ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2016–2017 НАВЧАЛЬНОМУ РОЦІ

Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва.
Рибак Олександр Владиславович,
молодший науковий співробітник Інституту математики НАН України.
Рудик Олександр Борисович,
доцент Київського університету імені Бориса Грінченка.

Стаття містить умови завдань II (районного) і завдань III (міського) етапів олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2016/2017 навчальному році й авторські розв'язання цих завдань. Публікацію адресовано учням класів з поглибленим вивченням математики, учасникам олімпіад з інформатики, студентам математичних спеціальностей, учителям і викладачам вищих навчальних закладів.

Завдання II етапу і задачі 1–3 III етапу упорядкував Данило Мисак, задачі 4–5 III етапу — Олександр Рудик, задачу 6 III етапу — Олександр Рибак.

I. УМОВИ ЗАВДАНЬ II ЕТАПУ

Максимальна оцінка за кожну з чотирьох задач — 100 балів. Для всіх задач обмеження на час — 1 секунда / тест; обмеження на пам'ять — 256 МБ.

1. Карти

(назва програми: **cards.cpp/cards.pas/cards.***)

Якось, граючи у карти, Нетямко помітив, що в нього на руці вісім різних карт червової масті: є всі карти від двійки до десятки за винятком однієї. Допоможіть Нетямку визначити, якої саме карти бракує.

Вхідні дані

У вхідному файлі у довільному порядку задано вісім різних натуральних чисел у межах від 2 до 10.

Вихідні дані

У вихідний файл виведіть натуральне число у межах від 2 до 10 — карту червової масті, якої бракує.

Приклад

Вхідний файл **cards.in**	Вихідний файл **cards.out**
6 3 10 9 2 8 4 5	7

Історична довідка

Гральні карти мають тисячолітню історію. Їх було винайдено в Китаї, а перша згадка датується IX століттям та описує «гру у листя».

2. Хрестики-нулики

(назва програми: **ttt.cpp/ttt.pas/ttt.***)

У хрестики-нулики грають на полі 3×3 двоє гравців, що ходять по черзі. Перший гравець ставить хрестик у довільну клітинку поля; його суперник ставить нулик у будь-яку іншу клітинку; перший гравець ставить ще один хрестик у будь-яку незайняту клітинку; далі його суперник ставить нулик і т. д. Виграє той, після чийого ходу деякі три хрестики або три нулики стоятимуть на одній горизонталі, на одній вертикалі чи на одній з двох діагоналей поля. Якщо всі клітинки поля вже зайнято, але жодні три хрестики або нулики не займають однієї горизонталі, вертикалі чи діагоналі, вважають, що гра завершилась унічию.

Якось Нетямко грав у хрестики-нулики з товаришем і замислився: чи не час закінчувати поточну партію, а якщо так, то хто виграв? Допоможіть хлопцю знайти відповіді на ці запитання.

Вхідні дані

Вхідний файл складається з трьох рядків, у кожному з яких міститься по три символи з набору: **x** (мала латинська літера *x*, що позначає хрестик), **o** (мала латинська літера *o*, що позначає нулик), **.** (крапка, що позначає порожню клітинку). Символи пробілами не розділено. Вхідні дані задають коректну позицію гри у хрестики-нулики (можливо, й початкову, коли ще не зроблено жодного ходу).

Вихідні дані

вихідний файл виведіть єдиний символ (не забувши про перенесення рядка):

x — малу латинську літеру *x*, — якщо гра вже закінчилася і виграли хрестики;

o — малу латинську літеру *o*, — якщо гра вже закінчилася і виграли нулики;

= — знак рівності, — якщо гра закінчилася внічию;

. — крапку, — якщо гра ще не закінчилася.

Приклади

Вхідний файл **ttt.in**	Вихідний файл **ttt.out**
..x ox. x.o	x
ooo .x. xx.	o
x.x xoo oxo	.

Історична довідка

Існує припущення, що хрестики-нулики походять зі Стародавнього Єгипту. В усякому разі у дуже схожу гру під назвою Terni Lapilli грали у I ст. до н. е. у Римській імперії, на підтвердження чому по всьому Риму залишилися сліди накреслених крейдою ігрових полів.

