





## Зміст

<b>1</b>	<b>Передмова</b>	<b>5</b>
<b>2</b>	<b>Огляд особливостей «олімпіадного» та «алгоритмічного» програмування</b>	<b>6</b>
<b>3</b>	<b>Задачі та їхні розв'язки</b>	<b>15</b>
3.1	Обласна інтернет-олімпіада 2013/14 н. р. . . . . .	15
	Задача А. «Тор» . . . . .	15
	Задача В. «Паркет–1» . . . . .	17
	Задача С. «Сума квадратів» . . . . .	20
	Задача D. «Дільники» . . . . .	22
3.2	II (районний/міський) етап 2013/14 н. р. . . . . .	23
	Задача А. «Електричка» . . . . .	23
	Задача В. «Цифрові ріки» . . . . .	25
	Задача С. «Логічний куб» . . . . .	26
	Задача D. «Всюдисущі чісла» . . . . .	28
3.3	Дистанційний тур III (обласного) етапу 2013/14 н. р. . . . . .	32
	Задача А. «ISBN» . . . . .	32
	Задача В. «Точні квадрати» . . . . .	34
	Задача С. «Ложбан» . . . . .	35
	Задача D. «Графічний пароль» . . . . .	37
3.4	Обласна інтернет-олімпіада 2014/15 н. р. . . . . .	38
	Задача А. «Три круги» . . . . .	38
	Задача В. «Перевезення вантажу» . . . . .	40
	Задача С. «Коло і точки» . . . . .	42

Збірник задач та їхніх розв'язків обласних, районних (міських)  
олімпіад з інформатики Черкаської області, (2013–2015 р.р.)

---

3.5	II (районний/міський) етап 2014/15 н. р. . . . . .	45
	Задача А. «Цифра» . . . . .	45
	Задача В. «Піраміда» . . . . .	47
	Задача С. «Ко-анаграмічно-прості» . . . . .	50
	Задача D. «Хмарочоси» . . . . .	52
3.6	III (обласний) етап 2014/15 н. р. . . . . .	54
	Задача А. «Гірлянда — garland» . . . . .	55
	Задача В. «Прямокутники — rectangles» . . . . .	56
	Задача С. «Генератор паролів — password» . . . . .	58
	Задача D. «Зámок — castle» . . . . .	62
3.7	Обласна інтернет-олімпіада 2015/16 н. р. . . . . .	67
	Задача А. «Високосні роки» . . . . .	67
	Задача В. «Фасування олії» . . . . .	69
	Задача С. «Нестандартне рівняння» . . . . .	70
	Задача D. «П'ять неділь у місяці» . . . . .	71
3.8	II (районний/міський) етап 2015/16 н. р. . . . . .	74
	Задача А. «Купе» . . . . .	74
	Задача В. «Гра» . . . . .	75
	Задача С. «Сучасне мистецтво» . . . . .	77
	Задача D. «Кількість ділянок на проміжку» . . . . .	81

**4 Список використаних джерел**

**85**

## 1 Передмова

Цей збірник містить задачі олімпіад з інформатики (програмування), що відбувалися у Черкаській області у 2013/14 та 2014/15 навчальних роках. Точніше, він охоплює тури обласних інтернет-олімпіад, а також другі (районні/міські) та треті (обласні) етапи Всеукраїнської олімпіади з інформатики. Для кожної задачі наведені умова та вказівки щодо її розв'язування.

Умови задач, що входять до збірника, формувалися авторським колективом у складі Порубльова І. М. (ЧНУ), Богатирьова О. О. (ЧНУ), Фурника І. В. (ЧОІПОПП), Шемшура В. М. (ЧОІПОПП), Безпальчука В. М. (ЧНУ), Черненко Р. В. (випускник ЧНУ 2011 р.), Поліщука Д. І. (випускник ФіМЛі 2004 р.). Умови задач наведені без редагування, у вигляді, максимально близькому до того, який вони мали на самих етапах олімпіади.

Більшість тексту розборів (пояснень) до задач написана Порубльовим І. М., з урахуванням порад вищезгаданих авторів. Крім того, низку слушних пропозицій висловив Полосухін В. А. (випускник ФіМЛі 2014 р., який брав участь лише у підготовці розборів після відповідних турів). У деяких окремих випадках використані також розв'язки учнів — учасників відповідних турів олімпіади.

Збірник не призначений замінити підручник з програмування. Використовуючи конкретні задачі, нереально побудувати збалансований курс, що розглядає продуманий перелік тем; та й співвідношення обсягу задач (23 задачі з 6 турів) та обсягу збірника (85 сторінок) робить неможливим детальний розгляд усіх потрібних для розв'язання цих задач алгоритмів. Тому основна увага у збірнику приділена поясненням нестандартних рішень у цих задачах. Коли задача зводиться до реалізації відомого алгоритму, зазвичай наводиться посилання на джерела в мережі Інтернет або літературу. Рекомендується використовувати цей збірник у поєднанні із підручниками з програмування, монографіями та/або сайтами, де розглядаються ефективні алгоритми, тощо.

Розділ 2 (стор. 6–15) містить огляд специфічних питань, які часто використовуються під час обговорення саме олімпіадних задач.

На олімпіаді з інформатики (програмування) розв'язком задачі учасником є програма. Даний збірник майже не містить текстів таких програм, але вони доступні як посилання на сайт [ideone.com](http://ideone.com). Там їх можна бачити з підсвіткою синтаксиса, скачувати (у вигляді, придатному для редагування і компілювання), а при бажанні — створювати власну копію (ця дія називається «fork») і працювати з нею безпосередньо на сайті (це буває зручно, якщо потрібно швидко щось спробувати на чужому комп'ютері, де нема середовища програмування). Крім того, у розділі «Олімпіади» форуму сайту [cit.skirp.edu.ua](http://cit.skirp.edu.ua) розміщено і цю версію збірника, і версію, де всі тексти програм розміщено у самому збірнику.

## 2 Огляд особливостей «олімпіадного» та «алгоритмічного» програмування

**Який формат задач?** На олімпіаді з інформатики (не інформаційних технологій, а інформатики; пишуть також «інформатики (програмування)») учасник отримує текстові задачі, розв'язком кожної з яких повинна бути програма мовою програмування, що зчитує вхідні дані (з клавіатури чи текстового файлу), обробляє їх згідно з умовою задачі та виводить результат (на екран чи у текстовий файл). Існують також деякі інші формати, але вони рідко використовуються. «Міркування на тему», виражені учасниками словесно, не приймаються і не перевіряються ще з кінця 1990-х рр.

**Звідки беруться оцінки, проміжні між 0 і максимумом?** Як правило, кожна задачі перевіряють за допомогою *тестів*. Це розроблена (автором задачі або журі) сукупність прикладів, кожен з яких містить вхідні дані й відповідну їм правильну відповідь. Кожен розв'язок кожного учасника запускається на усіх цих прикладах, і за ті з них, де програма вивела правильний результат, уклавшись у обмеження за часом та обсягом пам'яті, нараховуються бали.

Існують програми, які виводять інколи правильні відповіді, інколи неправильні. Бувають правильні, але неефективні програми, які частину тестів прохо-

дять (отримуючи бали), а на решті отримують замість балів вердикт «Перевищено час роботи». І так далі. Звідси й різні бали.

У цього способу оцінювання є недоліки. Наприклад, описані на стор. 28 засоби набрати частину балів за задачу «Логічний куб» можна сприймати як нечесні. Але коли учасники розуміють систему оцінювання, а розробник тестів дбає, щоб відсоток балів за надто прості неповні розв'язки був не таким великим — ці недоліки нівелюються. Крім того, поки що нема кращого способу проводити *автоматичну* перевірку програм, яка все-таки і швидка, і вільна від недоліків, подібних до «член журі неправильно прочитав почерк учасника».

У деяких задачах можливі різні правильні відповіді. Скажімо, у тому ж «Логічному кубі» слід виводити мінімальний шлях, і різні шляхи (послідовності вершин) можуть мати однакову мінімальну довжину. На якісно підготовленій олімпіаді для таких задач пишуть т. зв. *чекери* (*checkers*), які перевіряють відповідь програми учасника по смислу. Чекери пишуть автори задач або члени журі, а не учасники, тож кому цікаво — шукайте деталі самостійно (зокрема, на [codeforces.com](http://codeforces.com)), кому ні — можна обмежитися правилом, що у таких ситуаціях треба виводити будь-яку одну правильну відповідь.

**Чи можна здавати багато розв'язків однієї задачі?** Наразі на Всеукраїнській олімпіаді з інформатики учасники можуть багатократно здавати свої розв'язки у перевіряючу систему, визнавати результати (тривалість перевірки — від кількох секунд до кількох хвилин), і з усіх спроб автоматично вибирається найкращий результат. Обмеження на кількість спроб задачі існують, але лояльні (як-то «до 60 на учасника сумарно по всіх задачах»).

Інша справа, що правила не завжди були такими (на IV (фінальному) етапі вони такі з 2012 р., на III (обласному) — з 2013 р.), й невідомо, як довго вони будуть такими. Існує точка зору, що ці правила «спонукають клацати замість думати», тож на деяких інших змаганнях з програмування залишили «старий, але правильніший» формат, коли перевірка на *повному* наборі тестів відбувається *після* туру, а під час туру учасник визнає лише, чи проходить його програма

всього кілька тестів. Є й системи оцінювання, коли зайві спроби здачі можливі, але якимось «штрафуються». Краще уточнювати такі моменти перед туром.

**Які допустимі мови програмування?** Згідно Правил Всеукраїнської олімпіади з інформатики — C++, C, Pascal. Сервер Черкаського ОІПОПП [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua) їх підтримує (C++ — як g++, C — gcc, Pascal — Free Pascal). Фактично на [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua) наявні також Python, Java, Perl, моно C#, Free Basic, і на більшості олімпіад, що відбуваються на цьому сервері, якщо учасник хоче — може здавати програми цими мовами, отримані бали враховуються. Але адміністрація серверу та журі туру *не* несуть відповідальності та *не* приймають претензій, якщо виявиться, що деякі задачі деякими з не рекомендованих мов не можуть бути розв'язані на повні бали, бо, наприклад, навіть найкращий алгоритм не на всіх тестах вкладається у обмеження за часом чи обсягом пам'яті. (Для C++, C, Pascal така відповідальність є.)

Коли у тексті цього збірника згадується, що якусь частину задачі мовою Pascal треба писати самому, а мовою C++ можна використати готову бібліотечну функцію — часто (але не завжди) подібна функція є також і в мовах Python та Java. Так що знання цих мов (якщо, звісно, вони доступні на потрібному турі) все ж може розширити можливості учасника.

**Як працювати у ejudge?** В Інтернеті багато інструкцій щодо роботи з ejudge. Наприклад, на офіційному сайті розробника, починаючи з [ejudge.ru/wiki/index.php/Веб-інтерфейс\\_пользователя](http://ejudge.ru/wiki/index.php/Веб-інтерфейс_пользователя) Ще одна — [goo.gl/AxLPii](http://goo.gl/AxLPii) На думку авторів збірника, для початку важливі такі питання.

1. За адресою [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua) має з'явитися сторінка з вибором наразі доступних «змагань» (в т. ч. тренувань). При кліку на конкретне тренування, з'являється пропозиція зайти (залогінитися).
  - (a) Якщо у Вас вже є акаунт на [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua), спробуйте скористатися ним (акаунт може бути один на різні тренування). Якщо акаунт вже є, але Ви забули пароль — автоматичного відновлення/скидування паролю нема. В окремих виключних випадках можливе ручне скидування (див. також п. 4b), але якщо акаунт не цінний — реєструйте новий.



- (b) Щоб зареєструватися, натисніть «Создать учётную запись», наберіть бажаний login та свій email (реально доступний). Знайдіть у пошті лист (він може потрапити у спам), перейдіть по вказаному лінку, наберіть вибраний Вами login, та пароль, що прийшов у листі. Якщо буде написано «Не зарегистрирован» — клікніть на «Подтвердить регистрацию».
- (c) Натиснувши «Участвовать», Ви потрапите всередину тренування.
2. У більшості тренувань умови задач знаходяться в окремих документах за межами єджаджа. Щоб побачити їх, натисніть лінк «Условия». Коли умови задач відкриваються в окремій вкладці, *не* закривайте вкладку з єджаджем.
3. Щоб здати задачу, потрібно:
- (a) Добитися (наприклад, клікнувши на «Инфо»), щоб на сторінці були присутні маленькі прямокутнички з кодами задач (зазвичай — A, B, C, D).
- (b) Клікнути на потрібний (A, B, ...) прямокутничок.
- (c) Вибрати зі списку коло мітки «Язык» одну з доступних мов програмування.
- (d) Натиснувши кнопку вибору файлу навпроти мітки «Файл», знайдіть збережений на своєму комп'ютері файл з розв'язком. Це мусить бути вихідний (исходный, source, сирцевий) файл, з розширенням .pas, .cpp, .c тощо відповідно до мови програмування.
4. Можна задавати питання щодо незрозумілих моментів (типовий час надходження відповіді для тренувань — від кількох годин до кількох днів).
- (a) Рекомендований спосіб: якщо Ви благополучно увійшли всередину потрібного тренування, задавайте питання через лінк «Отправить вопрос».
- (b) Якщо попередній спосіб недоступний (наприклад, не можете залогінитися в єджадж) — можна писати на email oioick@ukr.net, а також у вже згаданий розділ «Олімпіади» форуму сайту cit.skiro.edu.ua.

**Чому нема Delphi?** Зі «справжнім» Delphi є і проблема ліцензійної чистоти, і складнощі взаємодії перевіряючої системи ejudge, що працює під OS Linux, з Windows-програмами. Як компроміс, у переліку мов ejudge.skiro.edu.ua є варіант «Free Pascal — Delphi mode». Він підключає деякі (не всі) «дельфійські» особливості (integer 32-бітовий; string дозволяє рядки, дов-

ші 255; функції містять спеціальну змінну `Result`; ...). До речі, цей режим можна увімкнути і зсередини своєї програми: написати `{mode delphi}` (замість `{APPTYPE CONSOLE}`) і здати під Free Pascal.

**Чи потрібно доводити правильність алгоритмів?** Залежить від обставин. З одного боку, доведення не вимагаються та не оцінюються. З іншого — не вмюючи доводити, важко оцінювати правильність ідей. Якщо програма видає неправильну відповідь, треба якось приймати рішення, чи шукати й виправляти технічну помилку, чи повністю відкинути цю ідею й шукати іншу. У такому сенсі доведення бувають корисні, навіть якщо робити їх виключно для себе. Приклади доведень можна бачити на стор. 18, 41, 46.

**Що означають записи, подібні до « $O(n^2)$ »?** Деталі можна знайти в Інтернеті чи літературі за назвою *асимптотичні позначення*  $O$ ,  $o$ ,  $\Omega$ ,  $\omega$ ,  $\Theta$ ; « $O$ » та « $o$ » нерідко читають «о» (одним звуком), але правильніше «омікрон»; « $\Omega$ » та « $\omega$ » — «омега» (велика й маленька); « $\Theta$ » — «тета». Формальні означення складнувати, наприклад один з варіантів означення  $O$  такий: числова функція  $f(n)$  є функцією порядку  $O(g(n))$ , коли існує деяка числова функція  $f_1(n)$ , така, що виконуються обидва твердження «для всіх  $n$ ,  $f(n) \leq f_1(n)$ » та « $\lim_{n \rightarrow \infty} \frac{f_1(n)}{g(n)} = c > 0$ , де  $c$  — скінченне число» (яке не зростає при  $n \rightarrow \infty$ )

Простіше (не зовсім точно) пояснення: нехай нас цікавить, наскільки стрімко зростає функція  $2n^2 + 17n + 12\sqrt{n} + 500$  при  $n \rightarrow \infty$ . При дуже великих  $n$ , доданок  $2n^2$  значно перевищує всі інші разом узяті, тож можна приблизно оцінювати всю суму самим лише  $2n^2$ . Підемо ще далі й скажемо, що нас не цікавить, чи  $2n^2$ , чи  $7n^2$ , чи  $\frac{1}{6}n^2$  — головне, що коефіцієнт при  $n^2$  є скінченним строго додатним числом. Оце й показує, що як  $2n^2 + 17n + 12\sqrt{n} + 500$ , так і  $7n^2$ , так і  $\frac{1}{6}n^2 + 100n\sqrt{n} + 12345$  являють собою  $O(n^2)$  та  $\Theta(n^2)$ . Відмінність між  $O$  і  $\Theta$  у тому, що  $O(g(n))$  дозволяє, щоб досліджувана функція була хоч «дуже приблизно рівною»  $g(n)$  (у щойно описаному смислі), хоч значно меншою, а  $\Theta(g(n))$  — лише «дуже приблизно рівна». Наприклад,  $2n\sqrt{n} + 17n$  можна вважати хоч  $O(n\sqrt{n})$ , хоч  $O(n^2)$ , хоч  $\Theta(n\sqrt{n})$ , але не  $\Theta(n^2)$ .

