com/pihZzm)

Детальніше про цей підхід реалізації перебору можна знайти в Інтернеті або літературі за ключовими словами «пошук з поверненнями», «бектрекінг» (рос. «поиск с возвратом», англ. «*backtracking*»).

На жаль, ідея «писати осмислений алгоритм, щоб різні вхідні дані оброблялися однотипно» мало придатна до такої особливості даної задачі, як задання рисунком відповідності ребер куба його вершинам (між якими є ребро і між якими нема). Цю відповідність важко сформулювати правилом («позначки вершин відрізняються або на 4, або на 1, але крім $d \leftrightarrow e$, а крім названих є ще $a \leftrightarrow d$ і $e \leftrightarrow h$ » — можна, але заплутано й незручно). Мабуть, легше задати явний перелік (чи як у розв'язку з попереднього абзацу, чи вписати у текст програми константний масив — матрицю суміжності графа, чи ще якимось). Але це треба ретельно перевіряти на відповідність умові, бо загальні засоби

контролю правильності тут безсилі.

Задача D. «Всюдисущі числа»

Вхідні дані: Клавiатура (stdin)

Результати: Екран (stdout)

Обмеження часу: 3 сек

Обмеження пам'яті: 64 мегабайти

Дано прямокутну таблицю $N \times M$ чисел. Гарантовано, що у кожному окремо взятому рядку всі числа різні й монотонно зростають.

Напишіть програму, яка шукатиме перелік (також у порядку зростання) всіх тих чисел, які зустрічаються в усіх N рядках.

Вхідні дані слід прочитати зі стандартного входу (клавіатури). У першому рядку задано два числа N та M . Далі йдуть N рядків, кожен з яких містить рівно M розділених пропусками чисел (гарантовано у порядку зростання).

Результати виведіть на стандартний вихід (екран). Програма має вивести в один рядок через пробіли у порядку зростання всі ті числа, які зустрілися абсолютно в усіх рядках. Кількість чисел виводити не треба. Після виведення всіх чисел потрібно зробити одне переведення рядка. Якщо нема жодного числа, що зустрілося в усіх рядках, виведення повинно не містити жодного видимого символу, але містити переведення рядка.

Приклад

Вхідні дані	Результати
4 5 6 8 10 13 19 8 9 13 16 19 6 8 12 13 15 3 8 13 17 19	8 13

Оцінювання 20% балів припадатиме на тести, в яких $3 \leq N, M \leq 20$, значення чисел від 0 до 100.

Ще 20% — на тести, в яких $3 \leq N, M \leq 20$, значення чисел від -10^9 до $+10^9$.

Ще 20% — на тести, в яких $1000 \leq N, M \leq 1234$, значення від 0 до 12345.

Решта 40% — на тести, в яких $1000 \leq N, M \leq 1234$, значення від -10^9 до $+10^9$.

Здавати потрібно одну програму, а не чотири; різні обмеження вказані, щоб пояснити, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі Очевидний підхід — брати кожне число 1-го рядка, і шукати його в усіх інших рядках. Якщо знайдене в усіх — поточне число слід включити у відповідь (а якщо ні, то ні). І цей підхід *може* бути правильним. Але *залежно* від того, чи реалізувати його наївно, чи ефективно.

Наївний підхід полягає у тому, щоб для пошуку кожного числа в кожному рядку запускати свій цикл, переглядаючи увесь рядок. Але такий розв'язок потребує $O(N \cdot M^2)$, що може не помістатися у обмеження часу. Власне, $1234^3 \approx 1,88 \cdot 10^9$ простих порівнянь на потужнішому сервері цілком могло б і поміститися у 3 сек. Але в тому й смисл задачі, щоб реалізувати щось ефективніше, і на потужнішому сервері просто ставили б жорсткіший time limit. А такий розв'язок благополучно отримує свої 40 балів зі 100.

Правильний розв'язок №1 Той самий підхід можна реалізувати ефективніше завдяки тому, що кожен рядок гарантовано впорядкований. Адже для впорядкованих масивів можливий *бінарний пошук* (скорочено «*бінпошук*», він же «*двійковий пошук*», він же «*дихотомія*»). Суть бінпошуку така: коли треба знайти значення, починають з середнього (за індексом) елемента; якщо (раптом) він якраз рівний шуканому значенню, пошук успішно завершений; якщо шукане значення більше за середній елемент, то можна відкинути усю ліву половину масиву, а якщо менше — усю праву половину. На наступному кроці (якщо він взагалі потрібен) робиться те саме, але з тією половиною, що залишилась. І так далі. Тобто, за одне порівняння завжди можна відкинути щонайменше половину масиву, і пошук у масиві з 1234 елементами потребує до ≈ 11 порівнянь, а асимптотично — $O(\log M)$. А $N \cdot M$ штук *таких* пошуків чудово поміщаються у обмеження.

Рекомендується знайти в Інтернеті або літературі додаткові деталі щодо бінпошуку. Бо це такий алгоритм, у якому, навіть добре знаючи загальну ідею, легко помилитися й отримати код, який часто працює правильно, але іноді

зациклюється або/та видає неправильні результати.

Тим, хто пише мовою C++, рекомендується вивчити, як у деяких ситуаціях (включно з цією задачею) можна не писати бінпошук самому, якщо навчитися правильно користуватися функцією `lower_bound` бібліотеки `algorithm`.

Правильний розв'язок № 2 Ще один правильний розв'язок (із *кращою* асимптотичною оцінкою $O(N \cdot M)$ проти $O(N \cdot M \cdot \log M)$ у попереднього; але ловити цю відмінність за часом роботи програми не дуже реально, тому попередній розв'язок теж вважається ефективним) — багатократно застосовувати модифікацію *злиття* (рос. «слияние», англ. «merge»).

Суть стандартного злиття така. Нехай є дві (*обов'язково впорядковані!*) послідовності (зазвичай два масиви або два фрагменти одного масиву, але можуть бути й інші послідовності, як-то файли чи зв'язні списки). З них можна легко й швидко сформувати впорядковану послідовність-відповідь, куди входять усі елементи обох заданих послідовностей, якщо діяти так. Призначаємо кожній зі вхідних послідовностей поточну позицію як початок цієї послідовності. І повторюємо у циклі такі дії: (1) беремо (пишемо у послідовність-відповідь) менший з поточних елементів послідовностей; (2) зсуваємо поточну позицію тієї вхідної послідовності (*лише однієї з двох!*), звідки взятий цей менший елемент. Коли одна з послідовностей закінчується, дописуємо у послідовність-відповідь увесь ще не використаний «хвіст» іншої послідовності. Це — *стандартне* злиття, яке робить те саме (але швидше), що дописування однієї з послідовностей після іншої та сортування всього разом. Очевидно, воно працює за час $O(l_1 + l_2)$ (де l_1 та l_2 — довжини вхідних послідовностей). У даній задачі треба інше (спільні елементи). Але, виявляється, під цю задачу теж легко модифікувати злиття. Треба лише при порівнянні поточних елементів різних послідовностей розрізняти три випадки: якщо поточний елемент 1-ої послідовності строго менший за поточний елемент 2-ої, то зсунути поточну позицію 1-ої послідовності (нічого не пишучи у відповідь); якщо строго більший, то зсунути позицію 2-ої послідовності (теж не пишучи); якщо по-

точні елементи рівні, то записати це однакове (спільне) значення у результат та зсунути поточні позиції обох послідовностей. Само собою, при завершенні однієї з послідовностей треба *не* дописувати «хвіст» іншої. Час роботи такого модифікату злиття теж $O(l_1 + l_2)$.

Зрештою, це — злиття *двох* послідовностей; пропонується першого разу злити 1-ий рядок з 2-им, а потім зливати з кожним черговим (3-ім, 4-им, ...) результатом попереднього злиття. *Завдяки* тому, що завжди вибираються лише спільні елементи, розмір кожної з цих послідовностей $\leq M$, тому $N-1 = O(N)$ застосувань злиття забирають час $O(N \cdot M)$. (*Якби* робилося стандартне злиття і розміри послідовностей зростали, оцінка була б значно більшою).

Бажано знайти в Інтернеті або літературі якусь додаткову інформацію про злиття. Її з одного боку достатньо, але з іншого — надто часто пишуть про злиття виключно як про складову рекурсивного сортування злиттям, тоді як тут потрібне *саме злиття*, без рекурсивної надбудови сортування. Крім того, не всі алгоритми, які правильно виконують стандартне злиття, однаково легко модифікуються на вибір лише спільних.

Навіщо в умові згадані групи тестів зі значеннями до 12345? Щоб надати можливість набрати ще ≈ 20 балів тим, хто не додумався ні до одного з розглянутих правильних способів, але знає наступний, зі складністю $O(N \cdot M + V)$, де V — діапазон значень. (Мається на увазі організувати програму як «*if* (розміри малі) *then* (вирішити способом згаданим на самому початку розбору) *else* (вирішити описаним далі способом)». Вхідні дані, коли великі одночасно і кількості, і значення, такий розв'язок все одно не пройде, але можливо набере трохи більше балів.)

Так от, у випадках, коли значення малі, досить ефективний такий підхід. Заведемо масив, *індексами* якого будуть *значення* зі вхідних даних, щоб щоразу, прочитавши деяке v , збільшувати $\text{num}[v]$ на 1 (а спочатку всі значення $\text{num}[\cdot]$ ініціалізуються нулями). Таким чином, після обробки усього двовимірного вхідного масиву кожне значення $\text{num}[v]$ означатиме, скільки разів зустріло-

ся число v ; оскільки у кожному окремо взятому рядку всі числа різні, то «число v зустрілося в усіх рядках» рівносильно $\text{num}[v]=N$.

Чому реалізація правильного алгоритму мовою C++ не вкладається в обмеження часу? Звісно, це можуть бути технічні помилки. Але ще є такий сумний факт, що введення/виведення засобами `istream/ostream` (зокрема, `cin/cout`) працює повільно. Настільки повільно, що перевищувати час буде саме лише читання, взагалі без обробки по суті. Вихід — читати функцією `scanf`, яка з незвички може здаватися незручною, зате швидко працює. І взагалі, є дуже вже багато моментів, коли C++ виграє у Паскаля; тож окремі ситуації, коли неграмотне використання C++ призводить до результатів гірших, ніж Паскаль, можна вважати проявом справедливості. . .

3 Дистанційний тур III (обласного) етапу 2013/14 н. р.

У 2013/14 навч. році III (обласний) етап у Черкаській області складався з двох турів, де один був частково дистанційним (учасники приїздили не до м. Черкаси, а до своїх райцентрів) і проводився на задачах черкаських авторів. Саме він і наведений у даному збірнику. Цей тур позиціонувався одночасно і як змагальний, і як відбірковий; тому рівень складності комплекта задач дещо нижчий, ніж зазвичай на III етапі.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №15).

2-й тур відбувався у Черкасах, але на задачах інших авторів, іншій системі (`ejudge`, але не ejudge.skipo.edu.ua), та й про його дорішування автору даного збірника нічого не відомо; тому він до збірника не включений.

Задача А. «ISBN»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout)
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

ISBN (з англ. International Standard Book Number — міжнародний стандартний номер книги) універсальний ідентифікаційний номер, що присвоюється книзі або брошурі з метою їх класифікації. ISBN призначений для ідентифікації окремих книг або різних видань та є унікальним для кожного видання книги. Даний номер містить десять цифр, перші дев'ять з яких ідентифікують книгу, а остання цифра використовується для перевірки коректності всього номеру ISBN. Для перевірки ISBN обчислюється сума добутків цифр на їхній номер, нумерація при цьому починається з крайньої правої цифри. В результаті має бути отримано число, що без остачі ділиться на 11.

Наприклад: **0201103311** — коректний номер, тому що $0 \times 10 + 2 \times 9 + 0 \times 8 + 1 \times 7 + 1 \times 6 + 0 \times 5 + 3 \times 4 + 3 \times 3 + 1 \times 2 + 1 \times 1 = 55$, що націло ділиться на 11.

Кожна з перших дев'яти цифр може приймати значення від 0 до 9.

Напишіть програму, що читає ISBN код з однією пропущеною цифрою (вона буде позначатись як символ « » (пробіл)) і виводить значення пропущеної цифри.

Приклад

Вхідні дані	Результати
020110 311	3

Примітка У справжніх ISBN-номерах в якості останньої цифри може бути також велика латинська X, що позначає 10. Але в цій задачі таких номерів гарантовано не буде.

Розбір задачі В принципі, *можна* вивести аналітичну формулу, але це потребує знань з теорії чисел (бажаючи можуть знайти інформацію за ключовими словами «теорія чисел», «кільце залишків за модулем», «мала теорема Ферма» і застосувати все це до даної задачі). Але все це було *би* доцільним,

якби довжина ISBN-коду становила не 10 цифр, а, наприклад, сотні тисяч, так що розглянутий далі значно простіший і значно більш «програмістський» (а не «математичний») підхід працював би надто довго.