3. Бики та корови

(назва програми: **bac.cpp/bac.pas/bac.***)

Правила гри в бики й корови такі. Гравець задумує довільне чотирицифрове (від 1000 до 9999) число, усі цифри якого різні; суперник намагається задумане число відгадати, висуваючи гіпотези: він поступово на-

зиває різні чотирицифрові числа, що також не містять однакових цифр. На кожну гіпотезу суперника гравець, що задумав число, повинен відповісти — вказати кількість угаданих суперником цифр: ті вгадані цифри, що стоять на правильних місцях, називаються *биками*, а ті, які є в задуманому числі, але стоять на інших позиціях, називаються *коровами*. Наприклад, якщо задумано число 7183, а названо 8123, то гравець, що задумував число, відповість «два бики та одна корова» (два бики — цифри 1 і 3, що стоять на своїх місцях, а корова — цифра 8, що стоїть не там, де треба).

Якось, граючи з товаришем, Нетямко спитав у нього про чисел, ретельно записав відповіді про кожне з них і був уже близький до перемоги, але зненацька його записи розсипалися й Нетямко заплутався, яка відповідь товариша відповідала якому названому Нетямком числу. Спираючись на переплутані записи, допоможіть Нетямку вгадати, яке число задумав його товариш.

Вхідні дані

У першому рядку вхідного файлу вказано натуральне число . У наступних рядках міститься по одному чотирицифровому числу, про яке Нетямко питав у товариша (усі числа різні). У наступних рядках записано по два цілих числа — відповіді товариша: перше число — кількість биків, а друге число — кількість корів у деякому з чисел, про які питав Нетямко. Зверніть увагу, що набір з чисел відповідає набору з відповідей у деякому *переплутаному* порядку.

Вихідні дані

У вихідний файл виведіть задумане товаришем Нетямка число. Відомо, що це число можна відновити однозначно і воно не збігається з жодним із чисел, заданих у вхідному файлі.

Приклад

Вхідний файл **bac.in**	Вихідний файл **bac.out**
3 4712 8796 3261 2 0 0 3 1 2	1726

Коментар до прикладу

Назвавши число 4712, Нетямко дістав відповідь «один бик і дві корови». Назвавши число 8796, Нетямко отримав відповідь «два бики і жодної корови». Назвавши число 3261, Нетямко дістав відповідь «жодного бика і три корови». Потім рядки в його записах переплуталися.

Історична довідка

Походження гри в бики та корови не встановлене, але відома ця гра вже щонайменше протягом ста років.

4. Вісім

(назва програми: **eight.cpp** / **eight.pas** / **eight.***)

Гра у вісім проходить на дошці 3?3, де розміщено у довільному порядку 8 плиток, позначених числами від 1 до 8 (кожне число трапляється рівно по разу), а дев'ята клітинка — вільна (рис. 1).

Мета гри — пересуваючи плитки, досягти того, щоб вони розмістилися в порядку зростання номерів зліва направо зверху вниз, а вільна клітинка розташувалася при цьому у правому нижньому куті (рис. 2).

На кожному кроці пересувати у вільну клітинку можна лише сусідні з нею плитки: перекладати плитки заборонено!

6	7	2
1		3
5	4	8

Рис. 1

1	2	3
4	5	6
7	8	

Рис. 2

Нетямко випадковим чином виклав на дошці початкову конфігурацію плиток і хотів уже було почати гру, але замислився: а раптом головоломка вимагає великої кількості пересувань плиток, а то й зовсім не підлягає вирішенню? Бо якщо так, то він навряд чи зможе її розв'язати… Допоможіть хлопцю зекономити час та розумові зусилля: за заданим початковим розташуванням плиток визначте, за яку найменшу кількість кроків плитки можна викласти у правильному порядку, або встановіть, що досягти цієї мети неможливо.

Вхідні дані

У вхідному файлі міститься опис початкової позиції гри: три рядки файлу містять по три цілих числа від 0 до 8, де числа від 1 до 8 позначають плитки з відповідними номерами, а число 0 — порожню клітинку. Числа не повторюються.

Вихідні дані

У вихідний файл виведіть найменшу кількість кроків, за які з початкової позиції можна одержати потрібну. Одним кроком ми вважаємо пересування однієї плитки на одну позицію: зсувати в одному напрямку водночас дві сусідніх плитки не можна! Якщо початкова позиція збігається з кінцевою, виведіть 0, а якщо бажану позицію одержати шляхом пересувань плиток неможливо, виведіть −1.

Приклад

Вхідний файл **eight.in**	Вихідний файл **eight.out**
1 2 3 4 6 8 7 0 5	5

Коментар до прикладу

Досягти потрібного розташування плиток з початкового за 5 кроків можна у спосіб, показаний на рис. 3.