Навіщо все це програмісту? Бо саме так зручно оцінювати залежність часу роботи програми від розміру вхідних даних. Кількість тактів CPU, потрібних для обчислення деякого виразу, залежить від моделі CPU та від того, чи поміщаються дані у кеші, й усе це аналізувати дуже складно. Але можна ігнорувати ці складнощі, заявляючи, що будь-який фрагмент програми, який не містить циклів чи викликів підпрограм, працює за  $\Theta(1)$ . Приклад такого аналізу (для алгоритму сортування вставками) наведено на рис. 1.

Як потім використати отримані асимптотичні оцінки? Порівнювати ефективність алгоритмів (для чого не завжди треба реалізовувати їх, часто досить уявити структуру циклів). А також оцінювати шанси алгоритму поміститись у такі-то обмеження часу при обробці вхідних даних таких-то розмірів.

Тактова частота сучасних комп'ютерів — кілька гігагерц (мільярдів тактів за секунду). Враховуючи неточності у питанні «скільки тактів займають які дії» та ігнорування констант у самих означеннях  $O$  чи  $\Theta$ , ці «кілька мільярдів» треба ще поділити (причому не ясно, на скільки), й у висновку виходить щось дуже приблизне «за секунду встигається десь  $10^7$ – $10^9$  дій». Але навіть настільки приблизні оцінки можуть бути корисні. Наприклад:  $n=10^8$ , складність алгоритму  $\Theta(n^2)$ . Тоді  $(10^8)^2 = 10^{16}$ , час роботи від  $10^{16}/10^9 = 10^7$  сек ( $\approx 4$  місяці) до  $10^{16}/10^7 = 10^9$  сек ( $\approx 30$  років) — безнадійно далеко від того, щоб укластись у пару секунд. Або: при  $n=10^5$  алгоритм складністю  $O(n\sqrt{n})$  ( $10^5 \cdot \sqrt{10^5} \approx 3,2 \cdot 10^7$ )

---

```
for i:=2 to N do begin
  curr := A[i];
  j := i-1;
  while (j>0) and (A[j]>curr) do begin
    A[j+1] := A[j];
    dec(j);
  end;
  A[j+1] := curr;
end;
```

$\left. \begin{array}{l} \Theta(1) \\ O(i), \text{ а також } O(n) \end{array} \right\}$

Сумарно буде  $\Theta(n)$  разів по  $O(n)$ , тобто  $O(n^2)$ . До речі, це  $O(n^2)$  неможливо перетворити у  $\Theta$  від чого б не було: при деяких вхідних даних (наприклад, уже відсортованих) маємо  $\Theta(n)$  разів по  $\Theta(1)$ , тобто  $\Theta(n)$ ; при деяких інших —  $\Theta(n)$  разів по  $\Theta(i)$ , які, враховуючи  $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ , кінець кінцем перетворюються у  $\Theta(n^2)$ .

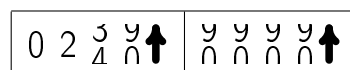
Рис. 1. Приклад проведення асимптотичного аналізу фрагменту програми

має шанси вклатися у секунду, але без «запасу», можуть бути важливі всілякі оптимізації у дрібницях.

Звісно, можна вимірювати час і по-простому в мілісекундах. Автоматичні перевіряючі системи (включаючи ejudge) саме в мілісекундах і вимірюють. Ці способи не заважають один одному, а доповнюють.

Нарешті, у збірнику масово використовуються записи  $O(n \log n)$ ,  $\Theta(\log n)$ , тощо, в яких *не* вказана основа. Це не помилка, а традиція. По-перше, завдяки тотожності  $\log_a n = \log_a b \cdot \log_b n$ , вирази  $O(\log_a n)$  та  $O(\log_b n)$  задають один і той самий клас функцій при довільних константних основах  $a$  та  $b$ , строго більших 1. По-друге, у комп'ютерних науках «стандартною» основою логарифма вважається 2, і  $\log n$  можна трактувати як  $\log_2 n$ .

### Що таке діапазон (цілочисельного) типу?



### Переповнення?

Уявіть лічильник електроенергії (чи води, чи кілометрів — байдуже). Кожен десятковий розряд відображається окремим барабаном, при завершенні значень одного відбувається збільшення наступного на 1... і у деякий момент не вистачає розрядів, і після великого числа одержується 0. Це і є *переповнення (overflow)*. У комп'ютері не барабани (і не буває зображених на рисунку проміжних станів), але теж скінченна кількість розрядів для подання чисел. Тільки розряди двійкові, а не десяткові, тому переповнення настає не після 9999 чи 999999, а після чисел виду  $2^k - 1$  (де  $k$  — кількість бітів), тобто, у двійковому поданні,  $\underbrace{11\dots1}_{k \text{ штук}}$ . Звідси, діапазон беззнакових типів — від 0 до  $2^k - 1$  (тут і далі обидві межі включно). Знакові типи мають діапазон від  $-2^{k-1}$  до  $2^{k-1} - 1$  (кому цікаво, звідки така несиметричність, див. [uk.wikipedia.org/wiki/Доповняльний\\_код](http://uk.wikipedia.org/wiki/Доповняльний_код)).

Наприклад, при спробі обчислити у 16 бітах  $1000_{Dec} \times 100_{Dec} = 100000_{Dec} = 11000011010100000_{Bin}$  («Dec» означає десяткову систему, «Bin» — двійкову) фактично буде отримано лише 16 останніх бітів  $1000011010100000_{Bin}$ , котрі у беззнаковому типі задають число  $34464_{Dec}$ , а у знаковому —  $(-31072_{Dec})$ .

Окремо відзначимо вирази, подібні до  $s := a * b$ , де  $a$  та  $b$  мають вужчий тип,  $s$  — ширший, і результат поміщається у ширший тип, алене у вужчий.

Чи результат буде обчислений правильно у ширшому типі, чи з переповненням у вузкому — залежить від багатьох обставин. Можна вивчати ті обставини, й такі знання бувають корисними. Можна в усіх сумнівних ситуаціях робити *приведення типів (typecasting)*, наприклад `c:=int64(a)*int64(b)`.

Можна знайти таблички з точними діапазонами (напр., [www.freepascal.org/docs-html/ref/refsu5.html](http://www.freepascal.org/docs-html/ref/refsu5.html)). Але до них слід ставитися обережно, бо ці діапазони можуть залежати від архітектури «заліза», операційної системи та компілятора. Скажімо, у [en.cppreference.com/w/cpp/language/types](http://en.cppreference.com/w/cpp/language/types) часто говориться «at least», тобто може бути й більше.

Найсумніше, коли типи з однаковими назвами мають різну розрядність на локальному комп'ютері, де пише учасник, і на сервері, де відбувається перевірка. Зокрема (але не тільки), така проблема виникає, коли учасник локально пише під Delphi чи PascalABC, а здає під Free Pascal, який наразі при відсутності додаткових вказівок вважає `integer` 16-бітовим (директива `{mode delphi}` робить `integer` 32-бітовим; див. також стор. 9).

**Формат з плаваючою комою (floating point) та похибка.** Існує експоненційний формат виведення дробових чисел: наприклад, `1.234e2` означає  $1,234 \cdot 10^2 = 123,4$ ; частина після «e» називається *порядок*; до «e» — *мантиса*. У пам'яті зберігається теж експоненційний формат, але двійковий. Розрядні сітки як у порядку, так і в мантиси обмежені.

Обмеженість порядку критична для дуже великих чи дуже близьких до 0 чисел. Наприклад, тип `double`, реалізований згідно з IEEE 754, не може містити значення, модуль яких більший  $10^{307}$  або менший  $10^{-324}$  (нуль можливий; неможливі значення між 0 і  $10^{-324}$ ). Обмеженість же мантиси призводить до *втрати точності* або *похибки* — наприклад, неможливості розрізнити у типі `double` `1,0000000000000001Dec` від 1 (або `1,0000000000000001 \cdot 10^{-9}` від  $10^{-9}$ , або `1,0000000000000001 \cdot 10^{98}` від  $10^{98}$  — важливий не порядок, а те, що цифри, якими числа відрізняються, не поміщаються у мантису).

В інших типах конкретні діапазони можуть бути іншими; але суть двох обмежень однакова в усіх типах з плаваючою комою.

Проблема похибок загострюється тим, що у тексті програми та вхідних даних числа пишуть у десятковій системі, а фактичне внутрішнє подання двійкове. Наприклад, програма [ideone.com/rnGPE1](http://ideone.com/rnGPE1) показує, що 10-кратне додавання 0.3 до (-3) не дає рівно 0. Адже  $0,3_{Dec}$  стає нескінченним періодичним двійковим дробом  $0,0(1001)_{Bin}$ , обмеженість мантиси призводить до його заокруглення, і похибка з'являється у самій константі.

Дії над числами теж можуть призводити до утворення чи зростання похибки. Сильно збільшують похибку тригонометричні та інші складні функції (але `sqrt` якщо й збільшує, то не сильно; щодо цього виклик `sqrt(x)` набагато кращий за виклик `power(x,0.5)`). А ще можуть сильно збільшити похибку віднімання  $a-b$  при  $a \approx b$  та додавання  $a+b$  при  $a \approx -b$ . Наприклад, для вагів, якими можна зважити людину, похибка  $\pm 10$  г дрібна; але якщо спробувати взнати масу аркуша паперу шляхом того, що хтось зважиться один раз, тримаючи цей аркуш у руках, потім ще раз без цього аркуша, й обчислить різницю вимірювань — результат безнадійно втоне у похибці. Приклад програми, де демонструється аналогічна втрата результату — [ideone.com/bKXT0z](http://ideone.com/bKXT0z) Тому буває важливо провести аналітичні перетворення, щоб отримати математично еквівалентну формулу з меншим впливом похибок.

Інший поширений прийом, який намагається нівелювати похибки — не порівнювати числа з плаваючою точкою «по-простому», а лише з «допуском на похибку», як у таблиці праворуч.

$a = b$	$abs(a-b) < EPS$
$a < b$	$a + EPS < b$
$a \leq b$	$a < b + EPS$
$\vdots$	$\vdots$

«EPS» означає «якесь маленьке додатне число», його треба задати самому (наприклад, `const EPS=1e-6`). Як правильно вибрати EPS — питання складне. Буває легше йти не від того, якої величини можуть сягати похибки, а від того, наскільки близькими можуть бути значення, які треба розрізняти, й брати EPS у кілька разів меншим... Іноді варто переписувати умови так, щоб враховувати не абсолютну, а відносну похибку...

Детальніше про floating point та похибки можна прочитати у багатьох місцях, зокрема [habrahabr.ru/post/112953](http://habrahabr.ru/post/112953)

**Щодо `cin/cout` та `scanf/printf` мовою C++.** Є такий сумний факт, що введення/виведення засобами `istream/ostream` (зокрема, `cin/cout`) працює повільно. Настільки повільно, що, наприклад, на II етапі 2013/14 н. р. у найбільшому тесті задачі «Всюдисущі чісла» (стор. 28–32) саме лише читання `cin`-ом вхідних даних, взагалі без обробки по суті, вже перевищувало обмеження часу. Вихід — читати функцією `scanf`, яка з незвички може здаватися незручною, зате швидко працює. Ще один спосіб зменшення проблеми (лише часткового зменшення, тобто його не завжди достатньо) — виклик `cin.sync_with_stdio(false)`, який трохи пришвидшує читання `cin`-ом за рахунок вимикання синхронізації, тобто після цього не можна читати поперемінно то засобами `cin`, то засобами `scanf`.

А взагалі, є дуже вже багато ситуацій, коли C++ виграє у Pascal; тож окремі випадки, коли неграмотне використання C++ призводить до результатів гірших, ніж Pascal, можна вважати проявом справедливості...

Ще гострішою проблема швидкості читання є у мові Java, але то вже за межами цього посібника. Шукайте самостійно.

## 3 Задачі та їхні розв'язки

### 3.1 Обласна інтернет-олімпіада 2013/14 н. р.

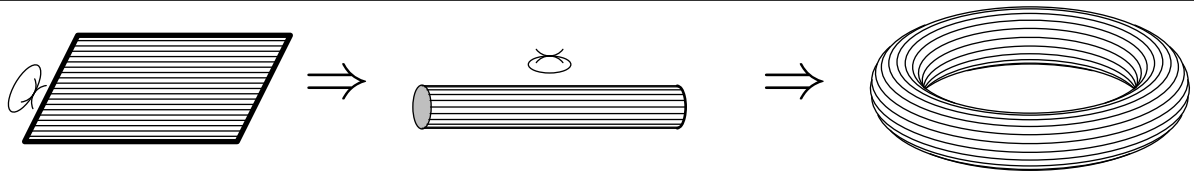
Задачі доступні для дорішування ([ejudge.skiro.edu.ua](http://ejudge.skiro.edu.ua), змагання №10).

#### Задача А. «Тор»

Вхідні дані: Клавіатура (`stdin`)      Обмеження часу: 1 сек

Результати: Екран (`stdout`)      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  $(r_1, c_1)$  тору у точку  $(r_2, c_2)$  тору по «зацикленому» простору тора. Скориставшись відомим принципом незалежності переміщень, можна побачити, що переміщення в горизонтальному і вертикальному напрямках не залежать одне від одного. Розглянемо переміщення як перетин вертикальних і горизонтальних сторін клітинки. Незалежно від траєкторії, необхідно перетнути лише певну конкретну кількість вертикальних ліній (позначимо цю кількість  $\Delta r$ ) і горизонтальних ліній (відповідно  $\Delta c$ ). (Звісно, якщо не розглядати відверто не мінімальні шляхи, де одна й та ж 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  $\Delta c$  слід вчинити аналогічно, потім додати  $\Delta r + \Delta c$ .

## Задача В. «Паркет–1»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 1 сек

Результати: Екран (stdout)    Обмеження пам'яті: 64 мегабайти

Щоб зобразити за допомогою паркету Супер-Креативний Візерунок, треба  $N_1$  дощечок розмірами  $1 \times 1$ ,  $N_2$  дощечок розмірами  $2 \times 1$ ,  $N_3$  розмірами  $3 \times 1$ ,  $N_4$  розмірами  $4 \times 1$  та  $N_5$  дощечок розмірами  $5 \times 1$ . Купити можна лише дощечки розмірами  $5 \times 1$ . Дощечки можна різати, але не можна склеювати. Наприклад, коли потрібні п'ять дощечок  $2 \times 1$ , їх не можна зробити з двох дощечок  $5 \times 1$ , але можна з трьох. Для цього дві з них розріжемо на три частини  $2 \times 1$ ,  $2 \times 1$  та  $1 \times 1$  кожну, а третю — на дві частини  $2 \times 1$  та  $3 \times 1$ . Отримаємо потрібні п'ять дощечок  $2 \times 1$ , а дві дощечки  $1 \times 1$  та одна  $3 \times 1$  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  $5 \times 1$  необхідно купити.

**Вхідні дані.** 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 10000, 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 складна необхідністю дуже акуратно розглядати випадки, але проста тим, що потрібні *лише* розгалуження та присвоєння.

Алгоритм наведено на стор. 19. Мова коду — Python 3; “=” (одинарне) — присвоєння; “==” (подвійне) — перевірити, чи дорівнює; “%” — залишок від ділення (mod); “//” (подвійне) — цілочисельне ділення (div); “a+=b” — те само, що a=a+b, тобто до a додати b і покласти результат у ту саму змінну a; аналогічно “a-=b”, “a%=b”. Спочатку, змінні n1, n2, n3, n4 та n5 містять потрібні кількості дощечок розмірами 1×1, 2×1, 3×1, 4×1 та 5×1 відповідно. У змінній res будується 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.

Аргументація дій у коментарях зовсім не зайва (див. також стор. 10). Бо задача є частковим випадком *задачі упаковки корзин (bin packing problem)*, для загального вигляду якої не відомо правильного швидкого алгоритма (про простий не йдеться; наука не знає навіть складного, щоб був правильний та швидкий одночасно). Наприклад, *якби* розміри початкових дощечок були не 5×1, а 7×1, і треба було сформувати 2 шт. 3×1 і 4 шт. 2×1, то треба було б узяти дві стандартні 7×1 і розрізати кожну на (3×1) + (2×1) + (2×1), діючи *всупереч* ідеї «перш за все вибирати якнайбільші розміри».