А для 10 цифр підходить і значно простіший алгоритм перебору, тобто пере-
пробувати усі 10 варіантів від 0 до 9 і для кожного подивитися, чи виконується
описана в умові правильність ISBN. Приклад реалізації такого підходу див.
ideone.com/g3SbMb

(Повний вміст даного посилання ideone.com/g3SbMb такий:

```
program ISBN;

{$mode delphi}

uses
  SysUtils;

var s : string;
    idx, i : Integer;
    sum : Integer;
    v : Char;

begin
  Readln(s);
  idx := Pos(' ', s);
  for v := '0' to '9' do begin
    s[idx] := v;
    sum := 0;
    for i:=1 to 10 do
      sum := sum + (11-i) * StrToInt(s[i]);
    if sum mod 11 = 0 then begin
      Writeln(v);
    end;
  end;
end.
```

— кінець цитати посилання ideone.com/g3SbMb)

Правда, якби були різні правильні відповіді, ця програма вивела б усі, хоча зазвичай на олімпіадах треба виводити будь-яку одну. Насправді це неактуально, бо дана задача не може мати різні правильні відповіді (хто вивчав питання, згадані у попередньому абзаці, можуть це легко довести, решта можуть або повірити, або дописати у розв'язок **break** для обривання роботи після виведення першої відповіді.)

Задача В. «Точні квадрати»

Вхідні дані: Клавіатура (stdin) або файл input.txt
Результати: Екран (stdout)
Обмеження часу: 1 сек
Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка знаходитиме кількість натуральних чисел із проміжку $[a; b]$, які задовольняють одночасно двом таким вимогам:

- число є точним квадратом, тобто корінь з нього цілий (наприклад, точними квадратами є $1=1^2$, $9=3^2$, $1024=32^2$; а 8, 17, 1000 не є точними квадратами).
- сума цифр цього числа кратна K . (Наприклад, сума цифр числа 16 рівна $1+6=7$.)

Програма повинна прочитати три числа в одному рядку a b K і вивести одне число — кількість чисел, які задовольняють умовам.

Оцінювання В усіх тестах виконується $1 \leq a \leq b \leq 2 \cdot 10^9$, $2 \leq K \leq 42$.

40% балів припадає на тести, в яких виконується $1 \leq a \leq b \leq 30\,000$, $K=9$.

Приклад

Вхідні дані	Результати
7 222 9	4

Примітка Цими чотирма числами є 9, 36, 81, 144.

Розбір задачі Ніби нескладна задача — перебрати, перевіряючи виконання двох умов... Але ideone.com/gLqsYd

(Повний зміст даного посилання ideone.com/gLqsYd такий:

```
var a,b,k,n,i:longint;

function SumOfDigits(g:longint):longint;
var s:longint;
begin
  s:=0;
  repeat
    s:=s+g mod 10;
    g:=g div 10;
  until g=0;
  SumOfDigits:=s;
end;

BEGIN
```

```

read(a,b,k);
for i:=a to b do
    if (frac(sqrt(i))=0) then
        if (SumOfDigits(i) mod k =0) then n:=n+1;
writeln(n);
END.

```

— кінець цитати посилання ideone.com/gLqsYd)

набирає лише 50 балів зі 100.

Проблема у тому, що перевіряти аж два мільярди чисел — все ж забагато. Навіть якщо (абсолютно слушно) рахувати суму цифр лише для тих із них, які пройшли перевірку `frac(sqrt(n))=0`. (До речі, взагалі-то надійніше перевіряти «чи є точним квадратом» як `sqr(round(sqrt(n)))=n`, адже функція `sqrt` в принципі може повернути неточне значення, як-то 4.999999999999999 замість 5, а тоді й `frac` поверне зовсім не 0. Спосіб `sqr(round(sqrt(n)))=n` не має такого недоліку. Насправді у даній задачі й на даному сервері правильно працює і перший спосіб, але в принципі тут можлива проблема.)

Тобто, головна проблема — як пришвидшити наведений розв’язок. Досить просто: можна *генерувати* відразу *лише* точні квадрати, а не перебирати геть усі числа проміжку й «у кожного питати». Достатньо запускати не цикл від a до b , а від $\approx\sqrt{a}$ до $\approx\sqrt{b}$ й працювати з i^2 . Правда, тут можна заплутатися з тим, як саме перетворити « $\approx\sqrt{a}$ » та « $\approx\sqrt{b}$ » у точні цілі значення. Не дуже красивий, зате безсумнівно правильний спосіб — узяти межі з невеличким «запасом», а потім отримане i^2 все-таки перевірити на належність проміжку $[a; b]$. Такий розв’язок, навіть із цією зайвою перевіркою, безсумнівно вкладатиметься у 1 сек з великим запасом, бо тепер кількість ітерацій $\leq \sqrt{2 \cdot 10^9} \approx 45$ тис. Див. ideone.com/Ep1T1L

(Повний вміст даного посилання ideone.com/Ep1T1L такий:

```
program SQ;

{$mode delphi}

uses
    SysUtils;
```

```
function sumOfDigits(n : integer) : Integer;
var s : string;
    k : Byte;
Begin
    s := IntToStr(n);
    Result := 0;
    for k := 1 to Length(s) do
        Result := Result + StrToInt(s[k]);
    End;

var K, A, B, i, sqr_i, res : Integer;

begin
    Readln(a, b, K);
    res := 0;
    for i:=Round(Sqrt(a)) to Round(Sqrt(b)) do begin
        sqr_i := Sqr(i);
        if (sqr_i>=a) and (Sqr_i<=b) and (sumOfDigits(sqr_i) mod K = 0) then begin
            res := res + 1;
        end;
    end;
    Writeln(res);
end.
```

— кінець цитати посилання ideone.com/Ep1T1L)

Задача С. «Ложбан»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout)
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Ложбан (англ. *Lojban*) — це штучна мова, яка була створена у 1987 році Групою Логічних Мов та базується на логлані (логічна мова). При створенні основною ціллю була повніша, вільно доступна, зручна для використання мова. Вона є експериментальною і сконструйована для перевірки гіпотези Сепіра-Ворфа. Ця гіпотеза припускає, що люди, які говорять різними мовами, по-різному сприймають світ і по-різному мислять.

Дана мова є однією з найпростіших штучних мов. Наприклад цифри від 0 до 9 записуються наступним чином:

1	pa	4	vo	7	ze	
2	re	5	mu	8	bi	0 no
3	ci	6	xa	9	so	

Великі числа утворюються склеюванням цифр разом. Наприклад, число 123 це `pareci`.

Завдання Напишіть програму, яка зчитує рядок на Ложбані (що представляє собою число $\leq 1\,000\,000$) та виводить цей рядок у цифровому вигляді.

Приклад

Вхідні дані	Результати
<code>renopavo</code>	2014

Розбір задачі По-перше, слід у великому тексті умови знайти ті $\approx 20\%$, які справді потрібні: «Є текст, який гарантовано отриманий таким чином: взяли число від 0 до 1000000, й замінили кожен цифру на дві букви згідно наведеної таблиці. Провести зворотнє перетворення цього тексту у число.».

По-друге, все могло *бу* бути вельми складним, *якби* траплялися ситуації, коли одне зі слів — початок іншого (як-то «7 позначається як `mis`, 8 — як `misiv`»). Так що треба відмовитися від ідеї писати універсальну програму, яка могла би працювати з різними позначеннями цифр, і ретельно дослідити, якими конкретними, заданими в умові, словами кодуються цифри. І побачити, що тут не лише нема такої ситуації, а ще й усі ці слова дволітерні.

Так що задача насправді досить проста. Треба лише зуміти прочитати це у громіздкій умові. Наприклад, див. ideone.com/BNuD1L

(Повний вміст даного посилання ideone.com/BNuD1L такий:

```
program lojban;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var sAll, sCurr : string;
    i : Integer;

begin
  Readln(sAll);
```

```
i := 1;
while i < Length(sAll) do begin
  sCurr := Copy(sAll, i, 2);
  if sCurr = 'no' then
    write('0')
  else if sCurr = 'pa' then
    write('1')
  else if sCurr = 're' then
    write('2')
  else if sCurr = 'ci' then
    write('3')
  else if sCurr = 'vo' then
    write('4')
  else if sCurr = 'mu' then
    write('5')
  else if sCurr = 'xa' then
    write('6')
  else if sCurr = 'ze' then
    write('7')
  else if sCurr = 'bi' then
    write('8')
  else if sCurr = 'so' then
    write('9');
  i := i+2;
end;
Writeln;
end.
```

— кінець цитати посилання ideone.com/BNuD1L)

Або, ще нахабніше використавши, що вхідні дані *гарантовано* являють собою якесь закодоване число, можна написати щось іще простіше, наприклад ideone.com/Odw21F

(Повний вміст даного посилання ideone.com/Odw21F такий:

```
program lojban;
var lojb : string;
    i : integer;
BEGIN
  readln(lojb);
  for i:=1 to length(lojb) do
  begin
    case lojb[i] of
      'p': write('1');
      'r': write('2');
      'c': write('3');
      'v': write('4');
      'm': write('5');
      'x': write('6');
      'z': write('7');
      'b': write('8');
```

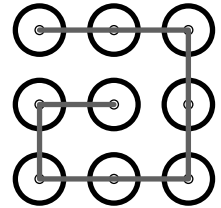
```
's': write('9');  
'n': write('0');  
end;  
end;  
END.
```

— кінець цитати посилання ideone.com/0dw21F)

Задача D. «Графічний пароль»

Вхідні дані:	Клавіатура (stdin)
Результати:	Екран (stdout)
Обмеження часу:	4 сек
Обмеження пам'яті:	64 мегабайти

Програміст Василь недавно придбав собі новий смартфон і встановив на ньому графічний пароль. Графічний пароль представляє собою ламану лінію, яка проходить через вершини сітки розміром 3×3 (не обов'язково через всі). Але згодом Василь зрозумів, що пароль на сітці 3×3 не є досить безпечним, і вирішив виправити цю проблему. Для цього він збільшив розмір сітки до 5000×5000 , але при цьому наклав обмеження на відрізки ламаної — тепер вони можуть бути лише горизонтальними та вертикальними. Василь ще хотів додати функцію, яка б виводила кількість самоперетинів у графічному паролі, але так склалося, що він не досить знайомий з ефективними алгоритмами, тому він просить вашої допомоги.



Вхідні дані Перший рядок містить ціле число N — кількість вершин в ламаній лінії ($2 \leq N \leq 1\,000\,000$). Кожен з наступних N рядків містить два цілих числа x та y — координати відповідної вершини ламаної ($0 \leq x, y < 5000$). Гарантується, що ламана у вхідних даних містить лише горизонтальні та вертикальні відрізки, які можуть перетинатися, але не можуть накладатись один на одного. Ніякі дві вершини ламаної (в т. ч. старт та фініш) не знаходяться в одній і тій самій точці.

Результати Необхідно вивести єдине ціле число — кількість самоперетинів у ламаній.

Приклади	Вхідні дані	Результати
	6 1 1 0 1 0 0 2 0 2 2 0 2	0
	5 0 1 3 1 3 2 2 2 2 0	1

Розбір задачі Головне при розв’язуванні цієї задачі — не перестаратися й не почати писати справжні «чесні» перевірки перетинів. Такі перевірки могли б бути доречними при більших розмірах сітки та меншій кількості відрізків ламаної. А при заданій кількості, відомі автору даного розбору оптимізації перебору всіх можливих перетинів не вкладаються у обмеження часу.

У даній конкретній задачі краще помітити такі факти:

1. $5000 \times 5000 = 25$ млн — багато для людини, але для комп’ютера — не так і багато. Не лише у розрізі «виконати 25 млн дій», а також і у розрізі «тримати в пам’яті 25 млн елементів».
2. Хоча один окремо взятий відрізок (ланка ламаної) може мати довжину аж 5000, а кількість відрізків може сягати мільйона, сумарна довжина відрізків насправді обмежується не $5000 \times 10^6 = 5 \cdot 10^9$, а тим, що раз відрізки лише горизонтальні й вертикальні, а накладання заборонені, то кожна з 5000×5000 вершин сітки може бути задіяна щонайбільше двічі, тобто сума довжин усіх відрізків не перевищує 50 млн.