Історична довідка

Гру у вісім (а точніше її варіант на дошці 4×4) було винайдено у 1870-х роках у США. Американський шахіст Сем Лойд, авторству якого інколи помилково приписують гру, запропонував велику грошову винагороду за розв'язання однієї з початкових позицій головоломки. Та ба — позиція виявилася нерозв'язною.

II. ІДЕЇ РОЗВ'ЯЗАННЯ ЗАВДАНЬ II ЕТАПУ

1. Карти

Задачу можна розв'язувати у різноманітні способи. Один з найпростіших такий: відняти від числа $54 = 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$ суму всіх

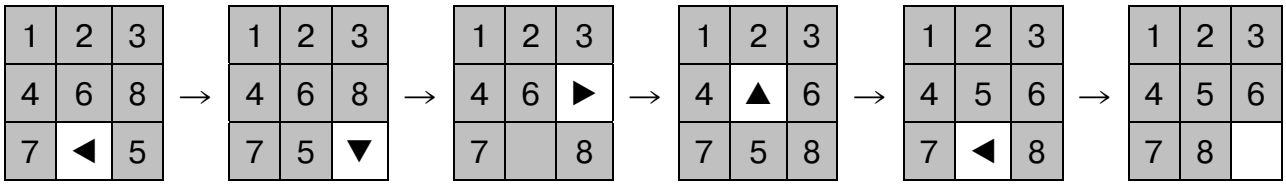

Рис. 3

чисел, записаних у вхідному файлі. Різниця і буде пропущеним числом.

2. Хрестики-нулики

Необхідно перевірити всі вісім ліній поля на предмет того, чи містять вони три хрестики або три нулики. Якщо це так, вивести відповідний символ і завершити виконання алгоритму. Якщо ні, то з'ясувати, чи залишилася на полі принаймні одна вільна клітинка (гра ще триває) або такої клітинки немає (гра завершилася внічию).

Розгляд восьми ліній можна реалізувати безпосереднім «ручним» перебором або ж певним чином оптимізувати. Наприклад, написати окремі функції перевірки горизонталей, вертикалей та діагоналей, у які передавати номер відповідної лінії. Інший варіант — кожну лінію розглядати як пару з її середньої клітинки та «зсуву» відносно неї бокових клітинок. Маємо чотири лінії, що проходять через центральну клітинку поля, і чотири лінії, що проходять по його краях. Середні клітинки та зсуви обох цих наборів ліній можна задати за допомогою формул, що дозволяє зручно їх запрограмувати (див. авторське розв'язання).

3. Бики та корови

Щоб розв'язати задачу, достатньо перебрати всі чотирицифрові числа, які міг задумати товариш Нетямка, і вибрати з них те, для якого набір відповідей на висловлені гіпотези буде з точністю до порядку збігатися із заданим у вхідному файлі.

Щоб з'ясувати, чи два набори чисел збігаються з точністю до порядку, можна обидва ці набори *відсортувати* й далі порівняти поелементно. Так само порівнюються і набори пар чисел: їх можна впорядкувати, наприклад, за кількістю биків, а в разі рівності цього показника — за кількістю корів. Утім, ще ефективнішим буде підхід, що використовується у *сортуванні підрахунком:* оскільки пари кількостей биків та корів можуть набувати лише одного з 13 видів:

(0/0, 0/1, 0/2, 0/3, 0/4, 1/0, 1/1, 1/2, 1/3, 2/0, 2/1, 2/2, 3/0),

ми просто порахуємо кількість пар кожного з цих видів і порівняємо між собою отримані кількості для двох наборів. А щоб рахувати було зручніше, відповідь «*b* биків і *c* корів» ототожнимо з числом $5b+c$, яке назвемо *індексом* такої відповіді. Тоді можливим парам відповідатиме проміжок індексів від 0 до 15 (пропуск індексів 9, 13 і 14 не є при цьому істотним).

4. Вісім

Задачу розв'язують *пошуком у ширину*: вершини графа — позиції гри, а ребро між двома вершинами-позиціями наявне тоді й лише тоді, коли з однієї з цих позицій можна перейти в іншу за один крок.