Посилити аргументацію (провести більш строге доведення цих коментарів) можна приблизно так. На усіх кроках (3), (5)–(8) справедливо «немає смислу викидати у відходи те, що можна використати; навіть якщо виявиться, що його можна узяти й пізніше, ми нічого не втрачаємо, узявши раніше». На кроках (5),

Фрагмент коду	Коментар
<code>res = n5</code>	(1) На кожну дощечку $5 \times 1$ неминуче потрібна окрема ціла дощечка.
<code>res += n4</code>	(2) На кожну $4 \times 1$ теж потрібна окрема...
<code>n1 -= min(n1, n4)</code>	(3) ...і кожен обрізок можна взяти як дощечку $1 \times 1$ . Тому <i>подальша</i> потреба в дощечках $1 \times 1$ складає вже не $n1$ , а або $n1 - n4$ , або 0.
<code>res += n3</code>	(4) На кожну дощечку розмірами $3 \times 1$ теж неминуче потрібна окрема дощечка...
<code>if n2 &gt; n3:</code> <code>    n2 -= n3</code>	(5) ... причому, при $N_2 > N_3$ використовуємо кожен обрізок як дощечку $2 \times 1$ ...
<code>else:</code> <code>    n3 -= n2</code> <code>    n2 = 0</code> <code>    n1 -= min(n3*2, n1)</code>	(6) ... а при $N_2 \leq N_3$ формуємо з обрізків усі дощечки $2 \times 1$ , а з решти ще й $\min(n3*2, n1)$ дощечок $1 \times 1$ (доки не закінчатся дощечки $3 \times 1$ або потреба у дощечках $1 \times 1$ ).
<code>res += n2//2</code> <code>n1 -= min(n1, n2//2)</code> <code>n2 %= 2</code>	(7) Оскільки всі дощечки $3 \times 1$ вже сформовані, тепер на кожну пару дощечок $2 \times 1$ потрібна окрема дощечка, причому обрізок можна використати як дощечку $1 \times 1$ .
<code>if n2==1:</code> <code>    res += 1</code> <code>    n1 -= min(n1, 3)</code> <code>    n2 = 0</code>	(8) Якщо на попередньому кроці кількість дощечок $2 \times 1$ була непарна, то зараз треба сформувати останню дощечку $2 \times 1$ , причому з обрізку можна зробити до 3 дощечок $1 \times 1$ .
<code>res += n1//5</code> <code>n1 %= 5</code>	(9) Якщо після усіх попередніх кроків усе ще є потреба в дощечках $1 \times 1$ , формуємо їх, розрізаючи кожну дощечку на 5 частин.
<code>if n1 &gt; 0:</code> <code>    res += 1</code>	(10) Якщо все ще є потреба у $1 \times 1$ , вона $\leq 4$ штук, достатньо ще <i>однієї</i> дощечки.

Рис. 2. Алгоритм розв'язання задачі «Паркет»

(6), (8) — «де можна, варто віддавати перевагу  $2 \times 1$  над  $1 \times 1$ , бо замість кожної  $2 \times 1$  завжди можна зробити хоч дві  $1 \times 1$ , хоч одну, а зробити  $2 \times 1$  замість двох  $1 \times 1$  можна далеко не завжди». На кроках (9), (10), формується залишок тих дощечок  $1 \times 1$ , які ніяк не могли бути сформовані раніше, бо, станом на початок кроку (9),  $n_1 > 0$  лише якщо на усіх попередніх кроках усі дощечки використовувалися геть без відходів.

А для більших розмірів стандартної дощечки десь колись настає момент, коли узагальнення таких міркувань перестають бути правильними...

## Задача С. «Сума квадратів»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 1 сек

Результати: Екран (stdout)    Обмеження пам'яті: 64 мегабайти

Для заданого натурального числа  $N$ , визначити, скількома різними способами можна розкласти його в суму двох точних додатних квадратів.

Іншими словами: для заданого  $N$ , з'ясувати, скільки є різних способів подати його як  $N = x^2 + y^2$ , причому  $x$  та  $y$  являють собою цілі строго додатні числа, а розкладення, в яких значення  $x$  та  $y$  лише обміняні місцями, вважаються однаковими.

**Вхідні дані.** Вхідні дані — натуральне число  $N$  — слід прочитати зі стандартного входу (клавіатури).

**Результати.** Результат — знайдену кількість способів — слід вивести на стандартний вихід (екран).

### Приклади:

Вхідні дані	Результати	Примітка
16	0	Розкласти у суму <i>додатних</i> точних квадратів неможливо
10	1	Єдине розкладення $10 = 1^2 + 3^2$
4225	4	Чотири різні розкладення: $4225 = 16^2 + 63^2 = 25^2 + 60^2 = 33^2 + 56^2 = 39^2 + 52^2$

100 балів (з 250) припадатиме на тести, в яких  $1 \leq N \leq 1234$ .

Ще 50 балів — на тести, в яких  $12345 \leq N \leq 123456$ .

Решта 100 балів — на тести, в яких  $12345678 \leq N \leq 123456789$ .

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

**Розбір задачі.** Задача передбачає *перебір* — проби різних значень, з перевіркою, чи виконується рівність. Треба лише організувати це правильно.

Один з класичних способів уникати подвійних урахувань розкладень, у яких значення  $x$  та  $y$  лише обміняні місцями — враховувати лише ті, де  $x \leq y$ .

Деякі учасники здавали реалізацію, наведену праворуч. Вона неправильна: і працює надто довго, і може знаходити зайві розкладення. На стор. 11 роз'яснено, чому такий ( $O(N^2)$ ) алгоритм очевидно не вкладеться у 1 сек. А на стор. 12 — що таке переповнення, внаслідок яких  $x*x+y*y=N$  може виявитися true при, наприклад,  $x=1$ ,  $y=65537$ ,  $N=131074$ .

Через помилку автора задачі, задача вийшла простішою, ніж планувалося. Повний бал набирає в т. ч. й «вилізана» у деталях програма, де принципова оптимізація лише одна: верхні межі циклів змінені з  $N$  на  $\sqrt{N}$ .

```
res:=0;
for x:=1 to N do
  for y:=1 to N do
    if x<=y then
      if x*x+y*y=N then
        res:=res+1;
```

```
res:=0;
sqrtN:=round(sqrt(N));
for x:=1 to sqrtN do
  for y:=x to sqrtN do
    if x*x+y*y=N then
      res:=res+1;
```

Все ж розглянемо можливу подальшу оптимізацію, яка зменшує складність з  $O(\sqrt{N} \times \sqrt{N}) = O(N)$  до  $O(\sqrt{N})$ . Коли  $x$  вибраний, рівність  $x^2 + y^2 = N$  не може виконатися для різних натуральних  $y$ . Тож вкладеного цикла по  $y$  можна позбутися, замінивши на перевірку, чи  $\sqrt{N-x^2}$  ціле та  $\geq x$ .

Перевірку, чи деяке  $\sqrt{K}$  ціле, можна робити як  $\text{frac}(\text{sqrt}(K))=0$ . Або як  $\text{sqr}(\text{round}(\text{sqrt}(K)))=K$ . Теоретично, у 1-му виразі можливий підвох, якщо  $\text{sqrt}$  поверне значення з похибкою і  $\text{frac}$  верне не 0, де насправді 0, а у другому — якщо  $\text{round}$  матиме вужчий тип і виникне переповнення. Практично вони обидва працюють правильно.

**Задача D. «Дільники»**

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 2 сек

Результати: Екран (stdout)    Обмеження пам'яті: 64 мегабайти

Для натурального числа  $N$ , виведіть у порядку зростання всі його різні натуральні дільники.

**Вхідні дані.** Вхідні дані слід прочитати зі стандартного входу (клавіатури). Це буде єдине натуральне число  $N$ .  $1 \leq N \leq 1234567891011$ .

**Результати.** Результат — послідовність усіх різних натуральних дільників, у порядку зростання — слід вивести на стандартний вихід (екран). Виводити обов'язково в один рядок, розділяючи пробілами.

**Приклади:**

Вхідні дані	Результати
9	1 3 9
120	1 2 3 4 5 6 8 10 12 15 20 24 30 40 60 120

120 балів (з 250) припадатиме на тести, в яких  $1 \leq N \leq 4321$ .

Решта 130 балів — на тести, в яких  $12345678 \leq N \leq 1234567891011$ .

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

**Розбір задачі.** Перш за все, слід розуміти, що основна проблема алгоритма `for i:=1 to n do if n mod i = 0 then write(i, ' ')` — він не має ніяких шансів устигнути за потрібний час (див. також стор. 11).

Є сенс перебирати дільники лише до  $\sqrt{N}$ . Дільники, більші за  $\sqrt{N}$ , можна обчислити діленням  $N$  на якийсь із дільників, менших  $\sqrt{N}$ . Адже: (1) якщо  $a$  — дільник  $N$ , то  $N/a$  — ціле, й також дільник  $N$ ; (2) числа  $a$  та  $N/a$  не можуть одночасно бути більшими  $\sqrt{N}$ .

Можна один раз прокрутити цикл від 1 до  $\sqrt{N}$ , і повиводити всі знайдені дільники, а потім наступний (не вкладений, а наступний) цикл від  $\sqrt{N}$  downto 1 і повиводити  $N \operatorname{div} i$ . Такий алгоритм матиме складність  $O(\sqrt{N})$ , що очевидно поміститься у обмеження часу ( $\sqrt{1234567891011} \approx 1,1 \cdot 10^6$ ). Можна трохи оптимізувати програму, якщо у першому циклі не лише виводити дільники, а ще й

запам'ятовувати їх у масив, щоб другий цикл перебирав лише дільники, а не всі числа проміжку. Але така оптимізація *не* принципова.

Незалежно від способу організації другого циклу, варто перевірити, чи правильно програма розбирається з такими випадками: (а)  $\sqrt{N}$  цілий і є одним (а не двома) з дільників, як 6 для 36; (б)  $\sqrt{N}$  не цілий, але є два дільники близько до  $\sqrt{N}$ , як 6 і 7 для 42; (в) дільників, близьких до  $\sqrt{N}$ , нема.

Приклад розв'язку — [ideone.com/wY3IY0](http://ideone.com/wY3IY0) Варто відзначити такі його моменти:  $N$  мусить бути 64-бітовим, але решту величин зручніше лишити 32-бітовими; для масиву `dividers`, розмір мільйон узятий з величезним запасом, насправді досить 7000; але правильно оцінити цю кількість дуже складно, тож якщо не знати, то краще взяти з запасом (але враховуючи обмеження пам'яті).

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

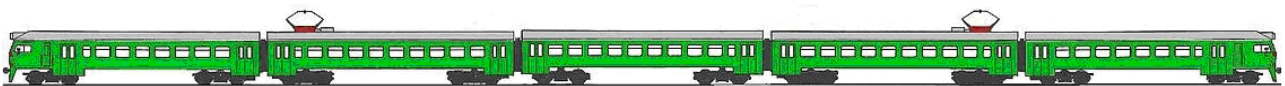
Задачі доступні для дорішування ([ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua), змагання №14).

#### Задача А. «Електричка»

Вхідні дані: Клавіатура (stdin)      Обмеження часу: 1 сек

Результати: Екран (stdout)      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.



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 ясно, 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 неможливо»); але 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](http://ideone.com/vIUkUt) Програма містить виконання всього кількох дій, тому має складність  $\Theta(1)$  і виконується 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)    Обмеження часу: 1 сек

Результати: Екран (stdout)    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 mod 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, щоб воно не втратилося від «відсікань»); це може бути забезпечено або присвоєнням у додаткову змінну, або передачею у функцію параметром-значенням (у Pascal — без модифікатора `var`). Приклад реалізації — [ideone.com/YWcv5B](http://ideone.com/YWcv5B)

Іншим способом є перетворення числа у рядкову величину й подальші звернення до окремих символів-цифр. Деталі сильно залежать від конкретної мови програмування, навіть конкретних бібліотек, тому їх важко пояснити теоретично. Дивіться конкретні розв'язки: [ideone.com/5poAvj](http://ideone.com/5poAvj) (мовою C++) та [ideone.com/d2F0rb](http://ideone.com/d2F0rb) (мовою Pascal із функціями `IntToStr` та `StrToInt` — найбільш лаконічний з усіх наведених).

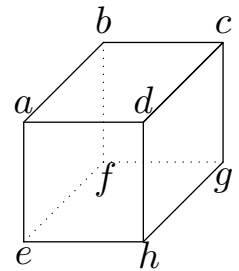
Протягом перетворень значення числа може перевищити 32767, тому треба забезпечити, щоб тип мав хоча б 32 біти (див. також стор. 12 та стор. 9).

## Задача С. «Логічний куб»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 1 сек

Результати: Екран (stdout)    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 неефективний спосіб, вартий уваги лише коли нема кращих ідей. За посиланням [ideone.com/jjq2Pq](http://ideone.com/jjq2Pq) можна побачити трохи модифікований варіант програми, яку здав один з учасників. Він набирає 270 балів з 300, що з одного боку немало, але з іншого — на ручний розгляд випадків явно пішло

багато часу, бали все одно не повні, а шукати помилку в *такому* нагромадженні — досить невдячна справа.

Щоб уникнути такої ситуації, краще реалізувати осмислений алгоритм, який обробляє різні вхідні дані більш-менш однотипно.

Коли мова йде про мінімальні (за кількістю переходів) 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 вершини куба, значення яких 1 (по яким можна проходити), а ребра — ті ребра куба, що поєднують вершини зі значеннями 1.

Причому, раз питають не відстань (як число), а шлях (як послідовність вершин), 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.

Інший можливий спосіб — повний перебір (наприклад, рекурсивний) усіх можливих шляхів, що не містять повторень вершин. Приклад такого розв'язку — [ideone.com/pihZzm](http://ideone.com/pihZzm) Детальніше про цей підхід реалізації перебору можна знайти в Інтернеті або літературі за назвами *пошук з поверненнями*, *бектрекінг* (рос. *поиск с возвратом*, англ. *backtracking*). Якщо говорити про бектрекінг взагалі, то цим не дуже складним способом суто теоретично можна розв'язати дуже багато задач; але бектрекінг зазвичай працює надто довго (наприклад,  $O(N!)$ ) і не вкладається в обмеження часу. А тут вкладається, бо вершин всього 8.

На жаль, ідея «писати осмислений алгоритм, щоб обробляти різні вхідні дані однотипно» мало придатна до заданої рисунком відповідності ребер куба його вершинам. Важко сформулювати правилом, між якими вершинами є ребро і між якими нема. (Найкраще, що вдалося — заплутане «*Позначки вершин відрізняються або на 4, або на 1, але крім  $d \leftrightarrow e$ , а крім названих є ще  $a \leftrightarrow d$  і  $e \leftrightarrow h$* ».) 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 якось). Але це треба ретельно звіряти з умовою, бо успішна побудова

шляхів у одній частині графа ніяк не перевіряє правильність задання ребер у іншій частині.

**Прості способи набрати частину балів.** Тести цієї задачі такі, що програма, яка, не вирішуючи задачу по суті, завжди виводить “NO”, набирає 90 балів з 300. Можливо, це й несправедливо багато. Але з умови очевидно, що така програма мусила хоч щось та набрати; хто не міг розв’язати правильно — мав би пошукати якісь такі варіанти. Трохи чесніший спосіб — перевіряти, чи є вершини кінцями одного ребра, і якщо так, то виводити ці вершини, а якщо ні, то “NO”. Він набрав 150 балів (рівно половину).

## Задача D. «Всюдисущі чісла»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 3 сек

Результати: Екран (stdout)    Обмеження пам’яті: 64 мегабайти

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

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

**Вхідні дані.** 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.** 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	8 13
6 8 10 13 19	
8 9 13 16 19	
6 8 12 13 15	
3 8 13 17 19	

**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 — для пошуку кожного числа в кожному рядку запускати свій цикл, переглядаючи увесь рядок. Такий розв'язок потребує  $O(N \cdot M^2)$  часу, що може не вкладатися в обмеження. Власне,  $1234^3 \approx 1,88 \cdot 10^9$  простих порівнянь на потужнішому сервері могло б і поміститися у 3 сек. Але в тому й смисл задачі, щоб реалізувати щось ефективніше; на потужнішому сервері або ставили б жорсткіший time limit, або давали вхідні дані ще більших розмірів. На тому сервері, що був у 2013 році, такий розв'язок благополучно отримував свої 120 балів з 300 (якщо вчасно робити break — трохи більше).

**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: щоб 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 половиною, що залишилась. І так далі. Тобто, за 1–2 порівняння можна зменшити діапазон пошуку щонайменше вдвічі, і пошук у масиві розміром 1234 потребує до  $\approx 20$  порівнянь (асимптотично —  $O(\log M)$ ).

А  $N \cdot M$  штук таких пошуків поміщаються в обмеження.