Завдяки всьому цьому, виявляється цілком допустимим такий простий підхід, як «завести масив 5000×5000 , ініціалізувавши всі елементи нулями; ходити уздовж ліній, збільшуючи значення у комірках, відповідних пройденим вершинам сітки; щоразу, коли збільшується комірка, яка вже не 0, додавати 1 до лічильника-відповіді». Величезний time limit 4 сек насправді потрібен не для

роботи алгоритму, а для читання величезних вхідних даних. Приклад реалізації наведеного алгоритма — ideone.com/SlR8oV

(Повний зміст даного посилання ideone.com/SlR8oV такий:

{Реалізація учасника Середенка Олександра, з дрібними модифікаціями}

```
var m:array[0..4999, 0..4999] of integer;
    k,i,j,x,y,x1,y1,r,p: longint;

begin
  readln(p);
  readln(x,y);
  r:=0;
  for k:=1 to p-1 do begin
    readln(x1, y1);
    if x1>x then begin
      for i:=x to (x1-1) do m[i, y]:= m[i,y]+1;
    end;
    if x1<x then begin
      for i:=x downto (x1+1) do m[i, y]:= m[i,y]+1;
    end;
    if y1>y then begin
      for i:=y to (y1-1) do m[x,i]:=m[x,i]+1;
    end;
    if y1<y then begin
      for i:=y downto (y1+1) do m[x,i]:=m[x,i]+1;
    end;
    x:=x1;
    y:=y1;
  end;
  for i:=0 to 4999 do
    for j:=0 to 4999 do
      if m[i,j]>1 then
        r:=r+1;
  writeln(r);
end.
```

— кінець цитати посилання ideone.com/SlR8oV)

4 Обласна інтернет-олімпіада 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №18).

Задача А. «Три круги»

Вхідні дані: Клавiатура (stdin)

Результати: Екран (stdout)

Обмеження часу: 1 сек

Обмеження пам'яті: 64 мегабайти

Є три круги радіусами R_1 , R_2 та R_3 .

Чи можна перекласти їх так, щоб відразу два менші круги лежали один поруч з іншим на найбільшому, не накладаючись один на одного і не звисаючи за його межі? Торкатися один одного менші круги можуть.

Вхідні дані Програма повинна прочитати три цілі числа R_1 , R_2 та R_3 , в один рядок, через пропуски (пробіли). Усі три значення R_1 , R_2 та R_3 є цілими числами в межах від 1 до 1000.

Результати Якщо перекласти круги вказаним чином неможливо, програма повинна вивести єдине слово "NO" (без лапок).

Якщо можливо, то в єдиному рядку повинно бути записано "YES, the ... disk is the maximal" (без лапок), де замість "..." повинно бути одне з трьох значень:

- "1st" (без лапок), якщо 2-й і 3-й круги можна покласти поверх 1-го;
- "2nd" (без лапок), якщо 1-й і 3-й круги можна покласти поверх 2-го;
- "3rd" (без лапок), якщо 1-й і 2-й круги можна покласти поверх 3-го.

Зверніть увагу: фраза повинна бути однаковою з правильною байт-у-байт, тобто всі великі чи маленькі літери, всі пропуски (пробіли) та інші подібні дрібниці важливі.

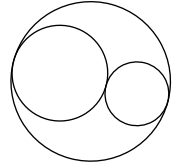
Приклади

Вхідні дані	Результати
1 2 3	YES, the 3rd disk is the maximal
2 3 4	NO
9 3 1	YES, the 1st disk is the maximal

Розбір задачі Два менші диски, яким заборонено накладатися, займають найменше місця тоді, коли торкаються. Тому гранична ситуація, коли

при хоч трохи меншому найбільшому диску вони вже не поміщаються (звисяють), а при рівно такому або більшому все гаразд, має вигляд, зображений на рисунку. Наприклад, 2-й і 3-й круги можна покласти поверх 1-го тоді й тільки тоді, коли $R_1 \geq R_2 + R_3$. Решта випадків аналогічні.

Щоб правильно виводити “NO”, легше не формулювати умову цього випадку, а зробити розгалуження вкладеними, щоб виводити “NO”, коли не виконалася жодна з трьох інших умов.



Приклад програми-розв’язку — ideone.com/rx20dn

(Повний вміст даного посилання ideone.com/rx20dn такий:

```
var
  r1,r2,r3: integer;
BEGIN
  read(r1,r2,r3);
  if r2+r3<=r1 then
    writeln('YES, the 1st disk is the maximal')
  else
    if r1+r3<=r2 then
      writeln ('YES, the 2nd disk is the maximal')
    else
      if r1+r2<=r3 then
        writeln('YES, the 3rd disk is the maximal')
      else
        writeln ('NO')
END.
```

— кінець цитати посилання ideone.com/rx20dn)

Можливий інший підхід — спочатку знайти, який з дисків максимальний, а вже потім провести одне порівняння. Для даної задачі це погана ідея: і тому, що треба виводити номер максимального диску (ще й у вигляді “1st”/“2nd”/“3rd”), і тому, що з’ясування, який диск максимальний, займе чи не більше дій, ніж розгляд випадків у попередньому розв’язку.

Але для багатьох інших задач (включно з деякими задачами цього ж збірника) деякий аналог другого підходу виявляється набагато кращим за аналіз випадків, аналогічний першому підходу.

Задача В. «Перевезення вантажу»

Вхідні дані: Клавiатура (stdin)

Результати: Екран (stdout)

Обмеження часу: 1 сек

Обмеження пам'яті: 64 мегабайти

Перевізник транспортує вантаж залізницею, а потім річковим транспортом.

Товар транспортується в потязі, який складається з N ($1 \leq N \leq 100$) вагонів в кожному з яких $k_1 \dots k_N$ ($1 \leq k_i \leq 1000$) коробок вантажу.

У річковому порті вантаж з потягу перевантажують на корабель, який може взяти не більше ніж P ($1 \leq P \leq 10000$) коробок. Якщо якісь коробки не помістяться на цей корабель, вони дуже довго чекатимуть наступного. Причому чекатимуть обов'язково у тих вагонах, в яких прибули у порт.

Для ефективного транспортування диспетчеру необхідно розвантажити якомога більше вагонів, завантаживши корабель, та відправити звільнені вагони на наступне завантаження.

Завдання Напишіть програму `transport`, яка визначала б максимальну кількість вагонів, які можна розвантажити, не перевищуючи вантажопідйомність корабля.

Вхідні дані 1-й рядок: єдине число N — кількість вагонів в потязі.

2-й рядок: $k_1 \dots k_N$ через пропуски (пробіли) — кількості коробок у вагонах.

3-й рядок: P — кількість коробок, яку може взяти на борт корабель.

Результати Максимальна кількість вагонів потягу, які вдасться розвантажити.

Приклад

Вхідні дані	Результати
3 5 7 3 9	2

Розбір задачі Ця задача, хоч і нескладна, потребує і з'ясувати правило, за яким слід вибирати вагони, і написати не зовсім елементарну програму.

Тож правило — *«Щоразу вибирати (ще не вибраний) вагон з мінімальною кількістю коробок, доки не вичерпається вантажопідйомність корабля або не будуть задіяні всі вагони»*.

Доведемо, що це дійсно дасть максимальну кількість вагонів. (Чи займатися взагалі такими доведеннями — залежить від багатьох обставин. З одного боку, вони не вимагаються та не оцінюються. З іншого — не вміючи доводити, важко оцінювати правильність ідей, а написання програми на основі неправильної ідеї може забрати час і принести дуже мало балів.)

Довести можна за такою (типовою, тобто її аналоги придатні у багатьох ситуаціях) схемою: «Припустимо, ніби замість вагона з мінімальною кількістю коробок взяли вагон з деякою більшою кількістю. Це збільшило кількість розвантажених вагонів так само на 1, якби взяли вагон з мінімальною кількістю, а залишок вантажопідйомності корабля зменшило сильніше. Отже, ніякого виграшу від того, щоб брати вагон з немінімальною кількістю коробок, нема».

Сформульовані правила — ще не зовсім алгоритм. Є мінімум два способи подальшого їх уточнення (як саме «вибирати (ще не вибраний) вагон з мінімальною кількістю коробок»?).

Можна застосувати сортування (воно ж упорядкування), після нього це будуть просто елементи масива по порядку. Особливо зручно, якщо писати мовою програмування, де існує готове бібліотечне сортування (наприклад, у реалізації ideone.com/sElRrj

(Повний вміст даного посилання ideone.com/sElRrj такий:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main(int argc, char* argv[])
{
    int N;
    cin >> N;
    vector<int> wagons(N);
    for(int i=0; i<N; i++)
        cin >> wagons[i];
```

```
sort(wagons.begin(), wagons.end());
int P;
cin >> P;
int res = 0;
while(res < N && wagons[res] <= P) {
    P -= wagons[res];
    res++;
}
cout << res << endl;
return 0;
}
```

— кінець цитати посилання ideone.com/sElRrj)

використано функцію `sort` бібліотеки `algorithm` мови C++). Та й якщо писати сортування самому, все одно є деяка зручність у тому, щоб окремо написати та ретельно вивірити правильність стандартної задачі сортування, окремо решту дій.

Можна багатократно проходити по усьому масиву, вибираючи мінімальний елемент, і після кожного такого проходу враховувати знайдений елемент та замінити його на якесь велике значення, щоб не знаходити повторно. Втім, особливих переваг цей спосіб не має. Зокрема, заявка «він швидший, бо не сортує усі елементи, а вибирає лише стільки, скільки треба» не витримує критики, бо за умови ефективного алгоритму сортування перший спосіб у середньому значно швидший.

Задача С. «Коло і точки»

Вхідні дані:	Клавіатура (stdin)
Результати:	Екран (stdout)
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Є одне коло та N точок.

Завдання Напишіть програму, яка знаходитиме, скільки з цих N точок потрапили всередину цього кола, скільки на самé коло і скільки ззовні кола.

Вхідні дані Перші два рядки вхідних даних задають коло. Можуть бути два різні формати його задання:

1. Якщо 1-ий рядок містить єдине число 1, то 2-ий рядок містить рівно три цілі числа x_C , y_C , R_C — координати центра кола та його радіус.
2. Якщо 1-ий рядок містить єдине число 2, то 2-ий рядок містить рівно шість цілих чисел x_A , y_A , x_B , y_B , x_C , y_C . Їх слід трактувати як координати трьох точок A , B , C , через які проведене коло, котре треба дослідити. Ці три точки гарантовано всі різні і гарантовано не лежать на одній прямій. Гарантовано також, що ніяка з цих трьох точок не збігається ні з одною з точок подальшого переліку з N точок.

Третій рядок вхідних даних завжди містить єдине ціле число N — кількість точок. Подальші N рядків містять по два цілі числа кожен — x та y координати самих точок.

Абсолютно всі задані у вхідних даних числа є цілими і не перевищують за абсолютною величиною (модулем) 1000. При цьому кількість точок і радіус гарантовано додатні, а координати можуть бути довільного знаку.

Скрізь, де в одному й тому ж рядку записано по кілька чисел, вони відділені одне від одного пропусками (пробілами).

Результати Програма повинна вивести три цілі числа — спочатку кількість точок всередині кола, потім кількість точок на самому колі, потім кількість точок ззовні кола.

Сума цих трьох чисел повинна дорівнювати N . Зокрема, якщо коло задане 2-им способом, то ті три точки, якими воно задане, треба не вважати точками, належними колу.

Приклади	Вхідні дані	Результати
	1 -1 2 5 4 0 0 -3 -3 -6 2 6 -2	1 1 2
	2 -1 -3 2 6 3 5 4 0 0 -3 -3 -6 2 6 -2	1 1 2

Примітки В обох наведених прикладах задане (різними способами) одне й те саме коло, тому що коло, проведене через точки $(-1; -3)$, $(2; 6)$ та $(3; 5)$ якраз і має радіус 5 та центр у точці $(-1; 2)$.

Перша одиничка відповіді виражає, що лише одна точка з переліку потрапила всередину кола (і це точка $(0; 0)$, але цього не питають). Друга одиничка відповіді виражає, що лише одна точка з переліку потрапила на саме коло (і це точка $(-6; 2)$, але цього не питають). Число 2 у відповіді позначає, що решта дві точки переліку (це точки $(-3; -3)$ та $(6; -2)$, але цього не питають) потрапили ззовні кола.

Можна здавати програму, яка враховує лише подання кола через координати центра і радіус. На тести, в яких коло подається 1-им способом, припадатиме майже половина балів, і така програма може їх отримати. Але навіть така програма повинна враховувати, що в цих тестах все одно буде 1-ий рядок з єдиним числом 1.

Разом з тим, якщо враховувати обидва випадки, це все одно повинна бути одна програма.

Розбір задачі Оскільки коло — множина точок, рівновіддалених від центру, то досліджувана точка потрапляє всередину кола, коли відстань між нею і центром кола менша за його радіус, на саме коло — коли рівна радіусу, і назовні, коли більша. Самá ця відстань може бути обчислена

як $\sqrt{(x_i - x_C)^2 + (y_i - y_C)^2}$ (де $(x_i; y_i)$ — координати досліджуваної точки, $(x_C; y_C)$ — центру кола). Тож (для програми, що працює *лише для 1-го* способу подання кола) лишається тільки написати цикл з розгалуженнями (вкладеними, щоб розібрати три випадки); наприклад, див. ideone.com/k9iLan (Повний вміст даного посилання ideone.com/k9iLan такий:

```
{ $mode delphi }

var mode : integer;
    Cx, Cy, CR : integer;
    i, N : integer;
    x, y : array[1..1000] of integer;
    num_inside, num_at, num_outside : integer;

BEGIN
    readln(mode);
    if mode = 1 then
        readln(Cx, Cy, CR)
    else begin
        Cx:=-1; Cy:=2; CR:=2;
        readln;
    end;
    readln(N);
    for i:=1 to N do
        readln(x[i], y[i]);
    num_inside := 0;
    num_at := 0;
    num_outside :=0;
    for i:=1 to N do
        if sqr(x[i] - Cx) + sqr(y[i] - Cy) = sqr(CR) then
            inc(num_at)
        else if sqr(x[i] - Cx) + sqr(y[i] - Cy) < sqr(CR) then
            inc(num_inside)
        else {sqr(x[i] - Cx) + sqr(y[i] - Cy) > sqr(CR)}
            inc(num_outside);
    writeln(num_inside, ' ', num_at, ' ', num_outside);
END.
```

— кінець цитати посилання ideone.com/k9iLan)

Розібратися, що робити з 2-им способом подання кола — *набагато* складніше, можливо навіть складніше усієї решти («2^{1/2} задач») даного туру. Усі відомі автору задачі способи так чи інакше зводять 2-ий випадок до 1-го, тобто спочатку переходять від трьох точок до центру й радіусу кола, проведеного через ці три точки, а потім використовують вже розглянутий розв'язок.

Один зі способів — побудувати описане коло $\triangle ABC$ через побудову перетину серединних перпендикулярів сторін, тобто ті ж дії, що й на уроці геометрії, але виконані замість циркуля і лінійки інструментами *обчислювальної геометрії* (рос. «вычислительная геометрия», англ. «*computational geometry*»). Вступ до обч. геометрії можна знайти у багатьох місцях, зокрема <https://goo.gl/6yppjy> Програму, що розв'язує цим способом, можна бачити за посиланням ideone.com/k1aFcF

(Повний вміст даного посилання ideone.com/k1aFcF такий:

```
#include <iostream>
#include <cmath>

using namespace std;

double SQR(double x) { return x*x; }

struct Point
{
    double x, y;
};

// many computational geometry functions were here, but they are intentionally deleted

pair<double, double> solve_system_2x2(const Point &p1, const Point &p2, const Point &right_part)
{
    double det_main = vect_mul(p1, p2);
    double det_1 = vect_mul(right_part, p2);
    double det_2 = vect_mul(p1, right_part);
    return make_pair(det_1/det_main, det_2/det_main);
}

void calculate_circle_by_3_points(Point &C, double &R)
{
    Point A1, A2, A3;
    cin >> A1.x >> A1.y >> A2.x >> A2.y >> A3.x >> A3.y;
    Point A1A2mid = 0.5 * (A1+A2);
    Point A1A2perp = rot_plus_pi_2(A2 - A1);
    Point A1A3mid = 0.5 * (A1+A3);
    Point A1A3perp = rot_plus_pi_2(A3 - A1);
    double k1 = solve_system_2x2(A1A2perp, -1.0 * A1A3perp, A1A3mid - A1A2mid).first;
    C = A1A2mid + k1*A1A2perp;
    R = dist(A1, C);
}

int main(int argc, char* argv[])
{
    int mode;
    Point C;
```

```
double R;
cin >> mode;
if(mode==1)
{
    cin >> C.x >> C.y >> R;
}
else
{
    calculate_circle_by_3_points(C, R);
}
int N;
cin >> N;
int num_inside = 0, num_at = 0, num_outside = 0;
for(int i=0; i<N; i++)
{
    Point A;
    cin >> A.x >> A.y;
    double d;
    if(fabs((d = dist(C,A)) - R) < 1e-4)
        num_at++;
    else if(d < R)
        num_inside++;
    else
        num_outside++;
}
cout << num_inside << " " << num_at << " " << num_outside << endl;
return 0;
}
```

— кінець цитати посилання ideone.com/k1aFcF)

(але звідти свідомо прибрані допоміжні функції, пояснені за попереднім посиланням).

Можна й вивести (на папері) прямі формули, наприклад розв'язавши систему
$$\begin{cases} (x-x_A)^2 + (y-y_A)^2 = (x-x_B)^2 + (y-y_B)^2, \\ (x-x_A)^2 + (y-y_A)^2 = (x-x_C)^2 + (y-y_C)^2, \end{cases}$$
 де $x_A, y_A, x_B, y_B, x_C, y_C$ — координати заданих точок (тому до них треба ставитися як до відомих значень, а не як до змінних), а x та y — координати центра кола, от їх-то і шукаємо. Розв'язати цю систему легше, ніж може здатися, бо після перетворення $(x-x_A)^2 = x^2 - 2x_A \cdot x + x_A^2$ та решти аналогічних можна позводити x^2 та y^2 , і система виявляється лінійною відносно x та y .

В принципі можливі й інші підходи.

5 II (районний/міський) етап 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №48).

Задача А. «Цифра»

Вхідні дані: Клавiатура (stdin) або файл input.txt
Результати: Екран (stdout) або файл output.txt
Обмеження часу: 1 сек
Обмеження пам'яті: 64 мегабайти

В заданому додатному числі потрібно закреслити одну цифру так, щоб число, яке залишиться в результаті, було найбільшим.

Напишіть програму, яка читає одне ціле значення n ($10 \leq n \leq 99999$), і виводить число без однієї цифри (це число має бути найбільшим серед усіх можливих варіантів закреслень цифри).

Приклади	Вхідні дані	Результати
	321	32
	129	29

Розбір задачі Діяти за принципом «Викреслювати мінімальну цифру» — *неправильно* (всупереч підступним прикладам з умови, які провокують таку хибну думку.) Наприклад, із числа 9891 треба викреслити 8 і отримати 991, а викреслення мінімальної цифри 1 дасть не максимальне 989.

Оскільки мова йде про досить малі обмеження (кількість цифр ≤ 5 , видаляється лише одна), найпростіший для написання правильний розв'язок — явно перебрати всі варіанти викреслювання однієї цифри (усе число без 1-ої, усе число без 2-ої, тощо), і вибрати з них максимальний. При цьому зручно (не обов'язково, але зручно) перевести прочитане число у рядок (`string`), і займатися вилученням цифр у `string`-овому поданні. Реалізацію див. ideone.com/jFBIM6

(Повний вміст даного посилання ideone.com/jFBIM6 такий:

```
{ $mode delphi }
```



```
uses SysUtils;  
  
var s_orig, s_deleted : string;  
    i : Integer;  
    n, curr_value, max_found : Integer;  
  
BEGIN  
    max_found := -1;  
    Readln(n);  
    s_orig := IntToStr(n);  
    for i := 1 to Length(s_orig) do begin  
        s_deleted := s_orig;  
        Delete(s_deleted, i, 1);  
        curr_value := StrToInt(s_deleted);  
        if curr_value > max_found then  
            max_found := curr_value;  
        end;  
        Writeln(max_found);  
    END.
```

— кінець цитати посилання ideone.com/jFBIM6)

Правильним є також і розв'язок «Знайти найлівіше місце, де зразу після меншої цифри йде більша, і викреслити меншу саме з цих двох; якщо жодного такого місця нема (наприклад, у числі 97652) — викреслити останню цифру».

Задача В. «Піраміда»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Гіллу Бейтсу захотілося увічнити пам'ять про свою корпорацію MegaHard. Він обрав перевірений часом метод — побудувати піраміду.

Піраміда має висоту n Стандартних Будівельних Блоків, і кожен її рівень — квадрат $k \times k$ блоків, де k — номер рівня, рахуючи згори. З'ясувалося, що фірма, що виготовляє Стандартні Будівельні Блоки, продає їх лише партіями по m штук.

Потрібно написати програму, що визначатиме, скільки блоків залишиться не використаними після побудови піраміди (Гілл Бейтс забезпечить закупівлю

мінімально необхідної для побудови піраміди кількості партій).

Напишіть програму, яка читає в один рядок через пропуск (пробіл) спочатку кількість бажаних рівнів піраміди n ($1 \leq n \leq 10^9$, тобто мільярд), потім розмір партії Стандартних Будівельних Блоків m ($1 \leq m \leq 10^6$, тобто мільйон), і виводить єдине ціле число — кількість Блоків, що залишаться не використаними, якщо купити найменшу можливу кількість цілих партій.

Приклад

Вхідні дані	Результати
7 16	4

Примітка Піраміда з 7 рівнів міститиме $1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 = 140$ блоків, і якщо купити 8 партій по 16 блоків, то цих 128 блоків не вистачить; тому треба купити 9 партій по 16 блоків, тоді з цих 144 блоків 4 залишаться зайвими.

Розбір задачі Ця задача проста, щоб набрати *частину* балів, але набрати повний бал чи значну частину балів не так просто. Розв'язок ideone.com/w8Yu4L

(Повний вміст даного посилання ideone.com/w8Yu4L такий:

```
var i, n, m : longint;
    res : int64;
BEGIN
    readln(n,m);
    res:=0;
    for i:=1 to n do
        res := res + sqr(int64(i));
    res:=res mod m;
    if res<>0 then
        res := m - res;
    writeln(res);
END.
```

— кінець цитати посилання ideone.com/w8Yu4L)

набирає половину балів. І тут є де помилітися й отримати ще менше (важливо і правильно врахувати смисл m , і взяти тип `int64`).

При n , починаючи з $\approx 3 \cdot 10^6$, сума $1^2 + 2^2 + \dots + n^2$ виходить навіть за межі типу `int64` (він же `long long`). З цим можна боротися, застосовуючи найпростіший прийом т. зв. *модульної арифметики*, а саме, роблячи додавання

чергового i^2 не як $s:=s+\text{sqr}(\text{int64}(i))$, а як $s:=(s+\text{sqr}(\text{int64}(i)))\bmod m$.

Реалізація, де вчасно робиться “... mod m”, набирає 150 балів (з 250). Тепер критичним стає те, що $\approx 10^9$ ітерацій циклу (ще й з громіздкою операцією mod) не поміщаються в обмеження часу (1 сек).

100%-ий розв’язок № 1 Якщо знати (або вивести під час туру) формулу $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$, можна поєднати її з засобами модульної арифметики і отримати зовсім інший розв’язок <http://ideone.com/g6nSKa>. Він взагалі не містить циклів, тож вкладається у 1 сек з величезним запасом.

Щоб написати цей розв’язок, треба знати деякі математичні факти та властивості, і це може не всім подобатися. Але, по-перше, задача має альтернативний розв’язок; по-друге, навіть якби цей розв’язок був єдиним, це не суперечило б традиціям Всеукраїнської олімпіади з інформатики.

У цьому розв’язку треба писати саме “... mod (6*m)” (а не “... mod m”), бо такі властивості модульної арифметики: хоча $(a + b) \bmod p = ((a \bmod p) + (b \bmod p)) \bmod p$ — правильна тотожність, що виконується для всіх a, b, p , і так само для множення, але для ділення виявляється не так. Так що кого зацікавило — просимо знайти детальніші відомості про модульну арифметику в Інтернеті або літературі.

Щодо того, як можна вивести формулу $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$ самому під час туру — див., наприклад, <http://dxdy.ru/topic22151.html>. Звісно, доцільність витрачання часу туру на подібні виведення формул істотно залежить від умінь конкретного учасника, та від того, чи має місце ситуація, коли учасник знає, що така формула в принципі існує, але не пам’ятає її точно.

100%-ий спосіб № 2 Навіть не знаючи ні формули зі способу № 1, ні модульної арифметики, можна побудувати (інший) повнобальний розв’язок, користуючись лише базовими, в межах загальнообов’язкового мінімуму, знаннями математики, а також спостережливістю та кмітливістю.

Якщо $n < 2 \cdot 10^6$, можна порахувати відповідь «в лоб», як на початку розбору. Інакше (при $n \geq 2 \cdot 10^6$, враховуючи, що $m \leq 10^6$) вийде, що проміжок від 1

до n складається з кількох (як мінімум, двох, як максимум — багатьох сотень мільйонів) проміжків від 1 до m , від $m+1$ до $2m$, від $2m+1$ до $3m$, і т. д.

Розглянемо (праворуч) послідовність очевидних тотожностей для проміжків від 1 до m та від $m+1$ до $2m$. У виразі m^2+2m+1 частина m^2+2m кратна m і тому не впливає на остаточною відповідь задачі.

$$\begin{array}{l|l} 1^2=1 & (m+1)^2 = m^2 + 2m + 1 \\ 2^2=4 & (m+2)^2 = m^2 + 4m + 4 \\ 3^2=9 & (m+3)^2 = m^2 + 6m + 9 \\ \vdots & \vdots \\ m^2=m^2 & (m+m)^2 = m^2 + 2m^2 + m^2 \end{array}$$

Аналогічно не впливають на відповідь частини m^2+4m , m^2+6m , і т. д. Тобто, $(m+1)^2 \bmod m = 1^2 \bmod m$, $(m+2)^2 \bmod m = 2^2 \bmod m$, $(m+3)^2 \bmod m = 3^2 \bmod m$, ..., а звідси — сума усього другого проміжку $(m+1)^2 + (m+2)^2 + \dots + (m+m)^2$ має той самий залишок від ділення на m , що й сума усього першого $1^2 + 2^2 + \dots + m^2$. З аналогічних причин, такий самий залишок мають і сума усього третього проміжку $(2m+1)^2 + (2m+2)^2 + \dots + (2m+m)^2$, і сума будь-якого подальшого. Тому досить цей однаковий для всіх проміжків залишок домножити на $n \operatorname{div} m$, а потім, якщо n не кратне m , окремо порахувати й додати шматочок від $(n \operatorname{div} m) \cdot m + 1$ до n (або від 1 до $n \bmod m$). Таким чином, сумарна кількість ітерацій усіх циклів не перевищить $m + (n \bmod m) < 2m \leq 2 \cdot 10^6$. Це довше, ніж у «способі № 1», але теж вкладається у 1 сек. Щоправда, якби обмеження було не $m \leq 10^6$, а $m \leq 10^9$, цей розв'язок став би неможливим (а «спосіб № 1» лишився б можливим).

Задача С. «Ко-анаграмічно-прости»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

Число називається *простим*, якщо воно має рівно два різні дільники — себе і одиницю. Наприклад: 23 — просте число, а 35 не просте, бо $5 \times 7 = 35$. Число 1

теж не просте (лише один діленьник).

Якщо у числі змінити порядок цифр, властивість простоти може змінитися: наприклад, 35 — не просте число, а 53 — просте.

Будемо називати число *ко-анаграмічно-простим*, коли при хоча б одному можливому порядку цифр утворюється просте число. Наприклад, 35 є ко-анаграмічно-простим (бо 53 просте), а 225 — не є, бо жодне з чисел 225, 252 та 522 не є простим.

Напишіть програму, яка читає одне ціле додатне значення n ($10 \leq n \leq 9999$) і виводить у першому рядку мінімальне можливе ко-анаграмічно-просте число, більше-рівне за n , а у другому рядку — ту перестановку цифр, яка робить його простим. Якщо при перестановці з'являються нулі спереду числа — слід вважати, що вони не впливають на значення числа (05 дорівнює 5), але виводити слід обов'язково із цими нулями (якщо правильно 05, то вивести 5 неправильно). Разом з тим, перше число відповіді починати з нуля не можна.

Якщо можливі різні правильні відповіді — виводьте будь-яку одну з них.

Приклади

Вхідні дані	Результати
35	35 53
49	50 05
225	227 227

Розбір задачі У цій задачі треба акуратно писати відносно велику, як для II етапу, програму, поєднуючи застосування різних стандартних алгоритмів. Але, не зважаючи на страшну назву «ко-анаграмічно» та інші громіздкості, тут не дуже-то й треба самому придумувати щось математичне. Якщо, звісно:

- знати стандартний алгоритм перевірки простоти числа;
- вміти *або перевіряти, чи* числа складаються з одних і тих самих цифр, *або генерувати перестановки* послідовності цифр.

Стандартний алгоритм перевірки простоти числа n (при $n \geq 2$) — пробувати

ділити його на 2, 3, ..., $\text{round}(\sqrt{n})$, і якщо хоча б раз поділилося без остачі — число складене (не просте), якщо не поділилося ні разу — просте. Важливо перевіряти до кореня (а не до n чи $n/2$), бо це набагато менше (отже, швидше). Перевіряти далі кореня нема потреби, бо якщо n кратне деякому a , то кратне також і n/a , а серед чисел a та n/a хоча б одне $\leq \sqrt{n}$.

Перевіряти, чи числа складаються з одних і тих самих цифр у різному порядку, можна по-різному. Один з простих і зручних підходів — відсортувати (наприклад, за неспаданням) цифри окремо одного з них, окремо іншого, й порівняти отримані відсортовані послідовності (вони рівні тоді й тільки тоді, коли числа складаються з одних і тих самих цифр). Оскільки кількість цифр дуже маленька, нема смислу використовувати quickSort чи подібні ефективні сортування, доречніші вставки або навіть звичайнісінька бульбашка.

Таким чином, алгоритм може бути приблизно таким. Функція (підпрограма) «перевірити, чи є число ко-анаграмічно-простим» має вигляд:

1. створимо копію цього числа у рядковому (**string**-овому) вигляді;
2. відсортуємо цифри (символи рядка) за неспаданням;
3. переберемо усі числа з відповідною кількістю цифр, від 0...01 до 9...99, і для кожного з них:
 - (а) перетворимо у рядок (знову як копію, щоб не псувати оригінал);
 - (б) теж відсортуємо цифри числа за неспаданням;
 - (с) якщо відсортовані послідовності виявилися різними — значить, поточне число не є перестановкою цифр досліджуваного і його слід пропустити, а якщо виявилися однаковими — запустити перевірку поточного числа на простоту.

Якщо перевірка у п. 3с хоча б один раз виявила, що число просте — функція в цілому має повернути результат «число є ко-анаграмічно-простим». Якщо жодного разу не виявила — результат «не є».

Тепер у основній програмі лишається тільки перевіряти, чи є ко-анаграмічно-

простим самé введене число n , потім $n+1$, $n+2$, ... Приклад такої реалізації — ideone.com/mGI9hm

(Повний вміст даного посилання ideone.com/mGI9hm такий:

```
program with_sort_;

{$mode delphi}

uses SysUtils;

const max_of_length:array[1..5] of Integer = (9, 99, 999, 9999, 99999);

function is_prime(n : integer):boolean;
var sqrt_n, p : Integer;
Begin
    if n < 2 then begin
        is_prime := false; exit
    end;
    for p:=2 to round(sqrt(n)) do
        if n mod p = 0 then begin
            is_prime := false; exit
        end;
    is_prime := true;
End;

function IntToStrLeadZeros(num_to_convert : integer; num_digits : integer) : string;
Begin
    Result := IntToStr(num_to_convert);
    if Length(Result) > num_digits then
        Writeln('Bad parameters IntToStrLeadZeros: ', num_to_convert, ', ', num_digits)
    else
        while Length(Result) < num_digits do
            Insert('0', Result, 1);
End;

function sort_chars_in_string(s : string) : string;
var i, k : Integer;
    t : Char;
Begin
    Result := s;
    for i:=Length(Result)-1 downto 1 do
        for k:=1 to i do
            if Result[k] > Result[k+1] then begin
                t:=Result[k]; Result[k]:=Result[k+1]; Result[k+1]:=t;
            end
        end
    end;
End;

function is_co_anagrammic_prime(n : integer; var proof : string) : Boolean;
var n_as_str, n_as_sorted_str : string;
    p, len : Integer;

Begin
```

```
n_as_str := IntToStr(n);
len := Length(n_as_str);
n_as_sorted_str := sort_chars_in_string(n_as_str);
for p:=2 to max_of_length[len] do
    if (n_as_sorted_str = sort_chars_in_string(IntToStrLeadZeros(p, len)))
        and is_prime(p) then begin
        is_co_annagramic_prime := true;
        proof := IntToStrWithLeadingZeroes(p, len);
        Exit;
    end;
is_co_annagramic_prime := False; // correct only due to length(s) is
                                // always in range <= length of max_of_length array
End;

var n : Integer;
    m : Integer;
    proof : string;

BEGIN
    Readln(n);
    m:=n;
    while not is_co_annagramic_prime(m, proof) do
        Inc(m);
        writeln(m);
        writeln(proof);
    END.
```

— кінець цитати посилання ideone.com/mGI9hm)

Програма може працювати помітно швидше, якщо замість перевірки усіх чисел з відповідною кількістю цифр (від 0...01 до 9...99) і перевірки кожного з них, чи складається воно з тих самих цифр, відразу генерувати лише потрібні числа (що складаються з тих самих цифр) і перевіряти на простоту лише їх. При обмеженні $n \leq 9999$ це насправді неважливо, тож утримаємося від детального опису, як робити це вручну (бажаючі можуть пошукати за ключовими словами «генерація перестановок», «generate permutations»). При використанні мови C++, можна скористатися готовою бібліотечною функцією перебору перестановок (функція `next_permutation` з бібліотеки `algorithm`); деталі щодо її використання знов-таки пропонується знайти самостійно.

Задача D. «Хмарочоси»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	2 сек
Обмеження пам'яті:	64 мегабайти

Як відомо, місто Прямий Ріг являє собою одну пряму вулицю, уздовж якої прокладена координатна вісь Ox . Останнім часом у місті розгорнувся будівельний бум, внаслідок якого ПрямМіськБуд звів N хмарочосів. Кожен хмарочос можна охарактеризувати висотою h_i та координатою x_i (усі x_i різні, усі h_i строго додатні).

За задумами мерії Прямого Рогу, настав час обладнати на дахах деяких із хмарочосів оглядові майданчики. Прибутковість такого майданчику залежить від того, скільки з даху даного хмарочосу видно інших хмарочосів. Тому Вас попросили написати програму, яка підготує відповідну статистику. Дані про хмарочоси зберігаються у мерії в порядку зростання x_i , тож саме в такому порядку вони і вводитимуться у Вашу програму. Якщо дахи трьох або більше хмарочосів виявляються розміщеними на одній прямій, ближчі затуляють дальші, і дальших не видно.

Вхідні дані Перший рядок містить кількість хмарочосів N , наступні N рядків містять по два цілі числа кожен — координату x_i та висоту h_i . Гарантовано, що $1 \leq N \leq 4321$, $0 \leq x_1 < x_2 < \dots < x_N \leq 10^6$ (мільйон), $1 \leq h_i \leq 10^5$ (сто тисяч). Програма може читати вхідні дані хоч з клавіатури, хоч зі вхідного файлу `input.txt` (але лише з чогось одного, а не поперемінно).

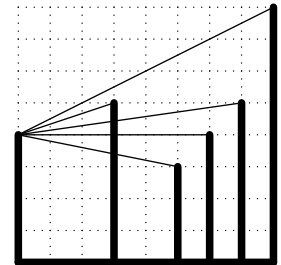
Результати Програма має вивести N рядків, i -ий рядок має містити одне ціле число — кількість інших хмарочосів, які видно з даху хмарочосу № i . Програма може виводити результати хоч на екран, хоч у вихідний текстовий файл `output.txt` (але лише на/у щось одне, а не поперемінно то туди то туди).

Приклад

Вхідні дані	Результати	
11	2	
0 4	5	
3 5	3	
5 3	4	
6 4	3	
7 5	10	
8 8	3	
10 5	4	
12 4	4	
14 3	3	
17 1	5	
19 7		

Розбір задачі

Введемо поняття *нахил з j -го хмарочосу на k -й* (на рисунку зображені нахили з 1-го хмарочосу на інші). Кожен нахил можна подавати у програмі або як кутовий коефіцієнт $\frac{y_k - y_j}{x_k - x_j}$, або як вектор $(x_k - x_j; y_k - y_j)$. (Про переваги використання таких векторів у багатьох геометричних задачах



див. <https://goo.gl/6yppjy> зокрема і за ключовими словами «обчислювальна геометрія» взагалі; але у цій задачі достатньо й кутових коефіцієнтів, треба лише подавати їх у якнайточнішому типі **extended** (він же **long double**).)

Розглянемо, як порахувати кількість хмарочосів, видимих з 1-го. 2-й хмарочос (якщо він взагалі існує, тобто $N > 1$) гарантовано видимий з 1-го. Запам'ятаємо нахил з 1-го хмарочосу на 2-й і оголосимо його *поточним нахилом затулення горизонту*. Потім бігтимемо по решті хмарочосів зліва направо, і нахил зі все того ж 1-го на кожен наступний виявлятиметься або нижчим-або-рівним, чим нахил поточного затулення горизонту (за годинниковою стрілкою), або вищим (проти годинникової стрілки). У першому випадку, поточного хмарочосу не видно й нічого робити не треба. У другому — треба додати одиничку на позначення того, що поточний хмарочос видно з 1-го, і оновити нахил поточного затулення горизонту.

Для інших хмарочосів, є ще хмарочоси зліва, які теж треба врахувати. Ускладнення не принципове, але *як краще* це врахувати?

Писати ще один аналогічний цикл, який біжить від поточної позиції не направо, а наліво — спосіб можливий, але не найкращий. Зокрема, тому, що саме в таких дублюваннях схожих фрагментів часто з'являються помилки. Особливо, якщо з першого разу написати неправильно, і потім в одному з фрагментів виправити, а в іншому забути або змінити неправильно.

Досить зручний спосіб, який і дозволяє не писати зайвого, і працює дещо швидше за інші — врахувати симетричність видимості (k -й видно з j -го тоді й тільки тоді, коли j -й видно з k -го). Завдяки цьому, коли при розгляді сусідів j -го хмарочоса з'ясовано, що з нього видно k -й ($k > j$), можна додати по одиниці і до кількості видимих з j -го, і до кількості видимих з k -го.

Втім, дане пришвидшення (удвічі) не принципове. А от намагання розв'язати задачу, взагалі не звертаючись до поняття нахилу затулення горизонту, а через три вкладені цикли (два перебирають, яку пару хмарочосів досліджуємо, і всередині них третій перебирає усі проміжні між ними, щоб подивитися, чи не затуляє) — погана ідея. Такий розв'язок набиратиме 150–190 балів з 250, а обробка решти тестів не поміщається у обмеження часу 2 сек. Конкретне значення у проміжку 150–190 залежить головним чином від того, чи обривається третій цикл негайно, як тільки можна встановити результат.

6 III (обласний) етап 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №48).

Задача А. «Гірлянда — garland»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

На Новорічні свята Василь разом з батьком вирішили зробити електричну

гірлянди із лампочок. При створенні гірлянди лампочки включаються в коло послідовно та деякі паралельно так, що в паралельному з'єднанні може знаходитись лише дві лампочки. Для з'ясування можливості підключення гірлянди до джерела живлення, Василю треба визначити загальний опір створеної гірлянди. З фізики йому відомо, що при послідовному з'єднанні загальний опір кола розраховується за формулою: $R_1 + R_2 + \dots + R_n$, а при паралельному: $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$.

Завдання Напишіть програму `garland`, яка б обчислювала загальний опір гірлянди.

Вхідні дані У першому рядку задано загальну кількість лампочок N , в другому — через пробіл опір кожної з лампочок (значення опору є цілим числом і знаходиться в межах від 1 до 1000). В третьому рядку знаходиться ціле число K — кількість пар лампочок ($0 \leq K \leq N/2$), які включені в коло паралельно. Кожен наступний рядок містить по два порядкових номери лампочок, що включені паралельно.

Результати Загальний опір гірлянди (як дійсне число без заокруглень).

Приклад

Вхідні дані	Результати	Схема прикладу
6 2 4 6 7 3 1 2 2 3 5 6	12.15	

Примітка Можна виводити також і у форматі `1.2150000000000E+01`.

Розбір задачі Задача в основному на реалізацію, тобто головне — уважно прочитати й акуратно реалізувати. Єдиний неочевидний момент — як рахувати суму всіх *тих опорів*, які не були задіяні у паралельних підключеннях. Один з можливих способів такий: спочатку порахувати суму абсолютно всіх опорів; читаючи чергову пару номерів опорів, з'єднаних паралельно, не лише обчислювати опір їхнього паралельного з'єднання $1/(1/R[i] + 1/R[j])$ і додавати його до результату, але також і віднімати з того ж результату $R[i] + R[j]$,

бо ці бори раніше додали, а виявилось, що даремно.

Ще один можливий спосіб — почати з розгляду паралельних пар, і щоразу, додавши до загальної суми $1/(1/R[i] + 1/R[j])$, *замінити* $R[i]$ та $R[j]$ на нулі. Завершивши розгляд усіх паралельних пар, подавати всі $R[i]$ (уже використані будуть нулями й не вплинуть на результат).

Приклади програм-розв'язків: ideone.com/QwkKWN

(Повний вміст даного посилання ideone.com/QwkKWN такий:

```
var data : array[1..100] of integer;
    i, N, M, k1, k2 : integer;
    overall_sum : longint;
    res : real;
BEGIN
    readln(N);
    for i:=1 to N do begin
        read(data[i]);
        overall_sum := overall_sum + data[i];
    end;
    readln(M);
    for i:=1 to M do begin
        readln(k1, k2);
        res := res + 1/(1/data[k1]+1/data[k2]);
        overall_sum := overall_sum - (data[k1] + data[k2]);
    end;
    writeln(res + overall_sum)
END.
```

— кінець цитати посилання ideone.com/QwkKWN)

(першим способом), ideone.com/QNgTua

(Повний вміст даного посилання ideone.com/QNgTua такий:

```
var
    n,n1,n2,i,m:integer;
    a: array [1..1000] of integer;
    res:real;

begin
    readln (n);
    for i:=1 to n do read (a[i]);
    res:=0;
    readln (m);
    for i:=1 to m do
        begin
            readln (n1,n2);
            res:=res + ((a[n1]*a[n2])/(a[n1]+a[n2]));
            a[n1]:=0;
```

```

    a[n2]:=0;
end;
for i:=1 to n do res:=res+a[i];
writeln (res);
end.

```

— кінець цитати посилання ideone.com/QNgTua)

(другим).

Задача В. «РyCTБCЦPjPsPеCтC,PSPëPеPë вT“ rectangles»

Вхідні дані: Клавiатура (stdin) або файл input.txt
 Результати: Екран (stdout) або файл output.txt
 Обмеження часу: 1 сек
 Обмеження пам'яті: 64 мегабайти

Р”Р°PSC– PтPIP° PтCTБCЦPjPsPеCтC,PSPëPеPë, CтC,PсCTБPsPSPë CЦPе-
 PëC... PтP°CTБP°P»PμP»CHъPSC– PIC–CтCЦPj PеPsPsCTБPтPëPSP°C,.

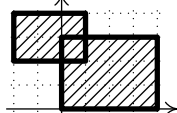
Завдання PкP°PiPëC€C–C,CHъ PтCTБPsPiCTБP°PjCт rectangles,
 CЦPеP° P± PIPëP·PSP°C‡P°P»P° PтP»PsC%0Cт C–C...PSCHъPsPiPs
 PsP±вT™C”PтPSP°PSPSCЦ, C,PсP±C,Pс CтCтC–C”C– C‡P°CтC,PëPSPë
 PтP»PsC%0PëPSPë, C%0Ps PтPsPеCTБPëC,P° C...PsC‡P° P± PsPтPSPëPj P·
 PтCTБCЦPjPsPеCтC,PSPëPеC–PI. PμP»PsC%0Cт CтPтC–P»CHъPSPсC–
 C‡P°CтC,PëPSPë PsP±PsC... PтCTБCЦPjPsPеCтC,PSPëPеC–PI
 (CЦPеC%0Ps C,P°PеP° C”) CтP»C–Pт PICтP°C...PsPICтPIP°C,Pë PsP-
 тPëPS CTБP°P..

Вхідні дані PjC–CтC,CЦC,CHъ PтPIP° CTБCЦPтPеPë, Pе-
 PsP¶PμPS P· CЦPеPëC... PsPiPëCтCтC” PsPтPëPS P· Pт-
 CTБCЦPjPsPеCтC,PSPëPеC–PI, Cт C,,PsCTБPjP°C,C– x_{\min} x_{\max} y_{\min} y_{\max} .
 PJCтC– PеPsPsCTБPтPëPSP°C,Pë C” C‡C–P»PëPjPë C‡PëCтP»P°PjPë,
 C%0Ps PSPμ PтPμCTБPμPIPëC%0CтCTБC,CHъ PтPs PjPsPтCтP»CT 1000.

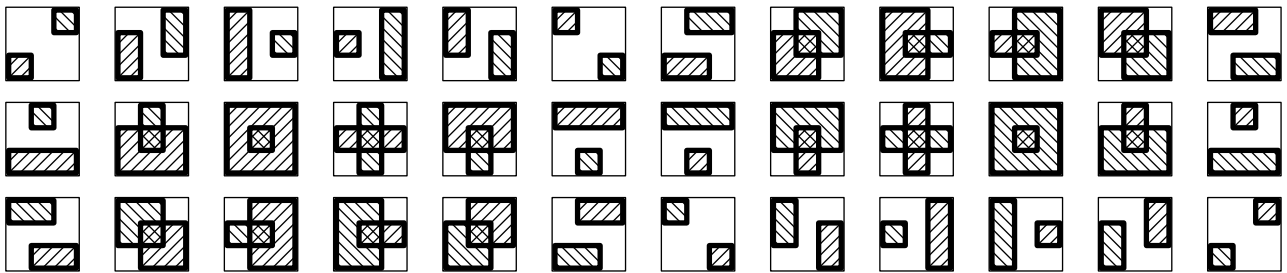
Результати P’P°C€P° PтCTБPsPiCTБP°PjP° PjP°C” PIPëPIPμCтC,Pë
 C”PтPëPSPμ C‡C–P»Pμ C‡PëCтP»Ps — P·PSP°PμPтPμPSCт PтP»PsC%0Cт

$$\text{P}_\text{S}\text{P}_\pm\text{B}\text{T}^\text{TM}\text{C}''\text{P}_\text{r}\text{PSP}^\circ\text{PSPSCl}_2.$$

Приклади

Вхідні дані	Результати	
0 10 0 10 20 30 20 30	200	
0 20 0 30 10 12 17 23	600	
0 4 0 3 -2 1 2 4	17	

Розбір задачі Цю задачу важко вирішити на повні бали шляхом аналізу випадків. Просто тому, що їх більше, ніж може здатися. Навіть якщо не розглядати випадки рівності деяких із координат (як-то «не розглядати окремо ситуацію $x_{\min 1} = x_{\max 2}$, а об'єднати з ситуацією $x_{\min 1} > x_{\max 2}$, написавши $x_{\min 1} \geq x_{\max 2}$ »), все одно лишаються такі потенційно різні випадки:

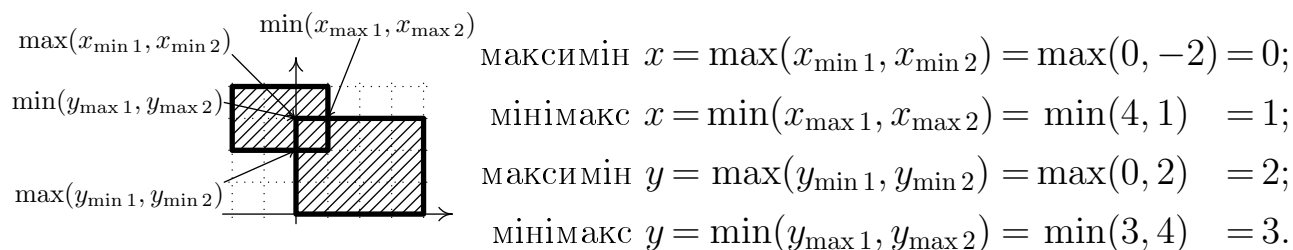


Випадки, коли прямокутники повністю обміняні місцями, наведено як різні, бо якщо іти шляхом аналізу випадків, виводячи для кожного свою формулу залежності результату від вхідних даних, вони можуть бути різними.

Звісно, деякі з цих випадків можна об'єднати. Але не так просто зробити це правильно, ніде не помилітисся. Тому пропонується розв'язати задачу інакше.

1-й спосіб, придатний для довільних координат Площа об'єднання дорівнює сумі площ окремо взятих прямокутників мінус міра перетину (спільної частини); якщо спільної частини нема, площею перетину вважається 0. (Це міркування є частковим випадком *принципу включень та виключень*, https://uk.wikipedia.org/wiki/Формула_включень-виключень).

Щоб знайти перетин, можна (окремо для x , окремо для y) узяти максимум (максимум із мінімумів) і мінімакс (мінімум із максимумів). Наприклад:



Якщо в обох випадках (x та y) максимін строго менший мінімакса, то площа перетину ненульова і її справді треба відняти.

Приклад реалізації цього способу: ideone.com/naDNCA

(Повний вміст даного посилання ideone.com/naDNCA такий:

```
{ $mode delphi }

function min(a, b : LongInt) : LongInt;
begin
    if a < b then
        Result := a
    else
        Result := b
end;

function max(a, b : LongInt) : LongInt;
begin
    if a > b then
        Result := a
    else
        Result := b
end;

var
    x1min, x1max, y1min, y1max,
    x2min, x2max, y2min, y2max,
    xCmin, xCmax, yCmin, yCmax,
    main_result : LongInt;

begin
    Readln(x1min, x1max, y1min, y1max);
    Readln(x2min, x2max, y2min, y2max);
    xCmin := max(x1min, x2min);
    xCmax := min(x1max, x2max);
    yCmin := max(y1min, y2min);
    yCmax := min(y1max, y2max);
    main_result :=
        (x1max-x1min)*(y1max-y1min) +
        (x2max-x2min)*(y2max-y2min);
    if (xCmax > xCmin) and (yCmax > yCmin) then
```



```
    main_result := main_result -  
        (xCmax-xCmin)*(yCmax-yCmin);  
    Writeln(main_result)  
end.
```

— кінець цитати посилання ideone.com/naDNCA)

2-й спосіб, менш універсальний, але тут і зараз теж правильний

В умові задано досить невеликі (як для комп'ютера) обмеження на значення координат. Це дає можливість тупо перебрати всі «клітинки» і для кожної з них перевірити, чи належить вона 1-му прямокутнику та чи належить 2-му. Важливо, що завдяки використанню операції `or` (C-подібними мовами `||`) можна не розбиратися з випадками, бо абсолютно не важливо, чи мають прямокутники непорожній перетин, чи не мають.

Приклад реалізації цього способу: ideone.com/BH0tpE

(Повний вміст даного посилання ideone.com/BH0tpE такий:

```
{ $mode delphi }  
  
var  
    x1min, x1max, y1min, y1max,  
    x2min, x2max, y2min, y2max,  
    x, y, main_result : LongInt;  
  
begin  
    Readln(x1min, x1max, y1min, y1max);  
    Readln(x2min, x2max, y2min, y2max);  
    for x := -1000 to 999 do begin  
        for y := -1000 to 999 do begin  
            if (x+0.5 > x1min) and (x+0.5 < x1max) and  
                (y+0.5 > y1min) and (y+0.5 < y1max)  
                or  
                (x+0.5 > x2min) and (x+0.5 < x2max) and  
                (y+0.5 > y2min) and (y+0.5 < y2max) then  
                Inc(main_result);  
            end;  
        end;  
        Writeln(main_result);  
    end.
```

— кінець цитати посилання ideone.com/BH0tpE)

Код навіть коротший і простіший, ніж для першого способу. Але якби значення координат були не від -1000 до $+1000$, а довільні (наприклад,

дозволялися дробові), цей спосіб був би принципово неможливим; якби координати були цілими, але з більшого діапазону — програма працювала б надто довго. Власне, вона і зараз працює довше, ніж 1-й спосіб; але при цьому вкладається в обмеження.

Задача С. «Генератор паролів — password»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	1 сек
Обмеження пам'яті:	64 мегабайти

У генераторі паролів закладена схема, яка за певними правилами утворює паролі із комбінацій цифр та великих літер англійського алфавіту. Цифри та літери в паролі можуть розташовуватись лише за зростанням (для літер зростання визначається алфавітом). Паролі містять, як мінімум, один символ (літеру або цифру). Якщо пароль містить і цифри, і літери, то цифри завжди йдуть після літер. Повторення символів не допускається.

Наприклад: SN15 — коректний пароль, а OLYMPIAD — некоректний пароль (оскільки літери розташовані не в алфавітному порядку).

Усі паролі генеруються послідовно у вигляді впорядкованого списку. Якщо два паролі містять різну кількість символів, то першим іде пароль з меншою кількістю. Якщо декілька паролів мають однакову кількість символів, то вони генеруються в алфавітному порядку (при цьому літери вважаються меншими за цифри).

Початок впорядкованого списку паролів має наступний вигляд: A, B, ..., Z, 0, 1, 2, ..., 9, AB, AC, ..., A9, BC, ...

Завдання Напишіть програму `password`, яка визначатиме n -ий пароль у списку паролів, який утворить генератор.

Вхідні дані Число n ($1 \leq n \leq 10^{10}$).

Результати Ваша програма має вивести n -ий пароль.

	Вхідні дані	Результати
Приклади	1	A
	37	AB

Оцінювання Приблизно 50% балів припадає на тести, в яких $n \leq 50000$. Ще приблизно 25% — на тести, в яких $n \leq 10^7$. Решта (приблизно 25%) — на тести, в яких $10^9 \leq n \leq 10^{10}$.

Примітка Англійський алфавіт має вигляд A B C D E F G H I J K L M N O P Q R S T U V W X Y Z. Літери алфавіту у таблиці ASCII йдуть поспіль, під номерами від 65 (“A”) до 90 (“Z”). Зростаюча послідовність цифр 0 1 2 3 4 5 6 7 8 9 також неперервна у таблиці ASCII (але не поспіль з літерами). Цифри йдуть під номерами від 48 (“0”) до 57 (“9”).

Розбір задачі Розглянуті правила не мають нічого спільного з реальними засобами генерації паролів, тобто легенда (“казочка”) даної задачі сильно відірвана від життя. Тим не менш, будемо користуватися термінами «пароль» і «номер паролю», раз уже вони введені в умові задачі.

Позбудемось окремих статусів літер і цифр та перейдемо до єдиного алфавіту В умові досить багато і трохи заплутано розказано про порядок літер між собою, цифр між собою, літер та букв у рамках одного паролю. Взагалі-то всі ті правила можна істотно спростити до таких:

1. будемо вважати алфавітом A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (саме в такому порядку);
2. всередині кожного пароля (крім 1-символьних), кожен наступний символ мусить бути строго більшим за попередній (згідно алфавіту п. 1).

Переформулювати правила порівняння різних паролів нема потреби.

Частковий розв’язок на 76 балів (зі 100) Оскільки в умові сказано, що досить багато балів припадає на не дуже великі значення n , можна писати перебір, тобто дійсно генерувати послідовно 1-й, 2-й, ... паролі аж до n -го. Виявляється, при цьому не дуже важко добитися, щоб генерувалися відразу лише допустимі (згідно з умовою задачі) паролі.

Розглянемо простіший випадок. Нехай потрібно вивести послідовно пари (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5), тобто пари з чисел від 1 до 5, щоб 2-й елемент пари завжди був строго більшим за 1-й. Це можна зробити за допомогою циклів

```
for i:=1 to 5 do
  for j:=1 to 5 do
    if i < j then
      writeln(i, ', ', j);
```

А можна зробити інакше:

```
for i:=1 to 4 do
  for j:=i+1 to 5 do
    writeln(i, ', ', j);
```

тобто замінити *перевірку* $i < j$ на *задання лише потрібного* діапазону. І якщо для 2-х вкладених циклів це дрібна оптимізація, то для, наприклад, 5-ти вкладених циклів пришвидшення істотне (\approx сотні разів).

Приклад такого розв'язку — ideone.com/Xg6sKT

(Повний вміст даного посилання ideone.com/Xg6sKT такий:

```
{This solution is an extra-optimized brute-force.
It should gain about 75% of points.}

program a;

{$APPTYPE CONSOLE}

var s : string;
    ABC012 : string;
    c : Char;
    N, passwords_generated : longint;
    i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12 : byte;

BEGIN
  ABC012 := '';
  for c := 'A' to 'Z' do
    ABC012 := ABC012 + c;
  for c := '0' to '9' do
```

```
    ABC012 := ABC012 + c;  
Readln(N);  
passwords_generated := 0;  
  
s:='A';  
for i1 := 1 to 36 do begin  
    s[1] := ABC012[i1];  
    Inc(passwords_generated); //writeln(passwords_generated, #9, s);  
    if passwords_generated = N then begin  
        Writeln(s); exit;  
    end;  
end;  
  
s:='AB';  
for i1 := 1 to 35 do begin  
    s[1] := ABC012[i1];  
    for i2 := i1+1 to 36 do begin  
        s[2] := ABC012[i2];  
        Inc(passwords_generated); //writeln(passwords_generated, #9, s);  
        if passwords_generated = N then begin  
            Writeln(s); exit;  
        end;  
    end;  
end;  
  
s:='ABC';  
for i1 := 1 to 34 do begin  
    s[1] := ABC012[i1];  
    for i2 := i1+1 to 35 do begin  
        s[2] := ABC012[i2];  
        for i3:=i2+1 to 36 do begin  
            s[3] := ABC012[i3];  
            Inc(passwords_generated); //writeln(passwords_generated, #9, s);  
            if passwords_generated = N then begin  
                Writeln(s); exit;  
            end;  
        end;  
    end;  
end;  
  
s:='ABCD';  
for i1:=1 to 33 do begin  
    s[1] := ABC012[i1];  
    for i2:=i1+1 to 34 do begin  
        s[2] := ABC012[i2];  
        for i3:=i2+1 to 35 do begin  
            s[3] := ABC012[i3];  
            for i4:=i3+1 to 36 do begin  
                s[4] := ABC012[i4];  
                Inc(passwords_generated); //writeln(passwords_generated, #9, s);  
                if passwords_generated = N then begin  
                    Writeln(s); exit;  
                end;  
            end;  
        end;  
    end;  
end;
```

```
    end;
  end;
end;

s:='ABCDE';
for i1:=1 to 32 do begin
  s[1] := ABC012[i1];
  for i2:=i1+1 to 33 do begin
    s[2] := ABC012[i2];
    for i3:=i2+1 to 34 do begin
      s[3] := ABC012[i3];
      for i4:=i3+1 to 35 do begin
        s[4] := ABC012[i4];
        for i5:=i4+1 to 36 do begin
          s[5] := ABC012[i5];
          Inc(passwords_generated); //writeln(passwords_generated, #9, s);
          if passwords_generated = N then begin
            Writeln(s); exit;
          end;
        end;
      end;
    end;
  end;
end;

s:='ABCDEF';
for i1:=1 to 31 do begin
  s[1] := ABC012[i1];
  for i2:=i1+1 to 32 do begin
    s[2] := ABC012[i2];
    for i3:=i2+1 to 33 do begin
      s[3] := ABC012[i3];
      for i4:=i3+1 to 34 do begin
        s[4] := ABC012[i4];
        for i5:=i4+1 to 35 do begin
          s[5] := ABC012[i5];
          for i6:=i5+1 to 36 do begin
            s[6] := ABC012[i6];
            Inc(passwords_generated); //writeln(passwords_generated, #9, s);
            if passwords_generated = N then begin
              Writeln(s); exit;
            end;
          end;
        end;
      end;
    end;
  end;
end;

s:='ABCDEFG';
for i1:=1 to 30 do begin
  s[1] := ABC012[i1];
  for i2:=i1+1 to 31 do begin
    s[2] := ABC012[i2];
```