Облік позицій можна вести по-різному. Один з підходів — ототожнити позицію з дев'ятицифровим (або восьмицифровим) числом, яке вийде, якщо прочитати числа на плитках зліва направо зверху вниз — наприклад, позиція, яку необхідно отримати, матиме тоді вигляд 123 456 780. Але в такому випадку індексацію доведеться вести або за допомогою якої-небудь нетривіальної структури даних (як-от *дерево*), або скориставшись контейнером *map* бібліотеки STL мови C++. Інший варіант — кожну позицію розглядати як перестановку дев'яти чисел (цифр від 0 до 8) й ототожнювати з порядковим номером відповідної перестановки (числом від 0 до 9! – 1=362 879), якщо розглядати перестановки в порядку від тих, що утворюють менші числа, до тих, що утворюють більші числа. Щоб відновити за певною перестановкою її порядковий номер, знайдемо кількість перестановок, «менших» за дану: це сума кількості всіх перестановок, що починаються на цифру, меншу за першу цифру даної; тих, що починаються на ту саму цифру, але мають меншу другу цифру; тих, що мають однакові перші дві цифри, але меншу третю, і т. д.

III. АВТОРСЬКІ РОЗВ'ЯЗАННЯ ЗАВДАНЬ II ЕТАПУ

1. Карти

```
/* GCC */
#include <stdio.h>
int main()
{
 freopen(«cards.in», «r», stdin);
 freopen(«cards.out», «w», stdout);
 int n = 54;
 for (int i = 0; i < 8; i++)
 {
  int k;
  scanf(«%d», &k);
  n –= k;
 }
 printf(«%d\n», n);
}
```

2. Хрестики-нулики

```
/* GCC */
#include <stdio.h>
char grid[9]; // Ігрове поле:
// перший рядок записано в комірках 0, 1, 2,
// другий — у комірках 3, 4 і 5,
// третій — у комірках 6, 7 і 8

// Функція перевіряє, чи зайнято хрестиками
// або нуликами лінію, середня комірка якої
// має у масиві grid індекс center, а бокові
// — індекси center-delta та center+delta:
bool checkLine(int center, int delta)
{
 return grid[center] != '.'
```

```
 && grid[center − delta] == grid[center]
 && grid[center + delta] == grid[center];
}
int main()
{
 freopen(«ttt.in», «r», stdin);
 freopen(«ttt.out», «w», stdout);
 // Зчитування даних та обчислення
 // кількості хрестиків і нуликів
 // на полі:
 int xCnt = 0, oCnt = 0;
 for (int i = 0; i < 3; i++)
 {
  for (int j = 0; j < 3; j++)
  {
   char c;
   scanf(«%c», &c);
   grid[i * 3 + j] = c;
   if (c == 'x') xCnt++;
   if (c == 'o') oCnt++;
  }
  scanf(«\n»);
 }
 for (int i = 1; i <= 4; i++)
 if (checkLine(4,i) // Лінії, що проходять
      // через центральну клітинку поля

 ‖ checkLine(i*2−1, i%3 == 1 ? 1 : 3))
                  // Решта ліній
 {//Якщо хоч одну лінію зайнято, виводимо
  //того, хто зробив останній хід:
   printf(«%c\n», xCnt > oCnt ? 'x' : 'o');
   return 0;
 }
 // Якщо жодну лінію не зайнято, дивимось,
 // чи залишилися ще вільні клітинки:
 printf(«%c\n», xCnt + oCnt < 9 ? '.' : '=');
}
```