Рекомендується знайти в Інтернеті або літературі додаткові деталі щодо бінпошуку. Бо це такий алгоритм, у якому, навіть добре знаючи загальну ідею, легко помилітись й отримати код, який часто працює правильно, але іноді за-циклюється 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 чітко розрізнити 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 масиви або фрагменти одного масиву, але можуть бути й інші, як-то файли чи зв'язні списки). З них можна легко й швидко сформува-ти 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 відповідь) менший з поточних елементів; (2) зсуваємо по-точну позицію тієї вхідної послідовності (*лише однієї з двох!*), звідки взятий цей менший елемент. Коли одна з послідовностей закінчується, дописуємо у послі-довність-відповідь увесь ще не використаний «хвіст» іншої. Це — *стандартне* злиття, яке робить те саме (але швидше), що дописування однієї з послідов-ностей 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 задачі треба інше (спільні елементи). Але, виявляється, під це легко модифікувати злиття. Треба при порівнянні поточних елементів різних послі-довностей розрізнити три випадки: якщо поточний елемент 1-ої послідовності

строго менший за поточний елемент 2-ої, то зсунути поточну позицію 1-ої (нічого не пишучи у відповідь); якщо строго більший, то зсунути позицію 2-ої (теж не пишучи); якщо поточні елементи рівні, то записати це однакове (спільне) значення у результат та зсунути поточні позиції обох послідовностей. 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, ...) результат попереднього злиття. *Завдяки* тому, що завжди вибираються лише спільні елементи, розмір кожної з послідовностей  $\leq M$ , тому  $N-1$  застосувань злиття займуть сумарний час  $O(N \cdot M)$ . *Якби* робилося стандартне злиття і розміри зростали, оцінка була б значно більшою.

Знайдіть додаткову інформацію про злиття. Її багато, але про злиття часто пишуть як про складову сортування, а тут потрібне *саме злиття*, без рекурсивної надбудови сортування. Крім того, не всі алгоритми, які правильно виконують стандартне злиття, легко модифікуються на вибір лише спільних елементів.

Користувачам C++ можна також ознайомитися з *готовою* потрібною модифікацією злиття — функцією `set_intersection` бібліотеки `algorithm`.

Таким чином, грамотні користувачі C++ в обох наведених способах розв'язання у виграші, бо можуть значну частину потрібного алгоритму не писати самостійно, а використати бібліотечні засоби. Водночас, малодосвідчені користувачі C++ у програші, бо можуть і не знати, як вирішити проблему, що читання `cin`-ом не вкладається у обмеження часу. (Як? Див. стор. 15.)

### **Навіщо в умові згадані групи тестів зі значеннями до 12345?**

Щоб дати можливість набрати ще  $\approx 20\%$  балів тим, хто не додумався ні до одного з правильних способів, але знає наступний, складністю  $\Theta(N \cdot M + V)$ , де  $V$  — розмір діапазону від мінімуму до максимуму можливих значень.

Мається на увазі, що програму в цілому можна організувати 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 не пройде, але, можливо, набере більше балів.

Так от, для малих значень ефективний такий підхід. Зведемо масив, *індексами* якого будуть *значення* зі вхідних даних, щоб щоразу, прочитавши деяке  $v$ , збільшувати  $\text{num}[v]$  на 1 (спочатку всі  $\text{num}[\cdot]$  ініціалізуються нулями). Таким чином, після обробки усіх вхідних даних кожне значення  $\text{num}[v]$  означатиме, скільки разів зустрілося число  $v$ ; оскільки у кожному окремо взятому рядку всі числа різні, то  $\text{num}[v] = N$  рівносильно « $v$  зустрілося в усіх рядках».

### 3.3 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 цьому збірнику. Цей тур позиціонувався одночасно і як змагальний, і як кваліфікаційний; тому рівень складності комплекта задач дещо нижчий, ніж зазвичай на III етапі.

Задачі доступні для дорішування ([ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua), змагання №15).

2-й тур відбувався у Черкасах, але на задачах інших авторів, іншій системі ([ejudge](http://ejudge), але не [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua)), та й про його дорішування авторам цього збірника нічого не відомо. Тому, він до збірника не включений.

## Задача А. «ISBN»

Вхідні дані: Клавіатура (stdin)      Обмеження часу: 1 сек

Результати: Екран (stdout)      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 (бажаючі можуть знайти, що таке *кільце залишків за модулем* та *мала теорема Ферма*, і застосувати до цієї задачі). Все це було би доцільним, якби довжина ISBN-коду становила, наприклад, сотні тисяч, так що розглянутий далі простіший і значно більш «програмістський» (а не «математичний») підхід працював би надто довго.

А для 10 цифр підходить і значно простіший перебір, тобто перепробувати усі варіанти від 0 до 9 і для кожного подивитися, чи виконується описана в умові правильність ISBN. Приклад реалізації див. [ideone.com/g3SbMb](http://ideone.com/g3SbMb) Правда, при різних правильних відповідях ця програма вивела б усі, хоча на олімпіадах треба виводити будь-яку одну. Але насправді це неактуально, бо різні правильні відповіді неможливі (хто вивчав питання, згадані у попередньому абзаці, можуть це довести; решта можуть або повірити, або дописати у розв'язок break для обривання циклу після виведення першої відповіді.)

Говорити про асимптотичну складність некоректно, бо нема того розміру вхідних даних, який міг би прямувати до  $\infty$ . Якби довжина ISBN-коду становила довільне  $N$  (але таке, щоб  $N+1$  було простим, і щоб аналогічна сума добутоків

ділилася без остачі на  $N+1$ ), можна було би сказати, що перебір має складність  $\Theta(N^2)$ , а теоретико-числовий розв'язок —  $\Theta(N + \log N) = \Theta(N)$ .

Ще може бути проблемою формат вхідних даних, особливо для мало-досвідчених користувачів C++: `cin>>a>>b` ніби читає в `a` те, що до пропуску, й у `b` те, що після; але в обох випадках «пробіл замінює найпершу цифру» та «... найостаннішу ...» всі 9 осмислених цифр ідуть в `a`. Може сп'ясти на думку читати у циклі окремі `char`-и; але при стандартних налаштуваннях `cin>>s` (`s` — типу `char`) взагалі пропускає пропуски, і взнати його позицію неможливо. Один зі способів вирішення цієї проблеми — перед читанням `char`-ів викликати `cin.unsetf(ios::skipws)`, щоб пропуски та переведення рядка таки читались у ті `char`-и. Інший — використати функцію `getline(cin, s)` (`s` — типу `string`); вона (як і `readln` мови Pascal) читає все, включаючи пробіли, до кінця рядка.

## Задача В. «Точні квадрати»

Вхідні дані: Клавіатура (`stdin`)    Обмеження часу: 1 сек

Результати: Екран (`stdout`)    Обмеження пам'яті: 64 мегабайти

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

1. 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).
2. 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 чісла і перевірити кожне на відповідність умовам. Причому, «чи є точним квадратом» вже розглядали на стор. 21, суму цифр — на стор. 25. . . Але програма [ideone.com/gLqsYd](http://ideone.com/gLqsYd) набирає лише 50 балів зі 100.

Проблема у тому, що перевіряти аж 2 млрд чисел — забагато. Навіть якщо (абсолютно слушно) рахувати суму цифр лише для тих, які пройшли перевірку «чи є точним квадратом». Пришвидшити розв'язок досить просто — *генерувати лише точні квадрати*, а не перебирати й перевіряти геть усі чісла проміжку. Достатньо запускати цикл не від  $a$  до  $b$ , а від  $\approx\sqrt{a}$  до  $\approx\sqrt{b}$  й працювати з  $i^2$ . Правда, тут можна заплутатися, як перетворити « $\approx\sqrt{a}$ » та « $\approx\sqrt{b}$ » у точні цілі значення. Не дуже красивий, зате точно 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](http://ideone.com/Ep1T1L)

З асимптотичною оцінкою цього розв'язку є неоднозначність. Попередній абзац наштовхує на  $\Theta(\sqrt{b} - \sqrt{a})$  або  $O(\sqrt{b})$ ; але якщо враховувати ще цикл у `sumOfDigits`, то вийде  $O(\sqrt{b} \cdot \log b)$ .

## Задача С. «Ложбан»

Вхідні дані: Клавіатура (stdin)      Обмеження часу: 1 сек

Результати: Екран (stdout)      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 *pa*ресі.

**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
repоравo	2014

**Розбір задачі.** По-перше, слід знайти в умові справді потрібну частину: *«Є текст, який гарантовано отриманий таким чином: взяли число  $\leq 1\,000\,000$ , й замінили кожен цифру на дві букви згідно з наведеною табличкою. Провести зворотнє перетворення цього тексту у число.»*

По-друге, все могло би бути вельми складнім, якби траплялися ситуації, коли одне зі слів — початок іншого (як-то «7 позначається як *mis*, 8 — як *misiv*»). Тому варто 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.

Задача насправді досить проста. Треба лише зуміти прочитати це у громіздкій умові. Наприклад, див. [ideone.com/BNuD1L](http://ideone.com/BNuD1L). Або, використавши, що вхідні дані *гарантовано* являють собою якесь закодоване число, можна написати щось іще простіше — наприклад, див. [ideone.com/0dw21F](http://ideone.com/0dw21F)

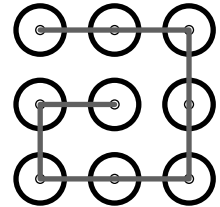
Така умова задачі в принципі могла б містити підвох: ніде не сказано прямо, чи числа натуральні. А раптом від'ємні? А раптом взагалі десяткові дробі? А якщо від'ємні, то скільки може бути цифр, адже  $(-10^{100500}) < 10^6$ ? За правилами переважної більшості олімпіад з інформатики, учасник має право задати питання журі щодо таких моментів умови. Або, враховуючи можливість багатократної задачі, пробувати варіанти... Конкретно в цій задачі цього туру підвоху не було (числа були цілі невід'ємні).

## Задача D. «Графічний пароль»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 4 сек

Результати: Екран (stdout)    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  $3 \times 3$  (не обов'язково через всі).



Але згодом Василь зрозумів, що пароль на сітці  $3 \times 3$  не є досить безпечним, і вирішив виправити цю проблему. Для цього він збільшив розмір сітки до  $5000 \times 5000$ , 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 накладатися 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	0
1 1	
0 1	
0 0	
2 0	
2 2	
0 2	

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

**Розбір задачі.** Головне при розв'язуванні цієї задачі — не перестаратися й не почати писати «чесні» перевірки перетинів. Такі перевірки могли б бути доречними при більших розмірах сітки та меншій кількості відрізків ламаної.

А при заданій кількості, відомі автору цього розбору оптимізації перебору всіх можливих перетинів не вкладаються у обмеження часу.

У цій конкретній задачі краще помітити такі факти:

1. 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  $5000 \times 5000$  вершин сітки може бути задіяна щонайбільше двічі, тобто сума довжин усіх відрізків не перевищує 50 млн.
2.  $5000 \times 5000 = 25$  млн — дуже 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». (Але, щоб укластися в обмеження пам'яті, вони мусять бути 1- чи 2-байтовими!)

Завдяки цьому, виявляється допустимим такий простий підхід, як *«завести масив  $5000 \times 5000$ , ініціалізувавши всі елементи нулями; ходити уздовж ліній, збільшуючи на 1 значення у комірках, відповідних пройденим вершинам сітки; відповіддю буде кількість комірок зі значенням 2»*. Великий time limit 4 сек потрібен не для роботи алгоритму, а для читання величезних вхідних даних. Приклад реалізації такого алгоритма — [ideone.com/S1R8oV](http://ideone.com/S1R8oV)

### 3.4 Обласна інтернет-олімпіада 2014/15 н. р.

Задачі доступні для дорішування ([ejudge.skiro.edu.ua](http://ejudge.skiro.edu.ua), змагання №18).

#### Задача А. «Три круги»

Вхідні дані: Клавіатура (stdin)      Обмеження часу: 1 сек

Результати: Екран (stdout)      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:

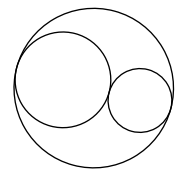
- “1st” (без лапок), якщо 2-й і 3-й круги можна покласти поверх 1-го;
- “2nd” (без лапок), якщо 1-й і 3-й круги можна покласти поверх 2-го;
- “3rd” (без лапок), якщо 1-й і 2-й круги можна покласти поверх 3-го.

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, коли торкаються. Тому гранична ситуація, коли при хоч трохи меншому найбільшому диску вони вже звисають, а при рівно такому або більшому все гаразд, зображена 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. Реалізацію див. [ideone.com/rx20dn](http://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 деякий аналог другого підходу виявляється набагато кращим за аналіз випадків, аналогічний першому підходу.

## Задача В. «Перевезення вантажу»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 1 сек

Результати: Екран (stdout)    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 (див. також стор. 10), що це дасть максимальну кількість вагонів. *«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, а залишок вантажопідйомності корабля зменшило сильніше. Отже, ніякого виграшу від того, щоб брати вагон з немінімальною кількістю коробок, нема».* Ця схема доведень типова, тобто її аналоги придатні у багатьох ситуаціях.

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 по порядку. Особливо зручно, якщо писати мовою, де є готове бібліотечне сортування (наприклад, у [ideone.com/sElRrj](http://ideone.com/sElRrj) використано функцію `sort` бібліотеки `algorithm` мови C++). Та й якщо писати сортування самому, все ж є деяка зручність у тому, щоб окремо написати та ретельно вивірити правильність 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 його складність  $\Theta(N \cdot P)$ , тобто, враховуючи  $P \leq N$ ,  $O(N^2)$ . А складність ефективних сортувань  $O(N \log N)$ , що набагато менше.

## Задача С. «Коло і точки»

Вхідні дані: Клавіатура (stdin)    Обмеження часу: 1 сек

Результати: Екран (stdout)    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.** Коло — множина точок, рівновіддалених від центру, тож досліджувана точка потрапляє всередину кола, коли відстань між нею і центром кола менша за радіус, на саме коло — коли рівна, і назовні, коли більша. Ця відстань рівна  $\sqrt{(x_i - x_C)^2 + (y_i - y_C)^2}$  (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 зі вкладеними розгалуженнями на три випадки. При бажанні, можна позбутися похибок (див. стор. 13–14), прибравши корені, тобто порівнювати  $(x_i - x_C)^2 + (y_i - y_C)^2$  з  $R^2$ . Остаточна реалізація — [ideone.com/k9iLan](http://ideone.com/k9iLan)

Розібратися, що робити з 2-им способом подання кола — значно складніше, можливо навіть складніше усієї решти  $2^{1/2}$  задач цього туру. Можна звести 2-ий випадок до 1-го, тобто перейти від трьох точок до центру й радіусу проведеного через ці три точки кола, та використати вже розглянутий розв'язок.

Один зі способів — будувати перетин серединних перпендикулярів двох сторін  $\triangle ABC$ , тобто ті ж дії, що й на уроці геометрії, але виконані замість циркуля і лінійки засобами *обчислювальної геометрії* (рос. *вычислительная геометрия*, англ. *computational geometry*). Вступ до обч. геометрії можна знайти у багатьох місцях, зокрема <https://goo.gl/6уррjу> Програма, що розв'язує задачу цим способом — [ideone.com/k1aFcF](http://ideone.com/k1aFcF) (але звідти свідомо прибрані допоміжні функції, пояснені за попереднім посиланням).

Інший спосіб — вивести на папері прямі формули, 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 (до них треба ставитися як до відомих значень), а  $x$  та  $y$  — координати центра кола, їх-то й шукаємо. Розв'язати цю систему легше, ніж може здатися, бо після перетворень, аналогічних  $(x-x_A)^2 = x^2 - 2x_A \cdot x + x_A^2$ , можна позводити  $x^2$  та  $y^2$ , і рівняння стають лінійними відносно  $x$  та  $y$  (що логічно, бо це серединні перпендикуляри).

Можна й не виражати 2-ий випадок через 1-ий.

Виявляється, відповідь на задачу можна визнавати за знаком детермінанта, наведеного праворуч. (*Детермінант*, він же *визначник* — стандартний термін теорії матриць. Як обчислювати детермінант, та чому цей

$$\begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_i & y_i & x_i^2 + y_i^2 & 1 \end{vmatrix}.$$

детермінант має описану далі властивість, бажачі можуть знайти в Інтернеті або літературі самостійно.)

Отже: рівність цього детермінанта 0 означає, що  $(x_i; y_i)$  лежить на колі, проведеному через  $(x_A; y_A)$ ,  $(x_B; y_B)$ ,  $(x_C; y_C)$ ; додатне значення детермінанта — що всередині; від'ємне — ззовні. Тільки це якщо обхід  $\triangle ABC$  (саме у порядку  $A, B, C$ ) відбувається у додатному напрямку (проти годинникової стрілки), а при зміні напрямку обходу  $\triangle ABC$  слід обміняти місцями смисл додатного і від'ємного знаків детермінанту. Зовсім не очевидно, але ж це *лише один* із способів розв'язання задачі...