```
for i3:=i2+1 to 32 do begin
  s[3] := ABC012[i3];
  for i4:=i3+1 to 33 do begin
    s[4] := ABC012[i4];
    for i5:=i4+1 to 34 do begin
      s[5] := ABC012[i5];
      for i6:=i5+1 to 35 do begin
        s[6] := ABC012[i6];
        for i7:=i6+1 to 36 do begin
          s[7] := ABC012[i7];
          Inc(passwords_generated); //writeln(passwords_generated, #9, s);
          if passwords_generated = N then begin
            Writeln(s); exit;
          end;
        end;
      end;
    end;
  end;
end;
end;
end;
end;
end;
END.
```

— кінець цитати посилання ideone.com/Xg6sKT)

Зовсім не взірець красоти й лаконічності. У ньому легко допустити і важко шукати технічні помилки. Але він все ж набирає чимало балів.

Повний (100%) розв'язок Перш за все, загальна кількість k -символьних паролів рівна $C(36, k)$, де $C(n, k)$, воно ж C_n^k — кількість сполучень (рос. сочетания, англ. combinations) з n по k . Це так, бо з 36 символів вибираються k різних, і одні й ті самі вибрані символи не можна переставляти місцями, бо дозволений лише порядок за зростанням (у алфавіті А, В, ..., Z, 0, 1, ..., 9). Так що почнемо розв'язок з того, що визнаємо довжину (кількість символів) шуканого пароля і його номер *серед паролів цієї довжини*.

Наприклад, прочитали у вхідних даних 2015.

- Кіль-ть 1-символьних паролів $C_{36}^1=36$, $36 < 2015$ — отже, у паролі більше одного символу, й номер серед (більш-ніж-1)-символьних $2015 - 36 = 1979$.
- Кіль-ть 2-символьних паролів $C_{36}^2=630$, $630 < 1979$ — отже, у паролі більше двох символів, і номер серед (більш-ніж-2)-символьних $1979 - 630 = 1349$.

- Кіль-ть 3-символьних паролів $C_{36}^3=7140$, $7140 \geq 1349$ — отже, пароль 3-символьний, і його номер серед 3-символьних рівний 1349.

Узнавши довжину пароля та його номер серед паролів відповідної довжини, починаємо визнавати цей пароль символ за символом, зліва направо — на тій підставі, що для кожного можливого початку можна визнавати (засобами комбінаторики) кількість паролів з таким початком і знову приймати рішення, чи цей початок треба пропустити (всі паролі з таким початком мають менші номери), чи використати (потрібний номер якраз потрапляє у діапазон).

Продовжимо аналіз того самого прикладу (вхідний номер 2015, раніше з'ясовано, що його номер серед 3-символьних рівний 1349).

- Кіль-ть 3-символьних паролів, що починаються з А, рівна $C_{35}^2 = 595$, бо продовження 2-символьне з 35 символів (від В до 9). $1349 > 595$ — отже, початок паролю не А, а якийсь подальший символ, і номер серед тих подальших $1349 - 595 = 754$.
- Кіль-ть 3-символьних паролів, що починаються з В, рівна $C_{34}^2 = 561$, бо продовження 2-символьне з 34 символів (від С до 9). $754 > 561$ — отже, початок паролю не В, а якийсь подальший символ, і номер серед тих подальших $754 - 561 = 193$.
- Кіль-ть 3-символьних паролів, що починаються з С, рівна $C_{33}^2 = 528$, бо продовження 2-символьне з 33 символів (від D до 9). $193 \leq 528$ — отже, початок паролю якраз-таки С, і його номер серед 3-символьних, що починаються з С — теж 193.

Далі відбувається аналогічний підбір наступного символу:

- Кіль-ть 3-символьних паролів, що починаються з “CD”, рівна $C_{32}^1 = 32$, бо лишається дописати один символ, від Е до 9. $193 > 32$ — отже, 2-га літера не D, а якийсь подальший символ, і номер серед тих подальших $193 - 32 = 161$.
- : І так далі.
- Продовживши аналогічні міркування, отримаємо, що шуканий пароль 16-й серед тих, що починаються з “CJ”, а оскільки після J можуть іти лише символи, починаючи з K, то цим 16-м буде Z.

! Остаточню, 2015-й пароль має вигляд “CJZ”.

Дані пояснення займають багато місця, але виключно тому, що наведено приклад (а у першому алгоритмі ніякий приклад не наводився). Правильна реа-

лізація даного комбінаторного алгоритму працює дуже швидко, вкладаючись у секунду з величезним запасом (у тисячі разів). Адже всього-то треба:

1. Познаходити $C(n, k)$, наприклад, усі зразу із проміжку $0 \leq k \leq n \leq 36$ за допомогою трикутника Паскаля.
2. Знайти кількість символів у паролі — віднімати циклом $C_{36}^1, C_{36}^2, \dots$; якби дійшли до C_{36}^{36} , а номер після усіх віднімань все ще лишався надто великим — це означало б, що пароля вказаних вигляду і номера взагалі не існує. Але такого не буде, бо паролів $(2^{36} - 1)$ все-таки більше, чим $n \leq 10^{10}$. Значить — тут не більш як 36 порівнянь та віднімань.
3. Для кожної позиції (1-й символ, 2-й, ...) запустити цикл, щоб знайти конкретне значення відповідного символу — теж не багато, бо і позицій, і значень символів не більше 36.

Якби розмір алфавіту був змінним (і при цьому не з'являлася «довга» арифметика), можна було б говорити про час роботи $O(A^2)$, де A — розмір алфавіту. При $A=36$, це *дуже* швидко...

Задача D. «Замок — castle»

Вхідні дані:	Клавіатура (stdin) або файл input.txt
Результати:	Екран (stdout) або файл output.txt
Обмеження часу:	1 сек для D1, 3 сек для D2
Обмеження пам'яті:	128 мегабайтів

Стародавній замок має прямокутну форму. Замок містить щонайменше дві кімнати. Підлогу замка можна умовно поділити на $M \times N$ клітин. Кожна така клітинка містить «0» або «1», які задають порожні ділянки та стіни замку відповідно.

Завдання Напишіть програму `castle`, яка б знаходила кількість кімнат у замку, площу найбільшої кімнати (яка вимірюється кількістю клітинок) та площу найбільшої кімнати, яку можна утворити шляхом видалення стіни або її частини, тобто, замінивши лише одну «1» на «0». Видаляти зовнішні стіни

заборонено.

Вхідні дані План замку задається у вигляді послідовності чисел, по одному числу, яке характеризує кожну клітинку. Перший рядок містить два цілих числа M та N — кількість рядків та кількість стовпчиків ($3 \leq M \leq 1000$, $3 \leq N \leq 1000$). M наступних рядків містить по N нулів або одиниць, що йдуть поспіль (без пробілів). Перший та останній рядок, а також перший та останній стовпчик формують зовнішні стіни замку і складаються лише з одиниць.

Результати Дана задача розділена в системі ejudge на дві підзадачі. У підзадачі D1 треба здати програму, що знаходить кількість кімнат та площу найбільшої кімнати замку (по одному числу в рядку), у підзадачі D2 — площу найбільшої кімнати, яка утвориться в разі видалення внутрішньої стіни.

Приклади

Вхідні дані	Результати (D1)	Результати (D2)
6 8 11111111 10011001 10011001 11111001 10101001 11111111	4 8	10
9 12 111111111111 101001000001 111001011111 100101000001 100011111101 100001000101 111111010101 100000010001 111111111111	4 28	38

Оцінювання Значна частина тестів буде містити план замку з кімнатами лише прямокутної форми. Не менше половини тестів такі, що $3 \leq M \leq 20$, $3 \leq N \leq 50$.

Розбір задачі Для самої лише підзадачі D1 і часткового випад-

ку «всі кімнати мають прямокутну форму» можна написати програму `ideone.com/HHr9a3`

(Повний вміст даного посилання ideone.com/HHr9a3 такий:

```
program Directonly;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var
  data : array[1..1000] of string;
  i, j, N, M : Integer;
  num_rooms, max_sz, curr_sz, curr_h, curr_w : Integer;

begin
  Readln(N,M);
  for i:=1 to N do
    Readln(data[i]);
  num_rooms := 0;
  max_sz := 0;
  for i:=2 to N-1 do
    for j:=2 to M-1 do
      if (data[i][j] = '0') and (data[i-1][j] = '1') and (data[i][j-1] = '1') then begin
        Inc(num_rooms);
        curr_h := 1;
        while data[i + curr_h][j] = '0' do
          Inc(curr_h);
        curr_w := 1;
        while data[i][j + curr_w] = '0' do
          Inc(curr_w);
        curr_sz := curr_h * curr_w;
        if curr_sz > max_sz then
          max_sz := curr_sz;
      end;
    end;
  Writeln(num_rooms);
  Writeln(max_sz);
end.
```

— кінець цитати посилання ideone.com/HHr9a3)

Вона спирається на те, що у випадку прямокутності кімнат кожна кімната однозначно задається лівим верхнім кутом, а перевіряти, чи справді клітинка є таким кутом, можна умовою $(data[i][j]='0')$ and $(data[i-1][j]='1')$ and $(data[i][j-1]='1')$, тобто сама клітинка вільна, а ліворуч і згори стіни. Очевидно (в т. ч. з 2-го тесту з умови), що для “закручених” кімнат це може й не бути правдою. Але в умові обіцяно значну частину тестів з кімнатами

прямокутної форми, тож при відсутності кращих ідей можна написати хоча б такий розв'язок. Він набирає 26 балів (з 50 за усю D1, зі 100 за усю D).

Повний розв'язок підзадачі D1 Для відстеження (як завгодно “закручених”) кімнат можна реалізувати будь-який з алгоритмів:

1. пошук ушир (він же пошук у ширину); рос. поиск в ширину, англ. breadth first search (BFS);
2. пошук углиб (він же пошук у глибину); рос. поиск в глубину, англ. depth first search (DFS);
3. різноманітні алгоритми графічної “заливки” (укр., рос. — заливка, англ. — flood fill).

Ці алгоритми неважко знайти в Інтернеті або в літературі самостійно, тож не будемо наводити їх тут.

Часто вважають, що в такій ситуації найпростішою є рекурсивна реалізація пошуку вглиб. Частково це правда, але рекурсивний пошук углиб має потенційний недолік: пам'яті, потрібної для зберігання рекурсії у програмному стекові, може бути набагато менше, ніж пам'яті взагалі. Як наслідок, рекурсивна реалізація DFS може завершуватися аварійно по причині нестачі пам'яті — навіть при фактичних витратах пам'яті, менших, ніж у пошуку вшир або заливці. Стекова пам'ять може бути значно «дефіцитнішою», й коли *вона* закінчилася, наявність іншої пам'яті рекурсії не допомагає.

Саме *може* бути. А може бути й інакше. І залежить це від налаштувань компілятора (керування якими доступне програмісту під час «нормальної» роботи, але як правило не доступне учаснику олімпіади). Конкретно на цій обласній олімпіаді було забезпечено досить великий розмір програмного стеку, тому можна було писати рекурсивний DFS і не мати проблем; але біда в тому, що при інших налаштуваннях компілятора проблеми цілком можливі.

Підзадача D2 Певну частину балів (орієнтовно до 20 з 50) можна отри-

мати, розв'язуючи підзадачу D1 багатократно (для абсолютно кожної «1» у внутрішній стіні, замінимо її на «0» і заново розв'яжемо D1; серед усіх таких розв'язків виберемо максимальний). Але це ніяк не може бути повноцінним ефективним розв'язком для розмірів порядку 1000×1000 .

Перепишемо підзадачу D1 так, щоб при підрахунку кількостей та розмірів кімнати не просто виділялися, а *різні кімнати* виділялися *різними значеннями* (а клітинки однієї кімнати — однаковими). Наприклад, якщо це робити прямо у масиві з позначками «0 — прохід, 1 — стіна», може вийти так:

Вхідні дані	Виділені кімнати	Масив з розмірами кімнат				
9 12		індекс	...	2	3	4 5
111111111111	1 1 1 1 1 1 1 1 1 1 1 1	значення	...	1	5	28 9
101001000001	1 2 1 3 3 1 4 4 4 4 4 1					
111001011111	1 1 1 3 3 1 4 1 1 1 1 1					
100101000001	1 5 5 1 3 1 4 4 4 4 4 1					
100011111101	1 5 5 5 1 1 1 1 1 1 4 1					
100001000101	1 5 5 5 5 1 4 4 4 1 4 1					
111111010101	1 1 1 1 1 1 4 1 4 1 4 1					
100000010001	1 4 4 4 4 4 4 1 4 4 4 1					
111111111111	1 1 1 1 1 1 1 1 1 1 1 1					

Якщо при цьому ще й зберігати розміри кімнат у масиві (так, щоб індексами масиву були ті самі чісла, якими позначено кімнати), то для визначення, яка вийде площа кімнати після руйнування деякої стіни, можна просто додати до одинички (площі самої зруйнованої стіни) площі кімнат-сусідів. Так що перебір усіх можливих «1» (у не-зовнішніх стінах) можна залишити, бо тепер для кожної такої «1» треба робити значно менше дій.

Здається «логічним» (і приклади з умови це «підтверджують»), ніби максимальна кімната буде утворена за рахунок об'єднання двох кімнат. Але насправді це лише поширений випадок, а не обов'язкова властивість: замість 2-х може бути будь-яке число від 1 до 4 (див. приклади). Тому краще, не роблячи необґрунтованих припущень, акуратно реалізувати для кожної не зовнішньої «1» перегляд усіх сусідів-кімнат, додаючи площі усіх різних.

Тест №9. Більшість внутрішніх стін «товсті», тож вилучення *однієї* «1» зазвичай не призводить до з'єднання кімнат. А в тому єдиному місці, де призводить (2-й знизу рядок) — утворюється кімната площею $2 + 1 + 1 = 4$. Набагато більшу площу $44 + 1 = 45$ можна отримати, зруйнувавши будь-яку (не зовнішню) стіну кімнати площі 44, утворивши «нішу» замість «проходу».

```

13 25
11111111111111111111111111111111
10000001100001100001100001100001
10000001100001100001100001100001
10000001100001100001100001100001
10000001100001100001100001100001
100000011000011111111100001
100000011000011111111100001
11111111100001100001100001100001
11111111100001100001100001100001
100000011000011111111100001
100000011000011111111100001
10000001100001100101100001
11111111111111111111111111111111

```

Тест №16 (лівіший з двох прикладів). Руйнування однієї «1» по центру призводить до з'єднання *відразу чотирьох* кімнат. Переконавшись у можливості такої ситуації, легко уявити і вхідні дані правішого з двох прикладів, де різні сусіди центральної одинички до того ж ще й з'єднані в одну кімнату, але десь далеко.

9 11	9 11
111111111111	111111111111
10011011001	10011011001
11001010011	11000010011
11101010111	11101010111
10100100101	10100100101
11101010111	11101010111
11001010011	11001010011
10011011001	10011011001
111111111111	111111111111

Отже — акуратно реалізувати для кожної не зовнішньої «1» перегляд усіх сусідів-кімнат, додаючи площі усіх різних.

3mict

1	Обласна інтернет-олімпіада 2013/14 н. р.	3
	А «Тор»	3
	В «Паркет-1»	4
2	II (районний/міський) етап 2013/14 н. р.	7
	А «Електричка»	8
	В «Цифрові ріки»	9
	С «Логічний куб»	12
	Д «Всюдисущі числа»	15
3	Дистанційний тур III (обласного) етапу 2013/14 н. р.	19

A «ISBN»	20
B «Точні квадрати»	22
C «Ложбан»	24
D «Графічний пароль»	27
4 Обласна інтернет-олімпіада 2014/15 н. р.	29
A «Три круги»	30
B «Перевезення вантажу»	32
C «Коло і точки»	34
5 II (районний/міський) етап 2014/15 н. р.	40
A «Цифра»	40
B «Піраміда»	41
C «Ко-анаграмічно-прості»	44
D «Хмарочоси»	49
6 III (обласний) етап 2014/15 н. р.	51
A «Гірлянда — garland»	51
B «РџСѢСЇРјРѕРєСѐС,РSPёРєРё вѢ“ rectangles»	54
C «Генератор паролів — password»	58
D «Замок — castle»	65