3. Бики та корови

```
/* GCC */
#include <stdio.h>
#define digitsCnt 4//Кількість цифр у числах
#define numLimit 10000
// Обмеження на величину числа: 10^digitsCnt
#define indexLimit 16     // Обмеження на
// «індекс» відповіді: дорівнює digitsCnt^2
int n, guesses[numLimit]; // Числа-гіпотези
bool guess_index[numLimit];
// guess_index[k] матиме значення true
// тоді й лише тоді, коли серед названих
// чисел було число k
// За кількістю биків та корів повертає
// «індекс» такої відповіді — ціле число
// у межах від 0 до digitsCnt^2−1,
// що однозначно відповідає даним кількостям
// (тобто якщо пари чисел різні,
//  то й індекси в них різні):
inline int index(int bulls, int cows)
{
 return bulls * (digitsCnt + 1) + cows;
}
// Перевіряє, чи всі цифри числа num різні:
inline bool valid(int num)
{
 int d[digitsCnt]; // Сюди буде записано
              // цифри числа num
 for (int i = 0; i < digitsCnt; i++)
 {
  d[i] = num % 10;
  for (int j = 0; j < i; j++)
  // Якщо якісь дві цифри збігаються,
  // повертаємо false
  if (d[j] == d[i])
   return false;
  num /= 10;
 }
 return true;   // Якщо жодні дві цифри
      // не збіглися, повертаємо true
}
// Рахує кількість биків і корів у числах
// a та b і повертає індекс відповіді:
inline int count(int a, int b)
{
 int dpos[10];
 // У dpos[d] буде записано позицію, на
 // якій стоїть цифра d у числі a, або −1,
 // якщо такої цифри у числі a нема
 int bulls = 0, // Кількість биків
   cows  = 0; // Кількість корів
 for (int i = 0; i < 10; i++)
  dpos[i] = −1; // Ініціалізація dpos
 for (int i = 0; i < digitsCnt; i++)
 {          // Заповнення dpos
  dpos[a % 10] = i;
  a /= 10;
 }
 for (int i = 0; i < digitsCnt; i++)
 // Розкладаємо на цифри число b
 // і порівнюємо з результатами для a,
 // записаними в dpos
 {
  int pos = dpos[b % 10];
  if (pos == i)
   bulls++;
  else if (pos != −1)
       cows++;
  b /= 10;
 }
 return index(bulls, cows);
}
// Після виклику функції в ans[ind] буде
// записано кількість чисел з масиву
// guesses, відповіді на які мають індекс
// ind, якщо задумано число num:
inline void getAnswers(int num, int ans[])
{
 for (int i = 0; i < indexLimit; i++)
  ans[i] = 0;        // Ініціалізація
 for (int i = 0; i < n; i++) //Підрахунок
  ans[count(num, guesses[i])]++;
}
// Визначає, чи збігаються два індексні
// масиви відповідей (формат масивів такий,
// який обчислює функція getAnswers):
```

```
inline bool compare(int ansA[], int ansB[])
{
 for (int i = 0; i < indexLimit; i++)
  if (ansA[i] != ansB[i])
   return false;
 return true;
}
int main()
{
 freopen(«bac.in», «r», stdin);
 freopen(«bac.out», «w», stdout);
 for (int i = 0; i < numLimit; i++)
  // Ініціалізація guess_index
  guess_index[i] = false;
 // Зчитування даних та заповнення
 // guess_index:
 scanf(«%d», &n);
 for (int i = 0; i < n; i++)
{
 int k;
 scanf(«%d», &k);
 guesses[i] = k;
 guess_index[k] = true;
}
int ans[indexLimit];
// Масив у тому ж форматі, що обчислює
// функція getAnswers, але для набору
// відповідей, заданих у вхідному файлі
for (int i = 0; i < indexLimit; i++)
 ans[i] = 0;    // Ініціалізація ans
for (int i = 0; i < n; i++)
{             // Заповнення ans
 int bulls, cows;
 scanf(«%d%d», &bulls, &cows);
 ans[index(bulls, cows)]++;
}
for (int i = 1000; i < 10000; i++)
if (! guess_index[i] && valid(i))
// Для всіх чотирицифрових чисел,
// що не траплялися у вхідному файлі
// та не містять однакових цифр
{
 int curAns[indexLimit];
 getAnswers(i, curAns);
 // Перевіряємо дане число та, якщо воно
 if (compare(ans, curAns))   // дає той
 // самий набір відповідей, який задано
 // у вхідному файлі (з точністю до
 // порядку ), виводимо знайдене число
 // та завершуємо виконання
 {
  printf(«%d\n», i);
  break;
 }
}
}
```