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

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №46).

В усіх задачах даного змагання, програма може читати вхідні дані хоч з клавіатури, хоч зі вхідного файлу `input.txt` (але лише з чогось одного, а не поперемінно). Аналогічно, програма може виводити результати хоч на екран, хоч у вихідний текстовий файл `output.txt` (теж лише на/у щось одне).

## Задача А. «Цифра»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`      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 розміри малі (кількість цифр  $\leq 5$ , видаляється одна), найпростіший правильний розв'язок — перебрати всі варіанти викреслення однієї цифри (усе число без 1-ої, усе без 2-ої, тощо), і вибрати з них максимальний. Зручно (хоча й не обов'язково) перевести число у рядок (`string`), і займатися вилученням символів у рядковому поданні. Реалізацію див. [ideone.com/jFBIM6](http://ideone.com/jFBIM6) При бажанні, її можна спростити, роблячи взагалі все виключно рядками. (Взагалі буває проблема «у числах  $7 < 10$ , а у рядках "7" > "10"», але тут вона не проявиться, бо кількості цифр усіх потрібних чисел однакові.)

Ще є правильний 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 97752) — викреслити останню цифру».

Якщо мати лише мету розв'язати цю задачу при цих обмеженнях — краще обмежитися першим правильним способом і не читати далі. А якщо мати бажання ще раз розглянути, як можна доводити правильність алгоритма (див. також стор. 10) — тоді подальший текст важливий.

Занумеруємо цифри початкового числа зліва направо  $\overline{a_1 a_2 \dots a_n}$ . Розглянемо спочатку випадок «нема жодного місця, щоб після меншої цифри йшла більша», тобто  $a_1 \geq a_2 \geq \dots \geq a_n$ . Згідно алгоритму, треба викреслити  $a_n$ , лишивши  $\overline{a_1 a_2 \dots a_{n-1}}$ . Якщо всупереч алгоритму викреслити деяку  $a_j$  ( $1 \leq j < n$ ), вийде  $\overline{a_1 a_2 \dots a_{j-1} a_{j+1} \dots a_n}$ . Початок  $\overline{a_1 a_2 \dots a_{j-1}}$  спільний, тож при порівнянні буде пропущений, фактично будуть порівнюватися частини, де  $a_{j+1}$  замість  $a_j$ ;  $a_{j+2}$  замість  $a_{j+1}$ ; ...;  $a_n$  замість  $a_{n-1}$ . Оскільки розглядаємо ситуацію  $a_1 \geq a_2 \geq \dots \geq a_n$ , то або  $a_j > a_{j+1}$ , або  $a_j = a_{j+1}$ . При  $a_j > a_{j+1}$ , число, отримане всупереч алгоритму, менше (гірше) отриманого згідно з алгоритмом, бо після спільного початку йде  $a_{j+1} < a_j$ . Якщо ж  $a_j = a_{j+1}$ , то можна приєднати цю цифру до спільного початку і повторити всі міркування для « $a_{j+2}$  замість  $a_{j+1}$ ». І так далі. Кінець кінцем, або десь отримаємо, що число всупереч алгоритму менше (гірше) числа згідно з алгоритмом, або дійдемо до  $a_j = a_{j+1} = \dots = a_n$ , тобто викреслення  $a_j$  призводить до того ж результату, що алгоритм.

Лишилося розглянути випадок, коли місце, де  $a_i < a_{i+1}$ , існує. Нехай  $i^*$  —

найлівіша з таких позицій, тобто  $a_1 \geq a_2 \geq \dots \geq a_{i^*}$  і  $(a_{i^*} < a_{i^*+1})$ . Невигідність видаляти замість  $a_{i^*}$  деяку  $a_j$  при  $1 \leq j < i^*$  доводиться аналогічно міркуванням попереднього абзацу. Лишилося довести неvigідність видаляти  $a_j$  при  $i^* < j \leq n$ , а це зовсім легко: згідно з алгоритмом отримуємо  $\overline{a_1 a_2 \dots a_{i^*-1} a_{i^*+1} \dots a_n}$ , всупереч  $\overline{a_1 a_2 \dots a_{i^*-1} a_{i^*} \dots a_{j-1} a_{j+1} \dots a_n}$ , тобто початок  $\overline{a_1 a_2 \dots a_{i^*-1}}$  спільний, потім  $a_{i^*+1} > a_{i^*}$ . Розглянуті випадки покрили всі можливі ситуації, доведення успішно завершено.

Чи має другий алгоритм переваги над першим? При заданих обмеженнях  $n \leq 99999$  — ні. Якби числа могли бути значно більшими (наприклад, до мільйона цифр; не « $n=1000000$ », це число з 7 цифр, а якби кількість цифр могла сягати мільйона) — тоді виявилось б, що перший алгоритм правильний, але повільний, а другий вкладається у розумні обмеження. Асимптотичні оцінки:  $\Theta(L^2)$  для першого,  $\Theta(L)$  для другого, де  $L$  — кількість цифр числа  $n$ .

## Задача В. «Піраміда»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      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](http://ideone.com/w8Yu4L) набирає половину балів. І тут є де помилитися 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 64-бітового типу. З цим можна боротися, застосовуючи найпростіший прийом *модульної арифметики*: робити додавання  $i^2$  не як `s:=s+sqr(int64(i))`, а як `s:=(s+sqr(int64(i)))mod m`. Реалізація з вчасними «... 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 спосіб № 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 й отримати зовсім інший розв'язок [ideone.com/g6nSKa](http://ideone.com/g6nSKa). Він взагалі не містить циклів (складність  $\Theta(1)$ ), тож працює миттєво.

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  $(6 \cdot m)$ » (а не «... mod  $m$ »), бо такі властивості модульної арифметики: хоча  $(a + b) \bmod p = ((a \bmod p) + (b \bmod p)) \bmod p$  — правильна для всіх  $a, b, p$  тотожність, і так само для множення, але для ділення виявляється не так. Детальніші відомості про модульну арифметику просимо знайти в Інтернеті або літературі самостійно.

Щодо того, як можна вивести формулу  $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$  са-мому під час туру — див., наприклад, <http://dxdy.ru/topic22151.html>. Зві-



сно, 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 (враховуючи, що  $m \leq 10^6$ ,  $n \geq 2 \cdot 10^6$ ) вийде, що проміжок від 1 до  $n$  містить кілька (як мінімум, два, як максимум — сотні мільйонів) проміжків від 1 до  $m$ , від  $m+1$  до  $2m$ , від  $2m+1$  до  $3m$ , і т. д.

Розглянемо очевидні тотожності на проміжках від 1 до  $m$  та від  $m+1$  до  $2m$ . Помітимо, що сума  $m^2+2m$  кратна  $m$  і тому не впливає на остаточну відповідь задачі.

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

Аналогічно не впливають  $m^2+4m$ ,  $m^2+6m$ , 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 проміжку  $(m+1)^2 + (m+2)^2 + \dots + (m+m)^2$  має той самий залишок від ділення на  $m$ , що й сума усього  $1^2 + 2^2 + \dots + m^2$ .

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  $n \bmod m$ ).

Отже, сумарна кількість ітерацій усіх циклів не перевищить  $m + (n \bmod m) < 2m \leq 2 \cdot 10^6$ . Це довше, ніж  $\Theta(1)$  зі «способу № 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-прості»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      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 дільник).

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  $2, 3, \dots, \text{round}(\sqrt{n})$ , і якщо хоча б раз поділилося без остачі — число складене, якщо не поділилося ні разу — просте. Важливо перевіряти до кореня (а не до  $n$  чи  $n/2$ ), бо це набагато менше (див. також стор. 22).

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 алгоритми сортування, доречніші будуть сортування вставками або навіть бульбашкове.

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

1. Створити копію цього числа-аргумента у рядковому (*string*-овому) вигляді;
2. Відсортувати цифри (символи рядка) за неспаданням;
3. Перебрати усі числа з відповідною кількістю цифр, від  $0 \dots 01$  до  $9 \dots 99$ , і для кожного з них:
  - (a) перетворити у рядок (знову як копію, щоб не псувати оригінал);
  - (b) теж відсортувати цифри числа за неспаданням;
  - (c) якщо відсортовані послідовності виявилися різними — значить, поточне число не є перестановкою цифр досліджуваного числа і його слід пропустити, а якщо 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 лишається тільки перевіряти, чи є ко-анаграмічно-простим саме введене число  $n$ , потім  $n+1, n+2, \dots$ . Приклад реалізації — [ideone.com/mGI9hm](http://ideone.com/mGI9hm)

Для цієї реалізації дуже складно і вивести асимптотичну оцінку часу роботи, і використати цю асимптотику для оцінювання конкретного часу роботи в мілісекундах. Наприклад, зовсім не ясно, як оцінити кількість ітерацій самого зовнішнього циклу (скільки чисел  $n, n+1, \dots$  треба перебрати, щоб гарантовано знайти ко-анаграмічно-просте). У перевірці простоти, хоч і видно верхню межу циклу  $\text{round}(\sqrt{n})$ , але ж розумна реалізація обриває цикл після першого знайденого дільника, і середня кількість ітерацій виявляється меншою (а наскільки — не ясно); і так далі.

Програма може працювати швидше, якщо, замість перебору всіх чисел з відповідною кількістю цифр (від  $0 \dots 01$  до  $9 \dots 99$ ), відразу генерувати лише ті, що складаються з потрібних цифр, і перевіряти на простоту лише їх. При обмеженні  $n \leq 9999$  це насправді неважливо, тож утримаємося від опису, як робити це вручну; бажаючі можуть знайти за назвою *генерація перестановок* (*generate permutations*). У мові C++ така генерація є готова (функція `next_permutation` бібліотеки `algorithm`).

## Задача D. «Хмарочоси»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 2 сек

Результати: Або екран, або `output.txt`      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.** 1-й 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).

**Результати.** Програма має вивести  $N$  рядків,  $i$ -ий рядок має містити одне ціле число — кількість інших хмарочосів, які видно з даху хмарочосу №  $i$ .

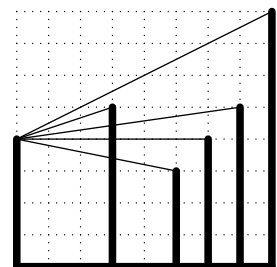
**Приклад:**

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.** Намагання розв'язати задачу через три вкладені цикли (два перебирають поточну пару хмарочосів, всередині них третій перебирає усі проміжні, дивлячись, чи не затуляє він видимість поточної пари) — погана ідея. Такий розв'язок набирає 150–190 балів з 250. Конкретне значення у проміжку 150–190 залежить, головним чином, від того, чи обривається третій цикл негайно, як тільки встановлений результат. Асимптотична складність: якщо 3-ій цикл обривається, то  $O(N^3)$ , якщо ні —  $\Theta(N^3)$ .

Для ефективнішого розв'язку, введемо поняття *нахил* з  $j$ -го хмарочосу на  $k$ -й (на рис. зображені нахили з 1-го хмарочоса на інші). Нахили можна подавати у програмі або кутовими коефіцієнтами  $\frac{y_k - y_j}{x_k - x_j}$ , або векторами  $(x_k - x_j; y_k - y_j)$ .

(Про переваги використання таких векторів у багатьох інших



задачах див. <https://goo.gl/6urrpju> зокрема і за назвою «обчислювальна геометрія» взагалі; але у цій задачі достатньо й кутових коефіцієнтів, якщо подавати їх у якнайточнішому типі `extended (long double)`.)

Розглянемо (припускаючи, що  $N > 1$ ), як порахувати кількість хмарочосів, видимих з 1-го. 2-й хмарочос гарантовано видимий з 1-го. Запам'ятаємо нахил з 1-го хмарочоса 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, ніж 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 цикл, який біжить від поточної позиції 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 вносити правки).

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 одиничці і до кількості видимих з  $j$ -го, і до кількості видимих з  $k$ -го. Втім, таке пришвидшення (удвічі) не принципове. Асимптотика однакова —  $\Theta(N^2)$ .

### 3.6 III (обласний) етап 2014/15 н. р.

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

В усіх задачах цього змагання, програма може читати вхідні дані хоч з клавіатури, хоч зі вхідного файлу `input.txt` (але лише з чогось одного, а не поперемінно). Аналогічно, програма може виводити результати хоч на екран, хоч у вихідний текстовий файл `output.txt` (теж лише на/у щось одне).

**Задача А. «Гірлянда — garland»**

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 64 мегабайти

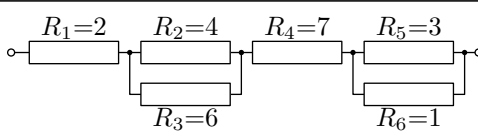
На Новорічні свята Василь разом з батьком вирішили зробити електричну гірлянду із лампочок. При створенні гірлянди лампочки включаються в коло послідовно та деякі паралельно так, що в паралельному з'єднанні може знаходитись лише дві лампочки. Для з'ясування можливості підключення гірлянди до джерела 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:  $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$ .

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 `1.21500000000000E+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 об'єкти 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, пододавати всі  $R[i]$  (уже використані будуть нулями й не вплинуть на результат).

Приклади програм-розв'язків: [ideone.com/QwkKWN](http://ideone.com/QwkKWN) (першим способом), [ideone.com/QNgTua](http://ideone.com/QNgTua) (другим). Асимптотика однакова —  $\Theta(N)$ .

## Задача В. «Прямокутники — rectangles»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 64 мегабайти

Дані два прямокутники, сторони яких паралельні вісям координат.

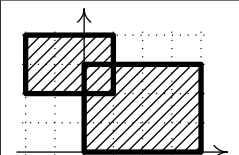
**Завдання.** Напишіть програму `rectangles`, яка б визначала площу їхнього об'єднання, тобто усієї частини площини, що покрита хоча б одним з прямокутників. Площу спільної частини обох прямокутників (якщо така є) слід враховувати один раз.

**Вхідні дані.** Вхідні дані містять два рядки, кожен з яких описує один з прямокутників, у форматі  $x_{\min} \ x_{\max} \ y_{\min} \ y_{\max}$ . Усі координати є цілими числами, що не перевищують по модулю 1000.

**Результати.** Ваша програма має вивести єдине ціле число — знайдену площу об'єднання.

### Приклади:

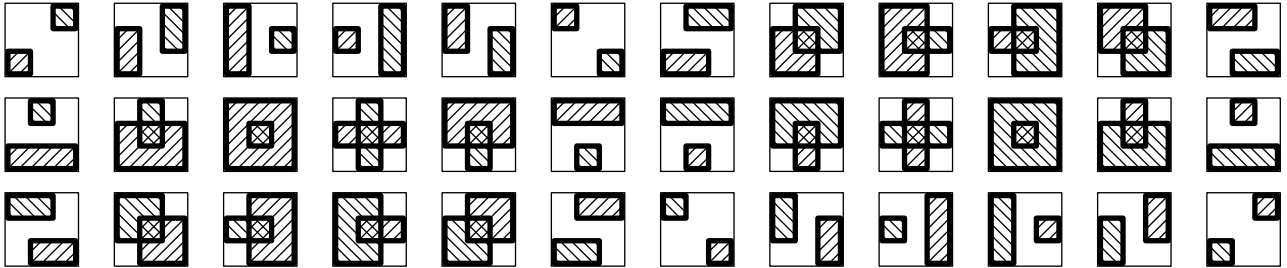
Вхідні дані	Результати
0 10 0 10 20 30 20 30	200
0 20 0 30 10 12 17 23	600

Вхідні дані	Результати	Зображення
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



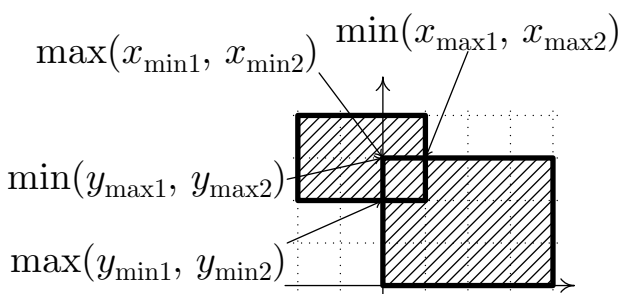
не розглядати випадки рівності деяких із координат (як-то «не розглядати окремо ситуацію  $x_{\min 1} = x_{\max 2}$ , а об'єднати з ситуацією  $x_{\min 1} > x_{\max 2}$ , написавши  $x_{\min 1} > x_{\max 2}$ »), все одно лишаються такі потенційно різні випадки:



Ситуації, коли прямокутники обміняні місцями, наведено як різні, бо якщо виводити для кожного випадку свою формулу залежності результату від вхідних даних, вони можуть бути різними. Звісно, деякі з цих випадків все одно можна об'єднати. Але не так просто зробити це правильно і ніде не помилитися. Тому пропонується розв'язати задачу інакше.

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

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



$$\begin{aligned} \text{максимін } x &= \max(\underbrace{x_{\min 1}}_{=0}, \underbrace{x_{\min 2}}_{=-2}) = 0; \\ \text{мінімакс } x &= \min(\underbrace{x_{\max 1}}_{=4}, \underbrace{x_{\max 2}}_{=1}) = 1; \\ \text{максимін } y &= \max(\underbrace{y_{\min 1}}_{=0}, \underbrace{y_{\min 2}}_{=2}) = 2; \\ \text{мінімакс } y &= \min(\underbrace{y_{\max 1}}_{=3}, \underbrace{y_{\max 2}}_{=4}) = 3. \end{aligned}$$

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

Асимптотична оцінка —  $\Theta(1)$ . Реалізація — [ideone.com/naDNCA](http://ideone.com/naDNCA)

**2-й спосіб (менш універсальний, але конкретно тут теж правильний).** В умові задано невеликі (як для комп'ютера) обмеження на значення координат. Це дає можливість 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, і абсолютно не важливо, чи мають прямокутники непорожній перетин, чи не мають. Ідея частково повторює згадану на стор. 38 (задача «Графічний пароль»), але тут не треба нічого (крім вхідних даних) запам'ятовувати.

Реалізація цього способу: [ideone.com/ВНОтрЕ](http://ideone.com/ВНОтрЕ) Код навіть коротший і простіший, ніж для першого способу. Але: якби дозволялися дробові значення координат, цей спосіб був би принципово неможливим; при цілих координатах, його асимптотична оцінка —  $O(X \cdot Y)$  (де  $X$  та  $Y$  — розміри діапазонів можливих значень координат), що немало.

## Задача С. «Генератор паролів — password»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`      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: `СН15` — 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.

**Приклади:**

Вхідні дані	Результати
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.** Легенда задачі відірвана від життя, ці правила не мають нічого спільного з реальною генерацією паролів. Тим не менш, будемо користуватися термінами «пароль» і «номер паролю», раз вони введені в умові.

**Позбудемось окремих статусів літер і цифр, перейшовши до єдиного алфавіту.** Правила про порядок можна спростити до таких:

- 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 (са́ме в такому порядку);
- всередині кожного пароля (крім 1-символьних), кожен наступний символ мусить бути строго більшим за попередній (згідно алфавіту п. 1).

Правила порівняння різних паролів не міняються (але примітка «літери менші за цифри» тепер слідує з «алфавітного порядку»).

**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 та `if`-а, як у лівому стовпчику рис. 3. А можна, як у правому, замінити перевірку  $i < j$  на задання лише потрібного діапазону.

<pre>for i:=1 to 5 do   for j:=1 to 5 do     if i &lt; j then       writeln(i, ', ', j);</pre>	<pre>for i:=1 to 4 do   for j:=i+1 to 5 do     writeln(i, ', ', j);</pre>
--	---

Рис. 3. Два способи генерації однієї послідовності пар

Для 2-х вкладених циклів це дрібна оптимізація ( $\approx$  удвічі). Але якщо застосувати ту саму ідею до циклів більшої вкладеності, пришвидшення істотне: для п'яти вкладених циклів  $\approx$ сотні разів, для десяти —  $\approx$ мільйони. (Доведення потребує знань комбінаторики, але не дуже глибоких, тож пропонується як вправа. До того ж, писати програму можна й без доведення...)

Приклад такого розв'язку — [ideone.com/Xg6sKT](http://ideone.com/Xg6sKT) Це зовсім не взірць краси 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 взнавати цей пароль символ за символом, зліва направо — на тій підставі, що для кожного можливого початку можна взнавати (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.

! Остаточо, 2015-й пароль має вигляд "CJZ".

Дані пояснення займають багато місця, але це тому, що наведено приклад. Правильна реалізація даного комбінаторного алгоритму працює дуже швидко, вкладаючись у секунду з величезним запасом. 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.

Від вираження асимптотичної оцінки складності алгоритму типовим чином (через  $n$  зі вхідних даних) утримаємось, бо надто багато залежить від розміру алфавіту, а залежність часу роботи від значення  $n$  заплутана. 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.

## Задача D. «Зámok — castle»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу:      D1 — 1 с, D2 — 3 с

Результати: Або екран, або output.txt      Обмеження пам'яті: 128 мегабайтів

Стародавній záмок має прямокутну форму. Záмок 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.**

**Як досить легко набрати частину балів.** Для самої лише підзадачі D1 і часткового випадку «всі кімнати мають прямокутну форму» в якості розв'язку можна запропонувати програму [ideone.com/HHr9a3](http://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 у шир* (він же *пошук у ширину*); рос. *поиск в ширину*, англ. *breadth first search (BFS)*;
2. *Пошук у глиб* (він же *пошук у глибину*); рос. *поиск в глубину*, англ. *depth first search (DFS)*;
3. Різноманітні алгоритми графічної *заливки (flood fill)*.

Усі ці алгоритми неважко знайти в Інтернеті або в літературі самостійно, й усі вони надто громіздкі, щоб пояснювати їх тут. І кожним із них можна реалізувати задачу зі складністю  $\Theta(N \cdot M)$ . Тільки для цього треба:

1. Просто перебирати усі клітинки замку, і щоразу, знайшовши 0, запускати BFS/DFS/заливку, щоб повністю виділити відповідну кімнату, позамінявши її нулі на інші значення, та обчислити її розмір.



2. Забезпечити, щоб кожен такий виклик BFS/DFS/заливки, котрому вказується, починаючи звідки дослідити кімнату, працював за час, пропорційний розміру цієї кімнати, а не усього замку.
- 

Часто вважають, що в такій ситуації найпростішою є рекурсивна реалізація пошуку вглиб. Частково це правда, особливо якщо писати мовою Pascal, де чергу (потрібну для пошуку вшир) треба реалізовувати самому, а механізм рекурсії є готовий. 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 виділяється окремо, її може бути значно менше, й коли *вона* закінчилася, наявність іншої вільної пам'яті рекурсії не допомагає.

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.

До речі, відокремленість стекової пам'яті від інших видів пам'яті пояснює також наступну ситуацію. Буває, що програма благополучно працює з великим глобальним масивом, а по смислу той масив потрібен лише всередині деякої підпрограми (не обов'язково рекурсивної). Природня спроба «сховати» цей масив від решти програми, зробивши його локальним, може призводити до аварійних завершень модифікованої програми, саме через нестачу стекової пам'яті (stack overflow error). Автори збірника ні в якому разі не закликають робити глобальними усі змінні! Змінні, потрібні лише у підпрограмі, *правильно* робити локальними. Але для великих масивів фіксованих (не динамічних) розмірів доводиться робити виключення з цього правила.

**Підзадача D2.** Частину балів (орієнтовно до 20 з 50) можна отримати, розв'язуючи підзадачу D1 багатократно (для абсолютно кожної «1» у внутрішній стіні, замінимо її на «0» і заново розв'яжемо D1; серед усіх таких відповідей виберемо максимальну). Але такий алгоритм має складність  $O(M^2 \cdot N^2)$ , тож ніяк не може бути ефективним розв'язком для  $M \approx N \approx 1000$ .

Перепишемо програму-розв'язок підзадачі D1 так, щоб при підрахунку кількостей та розмірів кімнати не просто виділялися, а *різні кімнати* виділялися *різними значеннями* (а клітинки однієї кімнати — однаковими). Якщо це робити прямо у масиві з позначками «0 — прохід, 1 — стіна», виділяючи кімнати позначками 2, 3, 4, ..., може вийти, наприклад, так:

Вхідні дані	Виділені кімнати	Масив з розмірами кімнат					
		індекс	...	2	3	4	5
9 12							
111111111111	1 1 1 1 1 1 1 1 1 1 1 1						
101001000001	1 2 1 3 3 1 4 4 4 4 4 1	значення	...	1	5	28	9
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 числа, 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. Тому перебір усіх можливих «1» (у внутрішніх 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
1000000110000110000110001
1000000110000110000110001
1000000110000110000110001
1000000110000110000110001
1000000110000111111110001
1000000110000111111110001
111111110000110000110001
111111110000110000110001
1000000110000111111110001
1000000110000111111110001
1000000110000111111110001
1000000110000111111110001
1000000110000110010110001
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
1111111111	1111111111
10011011001	10011011001
11001010011	11000010011
11101010111	11101010111
10100100101	10100100101
11101010111	11101010111
11001010011	11001010011
10011011001	10011011001
11111111111	11111111111

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

Складність цього алгоритму теж  $\Theta(N \cdot M)$ . Але отой константний множник, яким нехтують у асимптотичних позначеннях, значно більший.

### 3.7 Обласна інтернет-олімпіада 2015/16 н. р.

Задачі доступні для дорішування ([ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua), змагання №12).

#### Задача А. «Високосні роки»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 64 мегабайти

Сучасний григоріанський календар діє, починаючи з п'ятниці 15 жовтня (октября) 1582 року. Він відрізняється від попереднього юліанського трохи складнішим, але зате й більш точним правилом визначення, які роки високосні (мі-

стять дату 29 лютого (февраля)), а які ні. У юліанському правило було дуже просте: якщо номер року кратний 4 (іншими словами, ділиться на 4 без остачі), то лютий містить 29 днів, а якщо не кратний, то 28 днів. Але Земля обертається навколо Сонця за приблизно 365,242 доби, а не 365,250 рівно, що й призвело до накопичення похибки (10 днів за 1500 років). Тому правило визначення, чи високосний рік, у сучасному григоріанському календарі таке: **“Високосним є кожен четвертий рік, за винятком років, номер яких ділиться без залишку на 100, але не ділиться без залишку на 400.”** Зокрема, 1700, 1800 і 1900 роки були не високосними за григоріанським календарем (хоч і високосними за юліанським). Але 1600 і 2000 роки були високосними за обома календарями. Так і вийшло, що різниця між цими календарями, яка у XVI ст. складала 10 днів, наразі становить 13 днів.

Напишіть програму, яка читатиме номер року і визначатиме, чи високосний цей рік за кожним із цих календарів.

**Вхідні дані.** Єдине ціле число, у проміжку  $1583 \leq y \leq 4321$  — номер року.

**Результати.** Виведіть у першому рядку або єдине слово “YES”, якщо рік високосний за григоріанським календарем, або єдине слово “NO”, якщо не високосний. Другий рядок теж має містити єдине слово “YES”, або “NO”, з аналогічним смислом, але для юліанського календаря. Важливо, щоб слова “YES” та/або “NO” були написані великими латинськими літерами, без лапок.

**Приклади:**

Вхідні дані	Результати
2015	NO NO
2012	YES YES

Вхідні дані	Результати
2000	YES YES
1900	NO YES

**Розбір задачі.** З юліанським (старим) календарем усе дуже просто: `if year mod 4 = 0 then writeln('YES') else writeln('NO')`.

З григоріанським (новим) трохи складніше, але потребує лише акуратності, без придумок. Буквально слідуючи умові задачі, можна отримати таке: `if (year mod 4 = 0) and ((year mod 100 <> 0) or (year mod 400 = 0)) then writeln('YES') else writeln('NO')`. Інший більш-менш зручний спосіб — спочатку, якщо рік взагалі не кратний 4, то не високосний; інакше, якщо

кратний 400, то високосний; інакше, якщо кратний 100, то не високосний (випа- док кратності 400 вже розглянутий в іншій гілці); інакше, високосний (і взагалі не кратні 4, і особливі випадки вже розглянуті раніше).

Можна і не писати перевірку високосності за григоріанським календа- рем самому, а використати бібліотечну. Наприклад, мовою FreePascal є фун- кція `IsLeapYear`; мовою Java — метод `isLeapYear` класу `GregorianCalendar`.

Чи завжди можна використовувати на олімпіаді бібліотечні засоби замість писати сам(ому/ій)? Якщо бібліотека не стандартна — мабуть, ні (додавати бі- біліотеки на сервер на прохання учасника якось не прийнято). Крім того, на ло- кальному комп'ютері, де пише учасник, і на сервері, де відбувається перевірка, можливі відмінності у переліку бібліотек, що вважаються стандартними, та/або у версіях компілятора. Тут лишається тільки або знати конкретні факти (які бібліотеки в яких версіях є), або питати в журі, або пробувати (якщо багатокра- тна відправка розв'язків дозволена, можна витратити кілька спроб, щоб узнати, чи працює на сервері програма з використанням бажаної бібліотеки — навіть не написавши розв'язок повністю, дивитися, чи виходить вердикт «Помилка компіляції», чи «Неправильна відповідь», тощо).

## Задача В. «Фасування олії»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`      Обмеження пам'яті: 64 мегабайти

На виробництві по переробці насіння в день виробляють  $N$  літрів олії. Для її фасування використовують тару об'ємом 1, 2, 3, 4, 5, 6 літрів.

Напишіть програму `oil`, яка б давала можливість з'ясувати, яку мінімаль- ну кількість тари необхідно для фасування.

**Вхідні дані.** Єдине ціле число  $N$  — кількість літрів олії.

**Результати.** Програма повинна вивести єдине ціле число — мінімальну кількість тари.

**Приклади:**

Вхідні дані	Результати
7	2
12	2
19	4

**Розбір задачі.** Завдяки тому, що в переліку об'ємів тари є всі підряд об'єми 1, 2, 3, 4, 5 та 6 л, справедливо, що завжди можна використати  $N \text{ div } 6$  штук якнайбільших 6-літрових пляшок, а решту (якщо така є, тобто якщо  $N \bmod 6 > 0$ ) розмістити у ще одну, останню, пляшку. Ще меншої кількості пляшок точно не достатньо, бо сумарний об'єм меншої кількості пляшок точно був би строго меншим потрібного.

Тобто, відповідь можна виразити як `res:=N div 6; if N mod 6>0 then inc(res)`, або навіть одним виразом  $\lceil N/6 \rceil$  (тобто  $N/6$ , заокруглене догори) — це може бути записано як  $(N+5) \text{ div } 6$  (Pascal) чи `ceil(N/6.0)` (C/C++).

Насамкінець, якби у переліку об'ємів були не всі підряд числа від 1 до 6, а лише деякі, жадібний алгоритм «взяти якнайбільшу кількість якнайбільших пляшок» міг би виявитися і неправильним (наприклад: якби існували лише пляшки 1, 5 та 6 л, і пофасувати треба було би 16 л, то оптимальний спосіб 6+5+5 отримувався б всупереч згаданій жадібній ідеї).

## Задача С. «Нестандартне рівняння»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка шукатиме (шляхом перебору) кількість розв'язків на проміжку  $[a; b]$  рівняння

$$k \times f(n) = n,$$

де  $f(n)$  — сума квадратів цифр у десятковому записі числа  $n$ .

**Вхідні дані.** Єдиний рядок містить три розділених пробілами цілі числа, у порядку  $k$  а  $b$ . Гарантовано  $a \leq b$ , а також, що усі ці числа не менші 1 й не більші  $10^{18}$ .

**Результати.** Програма повинна вивести єдине число — кількість таких  $n$  з проміжку  $a \leq n \leq b$ , що задовольняють рівняння.

**Приклади:**

Вхідні дані	Результати
51 5000 10000	3

**Примітки.** Цими трьома розв'язками є 7293, 7854 та 7905. Наприклад, 7293 є розв'язком, бо  $51 \times (7^2 + 2^2 + 9^2 + 3^2) = 51 \times (49 + 4 + 81 + 9) = 51 \times 143 = 7293$ .

Але нічого цього виводити не треба, треба лише кількість розв'язків.

**Розбір задачі.** Таке «рівняння» не має стандартних аналітичних засобів розв'язання, тож справді треба якось перебирати різні потенційно можливі варіанти й перевіряти, які з них дають розв'язок. Але перебір «в лоб» (для кожного  $n$  від  $a$  до  $b$  рахувати суму квадратів цифр і перевіряти виконання чи не виконання «рівняння») надто довгий, аж до  $10^{18}$  ітерацій (час роботи — багато років). Тому, перебір треба суттєво оптимізувати. Помітимо такі факти:

1. Для виконання  $k \times f(n) = n$ , число  $n$  мусить бути кратним  $k$ , тож можна зразу перебирати лише ті числа від  $a$  до  $b$ , які кратні  $k$  (у цієї ідеї є дещо частково спільне з ідеями зі стор. 35 та 60...).
2. На всьому проміжку до  $10^{18}$  найбільшу суму квадратів цифр має  $\underbrace{99\dots9}_{18 \text{ штук}}$ , тому для всіх  $n$  виконується  $f(n) \leq 18 \times 9^2 = 1458$ .

Тобто, перебір можна почати з найменшого  $n$ , одночасно кратного  $k$  і більшого або рівного  $a$ , і продовжувати, збільшуючи щоразу на  $k$ , доки не виявиться більшим  $b$  або більшим  $1548 \times k$ . Очевидно, *такий* цикл повторюватиметься ніяк не більше 1548 разів, що для комп'ютера зовсім мало. Лишається тільки реалізувати все це акуратно, щоб не було проблем ні з переповненнями типу `int64` (`long long`), ні з похибками (якщо використовувати для проміжних обчислень типи з плаваючою точкою). Наприклад, [ideone.com/PCj7qR](http://ideone.com/PCj7qR)

## Задача D. «П'ять неділь у місяці»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`      Обмеження пам'яті: 64 мегабайти

Для сучасного григоріанського календаря (деякі маловідомі його деталі наведені у задачі А), порахуйте, які з місяців уведеного року містять аж 5 неділь (як днів тижня; рос. воскресений, англ. sundays).

**Вхідні дані.** Єдине ціле число, у проміжку  $1583 \leq y \leq 4321$  — номер року.

**Результати.** Розділені пробілами номери місяців, що містять по 5 неділь, обов'язково у порядку зростання. “Номери місяців” слід розуміти так: січень (январь) — 1, лютий (февраль) — 2, і так далі, грудень (декабрь) — 12. Вказувати кількість таких місяців не треба.

**Приклади:**

Вхідні дані	Результати
2015	3 5 8 11
2012	1 4 7 9 12

Вхідні дані	Результати
2000	1 4 7 10 12
1900	4 7 9 12

**Примітки.** Зокрема, у першому з прикладів стверджується, що у 2015 році є чотири таких місяці: березень (март), травень (май), серпень (август), листопад (ноябрь).

Для підтвердження, що це справді так, а також для загальної довідкової інформації, наведемо повний календар на 2015 рік:

Січень	Лютий	Березень	Квітень	Травень	Червень
5 12 19 26	2 9 16 23	2 9 16 23 30	6 13 20 27	4 11 18 25	1 8 15 22 29
6 13 20 27	3 10 17 24	3 10 17 24 31	7 14 21 28	5 12 19 26	2 9 16 23 30
7 14 21 28	4 11 18 25	4 11 18 25	1 8 15 22 29	6 13 20 27	3 10 17 24
1 8 15 22 29	5 12 19 26	5 12 19 26	2 9 16 23 30	7 14 21 28	4 11 18 25
2 9 16 23 30	6 13 20 27	6 13 20 27	3 10 17 24	1 8 15 22 29	5 12 19 26
3 10 17 24 31	7 14 21 28	7 14 21 28	4 11 18 25	2 9 16 23 30	6 13 20 27
4 11 18 25	1 8 15 22	1 8 15 22 29	5 12 19 26	3 10 17 24 31	7 14 21 28
Липень	Серпень	Вересень	Жовтень	Листопад	Грудень
6 13 20 27	3 10 17 24 31	7 14 21 28	5 12 19 26	2 9 16 23 30	7 14 21 28
7 14 21 28	4 11 18 25	1 8 15 22 29	6 13 20 27	3 10 17 24	1 8 15 22 29
1 8 15 22 29	5 12 19 26	2 9 16 23 30	7 14 21 28	4 11 18 25	2 9 16 23 30
2 9 16 23 30	6 13 20 27	3 10 17 24	1 8 15 22 29	5 12 19 26	3 10 17 24 31
3 10 17 24 31	7 14 21 28	4 11 18 25	2 9 16 23 30	6 13 20 27	4 11 18 25
4 11 18 25	1 8 15 22 29	5 12 19 26	3 10 17 24 31	7 14 21 28	5 12 19 26
5 12 19 26	2 9 16 23 30	6 13 20 27	4 11 18 25	1 8 15 22 29	6 13 20 27

**Розбір задачі.** Тут ще істотніше, ніж у задачі А, що можна або реалізувати все самостійно, або використати бібліотечні засоби.

**Самостійна реалізація.** Можна використати (згаданий в умові задачі А) факт, що 15.10.1582 було п'ятницею, отже 01.01.1583 — субота (наприклад,



по наданому в умові даної задачі календарю можна підрахувати, що 1 січня наступного року припадає на наступний після 15 жовтня день тижня).

Ще можна згадати (чи побачити у наведеному календарі), що не високосний рік містить цілу кількість тижнів + 1 день. Відповідно, високосний — цілу кількість тижнів + 2 дні. Отже:

01.01.1584	неділя	(01.01.1583 субота+1)
01.01.1585	вівторок	(01.01.1584 неділя+2, бо 1584 рік високосний)
01.01.1586	середа	(01.01.1585 вівторок+1)
01.01.1587	четвер	(01.01.1586 середа+1)
01.01.1588	п'ятниця	(01.01.1587 четвер+1)
01.01.1589	неділя	(01.01.1588 п'ятниця+2, бо 1588 рік високосний)
⋮	⋮	⋮

Значить, можна перебрати усі роки від 1583 по  $y-1$  (де  $y$  — рік зі вхідних даних; кілька тисяч ітерацій — для комп'ютера зовсім не багато) і щоразу додавати 1 чи 2, користуючись правильною перевіркою високосності з задачі А. Так узнаємо, яким днем тижня починається потрібний рік  $y$ .

Тепер можна, наприклад, послідовно перебрати усі дні цього року, як у [ideone.com/m71JWc](http://ideone.com/m71JWc). Можна і трохи оптимізувати, перебираючи лише місяці (а не дні), як у [ideone.com/gnuUXY](http://ideone.com/gnuUXY). Але раз раніше перебирали роки (можливо, кілька тисяч), виграш не буде справді помітним. Ще, можна замість функції `daysInMonth` тримати масив кількостей днів (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31) (у випадку високосності року 28 міняється на 29). І так далі — у цій задачі взагалі багато правильних різновидів розв'язку...

**Розв'язки, що істотно спираються на бібліотечні засоби.** Вони істотно залежать від конкретних бібліотек та конкретної мови програмування, тож нема сенсу займати тут багато місця описами цих бібліотек. Наведемо лише приклади: [ideone.com/2jLmH1](http://ideone.com/2jLmH1) (Delphi чи FreePascal), [ideone.com/X9xo5s](http://ideone.com/X9xo5s) (Java), а описи відповідних бібліотек пропонуємо знайти в Інтернеті самостійно. Див. також зауваження наприкінці пояснень до задачі А даного змагання.

### 3.8 II (районний/міський) етап 2015/16 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання № 51).

#### Задача А. «Купе»

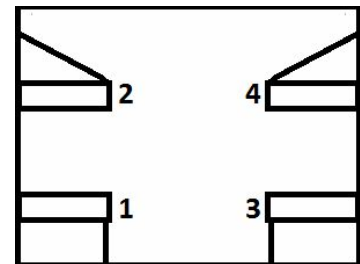
Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек

Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Купейний вагон містить 36 пасажирських місць — 9 купе по 4 місця у кожному. Половина серед цих місць нижні, решта — верхні. На рисунку наведено схему розміщення місць 1, 2, 3, 4 у купе №1. Купе №2 містить аналогічно розміщені місця 5, 6, 7, 8, і так далі, до купе №9, яке містить аналогічно розміщені місця 33, 34, 35, 36.

Напишіть програму, яка, прочитавши номери двох різних місць одного купейного вагону, визначатиме:

- чи в одному й тому ж купе розміщені ці місця;
- верхнє чи нижнє перше з цих місць;
- верхнє чи нижнє друге з цих місць.



**Вхідні дані.** Два числа  $p_1 p_2$  у одному рядку, розділені одним пропуском (пробілом). Гарантовано виконуються обмеження  $1 \leq p_1 \leq 36$ ,  $1 \leq p_2 \leq 36$ ,  $p_1 \neq p_2$ .

**Результати.** Перший рядок повинен містити або єдине слово NO (якщо місця у різних купе), або слово YES і після нього через пробіл номер того купе, в якому розміщені обидва ці місця. Другий рядок повинен містити або єдине слово LOW (якщо перше з уведених місць нижнє), або єдине слово HIGH (якщо верхнє). Третій рядок теж повинен містити або слово LOW, або слово HIGH, але стосовно другого з уведених місць.

**Приклади:**

Вхідні дані	Результати
2 3	YES 1 HIGH LOW

Вхідні дані	Результати
23 17	NO LOW LOW

**Примітка.** Перевірте правильність написання у Вашій програмі слів YES, NO, LOW, HIGH (ВЕЛИКИМИ латинськими літерами). Автоматична перевіряюча система зараховує відповідь, лише коли вона правильна буква-в-букву.

**Розбір задачі.** Для пошуку першого рядка відповіді слід реалізувати код, що знаходить номер купе за номером місця, й порівняти результати для двох введених місць. Для такого визначення може підійти формула  $(p+3) \operatorname{div} 4$ , де  $\operatorname{div}$  — цілочисельне ділення. Для мов програмування, де є стандартна функція заокруглення вгору (зазвичай називається `ceil`), можливий трохи природніший підхід: поділити на 4 з отриманням дробового результату, потім заокруглити вгору. Наприклад, це може мати вигляд `(int)(ceil(p/4.0))` (C/C++).

Легко бачити, що верхніми є місця з непарними номерами, нижніми — з парними. Тобто, якщо залишок від ділення на 2 (Pascal — `p mod 2`, C/C++ — `p%2`) рівний 0, то місце верхнє, якщо 1 — нижнє. Можливі й інші способи перевірки парності, зокрема: Pascal — `if odd(p) then...`, C/C++ — `if(p&1)...` (обидва дають `true` при непарному `p`, деталі знайдіть самостійно).

Складність усього разом узятого алгоритму очевидно  $\Theta(1)$ .

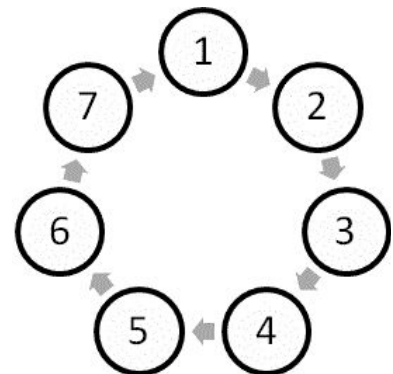
У [ideone.com/OiknfP](http://ideone.com/OiknfP) визначення номера купе і виведення LOW/HIGH оформлені як підпрограми. Це не обов'язково, але дозволяє уникнути деяких прикрих помилок, як-то «помилівся у формулі визначення номера купе, скопіював неправильну, потім для одного з місць виправив, а для іншого забув». Тому, винесення таких дій у підпрограми вважається правильним стилем і прикладом т. зв. «повторного використання коду» (`code reusing`).

## Задача В. «Гра»

Вхідні дані: Або клавіатура, або `input.txt`      Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`      Обмеження пам'яті: 64 мегабайти

В дитячій грі по колу розташовані кульки, кожна з них має номер від 1 до  $N$  ( $1 \leq N \leq 100$ ). По черзі з кола забирають кожну  $K$ -ту кульку. Це відбувається до тих пір, поки в колі не залишиться остання кулька. Напишіть програму `game`, яка б визначала номер останньої кульки, що залишилась. При вхідних даних 7 3 кульки забираються у послідовності №3, №6, №2, №7, №5, №1.



**Вхідні дані.**  $N, K$ .

**Результати.** Номер кульки, що залишилась.

**Приклад:**

Вхідні дані	Результати
7 3	4

**Розбір задачі.** Наголосимо, що приклад з умови «№3, №6, №2, №7, №5, №1» якраз відповідає описаним словами правилам:

- Маємо («закручену» в коло) послідовність 1 2 3 4 5 6 7. Рахуємо, починаючи з №1, доки не дійдемо до 3, тобто кажемо «1» на №1, «2» на №2, «3» на №3. Значить, прибрати з кола слід кульку №3.
  - Маємо («закручену») послідовність 1 2 4 5 6 7, і обрахунки починаються з кульки №4 (наступної після щойно прибраної кульки №3). Тобто, кажемо «1» на №4, «2» на №5, «3» на №6, і прибираємо з кола кульку №6.
  - «Закручена» послідовність 1 2 4 5 7, починаємо з кульки №7 — кажемо «1» на №7, «2» на №1, «3» на №2, і прибираємо кульку №2.
  - «Закручена» послідовність 1 4 5 7, починаємо з кульки №4 — кажемо «1» на №4, «2» на №5, «3» на №7, і прибираємо кульку №7.
  - «Закручена» послідовність 1 4 5, починаємо з кульки №1 — кажемо «1» на №1, «2» на №4, «3» на №5, і прибираємо кульку №5.
  - «Закручена» послідовність 1 4, починаємо з кульки №1 — кажемо «1» на №1, «2» на №4, «3» на №1 і прибираємо кульку №1.
- Лишається єдина кулька №4, вона й буде відповіддю.

На жаль, чимало учасників це не зрозуміли. Зокрема, кілька учасників стверджували, ніби кульки мають забиратися у порядку «№3, №6, №2, №5, №1, №4, №7». Хоча це протирічить тому, що кульки забирають з кола. Кінець кінцем, приклад для того і наданий, щоб учасники могли звірити з ним правильність свого розуміння задачі. Автори задачі та журі (принаймні, міське по м. Черкаси) вважають, що з урахуванням всього цього ніякої неоднозначності умови нема.

Щодо власне розв'язку — при  $n \leq 100$  годиться який-небудь спосіб «у лоб». Наприклад, підтримувати масив із номерами ще не забраних кульок, якимось так:

1. Заповнити масив як `for i:=1 to n do who[i]:=i` та встановити поточну позицію на початковий індекс 1;
2.  $n-1$  раз повторити такі дії:
  - (a) обчислити, яка позиція знаходиться на  $k-1$  правіше поточної (з урахуванням «закільцьованості» послідовності), і перейти туди;
  - (b) видалити елемент у поточній позиції (зі зсувом «хвоста»).

Повторювати треба  $n-1$  раз, бо один останній елемент лишається. Змінювати позицію від поточної треба на  $k-1$ , бо сама поточна — вже або перша, а не нульова (на першій ітерації циклу), або перша після щойно видаленої (на подальших ітераціях), тож 1 з  $k$  вже враховано, лишається  $k-1$ .

Приклади реалізації щойно наведеного алгоритму можна бачити за посиланнями [ideone.com/sRGzzS](http://ideone.com/sRGzzS) (Pascal) та [ideone.com/0tr18C](http://ideone.com/0tr18C) (C++ з активним використанням STL). Складність такого алгоритму становить  $O(n^2)$  (кожен зсув потребує  $O(n)$  дій), що для  $1 \leq n \leq 100$  цілком прийнятно (навіть «із запасом»).

Є й інші більш-менш прості способи, прийнятні при  $n \leq 100$ . Наприклад, при вилученні кульки можна не зсовувати хвіст масиву, а ставити якесь спеціальне число (наприклад,  $-1$ ), яке позначало б «видалено». Але тоді, щоб знайти число, яке видалятимуть наступним, треба послідовно перебирати елементи масива, відраховуючи  $k$  ще не видалених і пропускаючи вже видалені.

Для загального розвитку корисно знати, що для цієї задачі відомі й ефективніші алгоритми (що могло б бути актуальним, якби  $n$  було не до 100, а, наприклад, до  $10^5$  чи до  $10^6$ ). Їх можна знайти за ключовими словами «задача Йосипа Флавія» (рос. «задача Иосифа Флавия», англ. «Josephus problem») — адже саме так насправді й називається ця задача.

## Задача С. «Сучасне мистецтво»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 64 мегабайти

Галерея готує показ творів сучасного мистецтва кількох митців і розглядає різні шляхи оформлення виставки. Так як це сучасне мистецтво, всі твори одного

художника мало чим відрізняються один від одного.

Наприклад, припустимо, що є один твір митця А, два твори митця В і один твір митця С. Тоді є 12 різних шляхів оформити виставку в галереї:

- |         |         |         |          |
|---------|---------|---------|----------|
| 1. АВВС | 4. ВАВС | 7. ВВСА | 10. САВВ |
| 2. АВСВ | 5. ВАСВ | 8. ВСАВ | 11. СВАВ |
| 3. АСВВ | 6. ВВАС | 9. ВСВА | 12. СВВА |

Список приведений в алфавітному порядку.

Напишіть програму, що визначає  $n$ -ий спосіб оформлення виставки, враховуючи що всі способи слідуєть в алфавітному порядку.

Ваша програма приймає на вхід 5 цілих чисел:  $a$ ,  $b$ ,  $c$  і  $d$  (кожне від 0 до 5 включно), що визначають кількість робіт митців А, В, С і D відповідно, і останнє число  $n$  ( $1 \leq n \leq 2^{34}$ ) —  $n$ -ий спосіб оформлення виставки. Гарантується, що хоча б один митець виставляє свою роботу в галереї. Також гарантується, що  $n$  не перевищує загальної кількості способів.

Як результат ваша програма має вивести рядок (із літер А, В, С, D), що визначає  $n$ -ий спосіб оформлення виставки.

**Приклад:**

Вхідні дані	Результати
1 2 1 0 8	ВСАВ

**Розбір задачі.** Задача має багато спільного з задачею С «Генератор паролів — password» III (обласного) етапу 2014/15 н. р. (стор. 58–62). А саме, тут теж варто знайти, скільки всього можливих послідовностей починається з 'А' і або побачити, що задане у вхідних даних  $n \leq$  цієї кількості (тоді послідовність-відповідь починається з 'А' і продовжити підбирати літери на подальших позиціях), або, у протилежному випадку, відняти з заданого у вхідних даних  $n$  кількість цих послідовностей (що починаються з 'А') і зробити те саме щодо послідовностей, що починаються з 'В', 'С', 'D'.

Тож треба з'ясувати, як знаходити кількість послідовностей, які починаються з такої-то послідовності літер (і тому відомо, з яких літер у яких кількостях складаються). Такі виборки (усі елементи розставляють усіма можливими порядками, не змінюючи власне набір цих елементів) називають *перестановками*

(англ. *permutations*). Якщо у наборі елементів, які переставляють, є однакові (наприклад, "АВСВ", причому, якщо обміняти місцями одне 'В' з іншим 'В', це вважатиметься тим самим способом), то виборки називають *перестановками з повтореннями*. Просимо знайти у літературі чи Інтернеті доведення, що кількість перестановок з повтореннями може бути виражена як

$$\bar{P}(k_1, k_2, \dots, k_n) = \frac{(k_1 + k_2 + \dots + k_n)!}{k_1! \cdot k_2! \cdot \dots \cdot k_n!},$$

де "!" означає факторіал,  $k_1, k_2, \dots, k_n$  — кількості елементів кожного типу. В цій задачі, формула набуває вигляду  $\bar{P}(a, b, c, d) = \frac{(a + b + c + d)!}{a! \cdot b! \cdot c! \cdot d!}$ . (При розв'язуванні задачі доведення не потрібне, достатньо знати формулу; але доведення і допомагає її запам'ятати, і може бути корисним у інших ситуаціях.)

Лишається тільки поєднати розглянуті міркування. Зробіть це самостійно. А для контролю правильності розглянемо приклад  $a=1, b=2, c=3, d=4, n=2015$ .

При початку на 'А', можливих продовжень є  $\bar{P}(0, 2, 3, 4)$ , бо якщо взяти на першу позицію 'А', на подальші позиції не лишається жодної 'А',  $b=2$  штук 'В',  $c=3$  штук 'С',  $d=4$  штук 'D'.  $\frac{(0 + 2 + 3 + 4)!}{0! \cdot 2! \cdot 3! \cdot 4!} = 1260 < 2015$ , тобто шукана виборка починається не на 'А', а на одну з подальших літер, і номер шуканої виборки серед тих подальших дорівнює  $2015 - 1260 = 755$ .

При початку на 'В', можливих продовжень є  $\bar{P}(1, 1, 3, 4)$ , бо якщо взяти на першу позицію 'А', на подальші позиції лишається  $a=1$  штука 'А',  $b=1$  штука 'В',  $c=3$  штук 'С',  $d=4$  штук 'D'.  $\frac{(1 + 1 + 3 + 4)!}{1! \cdot 1! \cdot 3! \cdot 4!} = 2520 \geq 755$ , тобто шукана виборка починається якраз на 'В', і треба шукати подальші літери виборки.

При початку на "ВА", можливих продовжень є  $\bar{P}(0, 1, 3, 4)$ , бо якщо взяти на перші дві позиції "ВА", на подальші позиції лишається  $a=0$  штук 'А',  $b=1$  штука 'В',  $c=3$  штук 'С',  $d=4$  штук 'D'.  $\frac{(0 + 1 + 3 + 4)!}{0! \cdot 1! \cdot 3! \cdot 4!} = 280 < 755$ , тобто шукана виборка починається не на "ВА", а має на другій позиції одну з подальших літер, і номер шуканої виборки серед тих подальших дорівнює  $755 - 280 = 475$ .

При початку на "ВВ", можливих продовжень є  $\bar{P}(1, 0, 3, 4) = 280 < 475$ , тож шукана виборка починається не на "ВВ", а має на другій позиції одну з подальших літер, і номер серед тих подальших рівний  $475 - 280 = 195$ .

При початку “BC”, можливих продовжень  $\bar{P}(1, 1, 2, 4) = \frac{(1+1+2+4)!}{1! \cdot 1! \cdot 2! \cdot 4!} = 840 \geq 195$ , тож шукана виборка починається якраз на ‘BC’.

При початку “BCA”, можливих продовжень  $\bar{P}(0, 1, 2, 4) = \frac{(0+1+2+4)!}{0! \cdot 1! \cdot 2! \cdot 4!} = 105 < 195$ , тож шукана виборка має на другій позиції одну з подальших літер, номер серед подальших  $195 - 105 = 90$ .

При початку “BCB”, можливих продовжень  $\bar{P}(1, 0, 2, 4) = 105 \geq 90$ , тож шукана виборка починається якраз на ‘BCB’.

При початку “BCBA”, можливих продовжень  $\bar{P}(0, 0, 2, 4) = \frac{(0+0+2+4)!}{0! \cdot 0! \cdot 2! \cdot 4!} = 15 < 90$ , тож шукана виборка має на другій позиції одну з подальших літер, номер серед подальших  $90 - 15 = 75$ .

Тепер треба правильно врахувати (наприклад, передбачити іф для таких ситуацій), що літер ‘B’ спочатку було лише  $b=2$  й тому початок “BCBB” взагалі неможливий (кількість=0) і тому треба продовжити, перейшовши до “BCBC”.

І так далі.

У тексті все це займає багато місця, але виконується такий алгоритм дуже швидко — сумарна кількість обчислень формули  $\bar{P}(\dots)$  не перевищує  $4 \times (a+b+c+d)$ , тож загальна кількість дій може бути виражена як  $O((a+b+c+d)^2)$  навіть при не найкращій реалізації самої формули; якщо ж факторіали не обчислювати щоразу заново, а зберігати готові, можна оптимізувати усю програму разом узятую до  $\Theta(a+b+c+d)$ , але при настільки малих  $a, b, c, d$  це несуттєво.

**Як легко й просто набрати 60% балів (лише C++).** У бібліотеці `algorithm` мови C++ є функція `next_permutation`, яка дозволяє послідовно перебирати перестановки у порядку зростання (що й треба у цій задачі), і яка правильно працює з будь-якими перестановками (з повтореннями чи без). Так що можна сформувати `string s` зі значенням  $\underbrace{\text{AA} \dots \text{A}}_{a \text{ штук}} \underbrace{\text{BB} \dots \text{B}}_{b \text{ штук}} \underbrace{\text{CC} \dots \text{C}}_{c \text{ штук}} \underbrace{\text{DD} \dots \text{D}}_{d \text{ штук}}$  і  $n-1$  разів застосувати `next_permutation(s.begin(), s.end())`; складність такого алгоритму становить  $\Theta(n)$  (причому, `next_permutation`, хоч і має амортизовану складність  $\Theta(1)$ , не дуже швидко), тож він ніяк не може бути досить



ефективним при  $n \approx \overline{P}(5, 5, 5, 5) = \frac{20!}{5! \cdot 5! \cdot 5! \cdot 5!} \approx 1,173 \cdot 10^{10}$ . В умові цієї задачі не описано, скільки балів на які діапазони вхідних даних припадає, тож з умови не ясно, що це саме 60%; але очевидно, що якісь бали мусять бути.

**А інші?** На жаль, для інших мов програмування важко запропонувати простий спосіб набрати помітну частину балів. Це сумно, але: (1) 100%-ий алгоритм цілком можна написати будь-якою з доступних мов програмування; (2) нікому з учасників не заборонялося вчити й використовувати мову C++.

Писати власний аналог `next_permutation` не доцільно, бо потребує майже стільки ж зусиль, як ефективний розв'язок початкової задачі, а принести може, щонайбільше, ті ж 60% балів. Тим не менш, кому цікаво, може знайти у літературі чи Інтернеті *алгоритми генерації перестановок*.

Є кілька різних класичних алгоритмів такої генерації. І *одні* з них (які визначають, що поміняти у поточній перестановці, щоб перейти до наступної) оптимізовані під відносно швидшу організацію перебору усіх підряд перестановок (у `next_permutation` використано саме такий). А *інші* (зокрема, рекурсивні) дозволяють таку модифікацію, як вписування додаткових умов, щоб пропускати непотрібні. Говорячи конкретно про цю задачу, такі модифікації можна кінець кінцем перетворити до рекурсивної версії способу зі стор. 78–80. . .

## Задача D. «Кількість дільників на проміжку»

Вхідні дані: Або клавіатура, або input.txt      Обмеження часу: 1 сек

Результати: Або екран, або output.txt      Обмеження пам'яті: 4 мегабайти

Напишіть програму, яка знайде суму кількостей дільників усіх чисел у проміжку від  $A$  до  $B$  (обидві межі включно,  $A \leq B$ ).

**Вхідні дані.** У єдиному рядку через пробіл задані два натуральні числа  $A$  та  $B$ , які являють собою межі проміжку.

**Результати.** Виведіть у одному рядку єдине число — суму кількостей дільників усіх чисел проміжку.

**Приклад:**

Вхідні дані	Результати
119 122	27

**Примітка.** Число 119 має 4 дільники (1, 7, 17, 119); число 120 має 16 дільників (1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120); число 121 має 3 дільника (1, 11, 121); число 122 має 4 дільники (1, 2, 61, 122). Звідси відповідь  $4+16+3+4 = 27$ .

**Оцінювання.** 40% балів припадає на тести, в яких  $1 \leq A \leq B \leq 1000$ . Ще 20% балів припадає на тести, в яких  $1 \leq A \leq B \leq 10^7$ . Ще 10% балів припадає на тести, в яких  $10^8 \leq B \leq 10^9$ , але  $B-100 \leq A \leq B$ . Решта 30% балів припадає на тести, де виконуються обмеження  $1 \leq A \leq B \leq 10^{12}$  і не виконуються обмеження попередніх блоків.

Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього переліку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу правильно, але не ефективно.

**Розбір задачі. Розв'язок на 40–50%%.**

Для отримання 40% балів достатньо реалізувати те, що написано в умові. Наприклад, див. праворуч. Але складність такого алгоритма  $\Theta((B-A+1) \times B)$ , тож він отримує вердикт «Перевищено час роботи» на всіх блоках тестів, крім 1-го ( $1 \leq A \leq B \leq 1000$ ).

```
res:=0;
for c:=a to b do
  for p:=1 to c do
    if c mod p = 0
      then
        res:=res+1;
```

Враховуючи міркування зі стор. 22 про те, чому для знаходження дільників  $N$  достатньо перебирати «претендентів» до  $\sqrt{N}$  (а не до  $N$ ), наведений алгоритм можна модифікувати, лишивши незмінним зовнішній цикл `for c:=a to b do`, але зменшивши діапазон перебору внутрішнього з `c` до  $\sqrt{c}$ . Це дає складність  $\Theta((B-A+1) \times \sqrt{B})$ , що допустимо як при обмеженнях 1-го блоку « $1 \leq A \leq B \leq 1000$ », так і 3-го блоку « $10^8 \leq B \leq 10^9$ , але  $B-100 \leq A \leq B$ ».

**Повний розв'язок.** В усіх подальших алгоритмах, кількості дільників набагато легше не шукати на проміжку від  $A$  до  $B$ , а знайти один раз на проміжку від 1 до  $B$ , інший — на проміжку від 1 до  $(A-1)$ , й відняти другий результат з першого. Тим паче, що у програмуванні це означає, що такий пошук пишеться *один* раз, оформляється як функція, і двічі викликається.

Не варто дотримуватися (нав'язуваної приміткою з умови) тактики «знайдемо кількості дільників кожного числа й пододаємо». Кількості дільників окре-

мих чисел ніхто не питає, тож можна порахувати всі ті самі дільники якимось інакше. Наприклад, у іншому порядку.

Число 1 є дільником усіх чисел, тож дільник 1 приносить у шукану суму стільки, скільки чисел у проміжку. Число 2 є дільником усіх парних чисел, тож приносить у шукану суму стільки, скільки на проміжку парних чисел. І т. д. А серед усіх чисел від 1 до  $N$  є рівно  $N \operatorname{div} k$  чисел, кратних  $k$ .

На перший погляд, так треба перебирати всі дільники аж до  $B$ , і реалізація такого алгоритму легко пройде тести 1-го та 2-го блоків (отримавши таким чином 60% балів), але буде не досить ефективною ні для 4-го блоку тестів, ні навіть для 3-го. Але це можна оптимізувати, використавши вже згаданий факт «якщо  $p$  є дільником  $N$ , то  $(N/p)$  теж є дільником  $N$ , а з чисел  $p$  та  $N/p$  хоча б одне  $\leq \sqrt{N}$ ». Приблизно так: перебравши дільники лише до  $\sqrt{B}$ , подвоїти результат, щоб урахувати дільники, більші  $\sqrt{B}$ . Це не зовсім правда, бо враховує деякі дільники двічі; але цю похибку можна компенсувати, й отримати правильний алгоритм складності  $\Theta(\sqrt{B})$ .

Розглянемо всі дільники усіх чисел від 1 до 32. Кружечки позначають, що число «№ рядка» є дільником числа «№ стовпчика». Чорний (●) кружечок — дільник менший кореня відповідного числа, білий (○) — більший, напівзаповнений — рівний. Вертикальні лінії виражають згаданий попарний зв'язок між  $p$  та  $N/p$ . Штрихова горизонтальна лінія посередині між числами 5 та 6 виражає перебір дільників до  $\sqrt{B}$  (точніше кажучи,  $\lfloor \sqrt{B} \rfloor$ , воно ж  $\operatorname{trunc}(\operatorname{sqrt}(B))$  чи  $\operatorname{floor}(\operatorname{sqrt}(B))$ ).

Очевидно, сумарну кількість дільників можна виразити як кількість чорних кружечків, помножену на 2, плюс кількість напівзаповнених. А для пошуку

кількості чорних достатньо перебрати рядки з 1-го по  $\lfloor \sqrt{B} \rfloor$ -й, тільки треба правильно врахувати кількості білих та напівзаповнених.

Остаточо, при  $1 \leq i \leq \sqrt{n}$  (де  $n$  — аргумент щойно згаданої функції) кількість усіх кружечків у рядку № $i$  становить  $N \operatorname{div} i$ , і серед них є рівно 1 напівзаповнений і рівно  $i-1$  білий (що дає кількість чорних  $(N \operatorname{div} i) - i$ ).

Завершіть усі ці міркування та перетворіть їх до вигляду програми мовою програмування самостійно.

---

Насамкінець, можлива ситуація, коли учасник олімпіади придумав і вміє реалізувати як алгоритм, що проходить 1-й та 2-й блоки тестів, так і алгоритм, що проходить 1-й та 3-й блоки, але не придумав останню зі згаданих ідей, яка дозволяє розв'язати задачу повністю. Чи може такий учасник гарантувати собі 70% балів, узявши однією програмою бали і за 2-й, і за 3-й блоки? Запросто, якщо зробить так: прочитавши зі вхідних даних  $A$  та  $B$ , знайде, який із виразів  $(A+B)$  чи  $((B-A) \times \sqrt{B})$  виявляється меншим, і залежно від цього викличе чи то одну, чи то іншу підпрограму, що реалізує відповідний алгоритм.

## 4 Список використаних джерел

1. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — М.–СПБ–К., Вильямс, 2005. — 1296 с.
2. Левитин А. Алгоритмы: введение в разработку и анализ / А Левитин. — М.–СПБ–К., Вильямс, 2006. — 576 с.
3. Порублёв И. Н. Алгоритмы и программы. Решение олимпиадных задач / И. Н. Порублёв, А. Б. Ставровский — М.–СПБ–К., Диалектика, 2007. — 576 с.
4. Сайт [ejudge.skipo.edu.ua](http://ejudge.skipo.edu.ua) [Електронний ресурс]
5. Сайт [uk.wikipedia.org](http://uk.wikipedia.org) [Електронний ресурс]
6. Сайт [codeforces.com](http://codeforces.com) [Електронний ресурс]
7. Сайт [habrahabr.ru](http://habrahabr.ru) [Електронний ресурс]
8. Сайт [www.freepascal.org](http://www.freepascal.org) [Електронний ресурс]
9. Сайт [en.cppreference.com](http://en.cppreference.com) [Електронний ресурс]
10. Сайт [ideone.com](http://ideone.com) [Електронний ресурс]