4. Вісім

```
/* GCC */
#include <stdio.h>
#include <algorithm>
#include <vector>
#include <queue>
#include <map>

using namespace std;

#define sz 3 // Ширина/висота дошки
#define n 9  // Кількість клітинок sz^2

#define target 123456780
    // Позиція, якої потрібно досягти

// Функція приймає на вхід масив цифр
// (digits), розташування плитки, яку
// потрібно пересунути на порожнє місце
// (число k у межах від 0 до n−1), та
// розташування порожньої, тобто нульової,
// клітинки (число z у межах від 0 до n−1);
// повертає число, що відповідає позиції,
// яка утвориться після пересування:
inline int moveDigit (int digits[],
                int k, int z)
{ // Пересуваємо плитку:
 digits[z] = digits[k];
 digits[k] = 0;
 // Рахуємо відповідь:
 int result = 0;
 for (int i = 0; i < n; i++)
 result = result * 10 + digits[i];

 // Пересуваємо плитку назад, щоб
 // залишити вхідний масив незмінним:
 digits[k] = digits[z];
 digits[z] = 0;
 return result;
}
// За числом-позицією видає набір позицій, у
// які з даної можна потрапити за один крок:
inline vector<int> getMoves(int pos)
{
 int digits[n];  // Цифри числа pos
 int zero;       // Індекс у масиві digits
        // цифри 0 (порожньої клітинки)
 for (int i = n − 1; i >= 0; i—)
 {  // Заповнюємо digits і визначаємо zero
  digits[i] = pos % 10;
  pos /= 10;
  if (digits[i] == 0)
   zero = i;
 }
 vector<int> result; // Массив для всіх
 // досяжних за один крок позицій
 // Якщо порожня клітинка не в лівому
 // стовпці, можна пересунути плитку зліва:
 if (zero % sz != 0)
  result.push_back
  (moveDigit(digits, zero − 1, zero));

 // Якщо порожня клітинка не у правому
 // стовпці, можна пересунути плитку
 // справа:
 if ((zero + 1) % sz != 0)
  result.push_back
```

```
  (moveDigit(digits, zero + 1, zero));
 // Якщо порожня клітинка не у верхньому
 // рядку, можна пересунути плитку зверху:
 if (zero >= sz)
  result.push_back
  (moveDigit(digits, zero – sz, zero));

 // Якщо порожня клітинка не у нижньому
 // рядку, можна пересунути плитку знизу:
 if (zero < n – sz)
  result.push_back
  (moveDigit(digits, zero + sz, zero));
 return result;
}
int main()
{
 freopen(«eight.in», «r», stdin);
 freopen(«eight.out», «w», stdout);

 // Зчитуємо початкову позицію:
 int start = 0;
 for (int i = 0; i < n; i++)
 {
  int d;
  scanf(«%d», &d);
  start = start * 10 + d;
 }

 queue<int> q; // Черга пошуку в ширину,
          //що початково містить лише
          // стартову позицію
 q.push(start);
 map<int, int> m;   //m[pos] — кількість
      // кроків, за які можна потрапити
      // у позицію pos із початкової
 m[start] = 0;
 while (! q.empty())
 {         // Поки в черзі є елементи
  int cur = q.front();
  // Беремо за поточну та видаляємо
  // з черги першу позицію
  q.pop();
  int distance = m[cur] + 1;
  // Відстань у кроках від початкової
  // позиції до тих, у які можна
  // потрапити з поточної за один крок
  vector<int> moves = getMoves(cur);
  for (int i = 0; i < moves.size(); i++)
  // Для кожного можливого ходу
  if (m.count(moves[i]) == 0)
  // Якщо отримана позиція
  // не розглядалася раніше,
  {
    m[moves[i]] = distance;
    // оновлюємо її інформацію
    q.push(moves[i]);
    // додаємо до черги пошуку в ширину
  }
 }
 // Якщо цільова позиція не розглядалася,
 // то відповідь –1, а інакше шукану
 // величину записано у відповідній
 // комірці m:
 printf(«%d\n», m.count(target) == 0 ? –1
                : m[target]);
}
```

```
/* GCC */
#include <stdio.h>
#define sz 3 // Ширина/висота дошки
#define n 9  // Кількість клітинок sz^2
#define mx 362880 // Кількість можливих
          // позицій-перестановок n!
int target[n] = {1, 2, 3, 4, 5, 6, 7, 8, 0};
// Позиція, якої необхідно досягти

int f[n];     // f[k] — факторіал числа k!
int q[mx][n];  // Черга пошуку в ширину:
  // q[k] містить k-ту перестановку з черги
int qs, qe;  // Вказівники на початок
        // та кінець черги
int m[mx];   // m[p] — кількість кроків, за
  // які перестановку номер p можна отримати
  // з початкової (–1 для позицій, яких ще
  // не досягнуто)
int qm[mx]; // qm[k] — кількість кроків,
// за які перестановку q[k] можна отримати
// з початкової (відповідні величини можна
// відновити з масиву m, але це вимагатиме
// додаткових викликів getIndex)

// Функція getIndex за перестановкою p
// повертає її номер:
// (0, 1, 2, 3, 4, 5, 6, 7, 8) має номер 0;
// (0, 1, 2, 3, 4, 5, 6, 8, 7) має номер 1;
// (0, 1, 2, 3, 4, 5, 7, 6, 8) має номер 2;
// ...
// (8, 7, 6, 5, 4, 3, 2, 1, 0) має номер
//                  362879 = 9!–1
inline int getIndex(int p[n])
{
 bool taken[n];// taken[d] — чи траплялася
          // на даний момент цифра d
 for (int i = 0; i < n; i++)
  taken[i] = false;    // Ініціалізація
                // масиву taken
 int result = 0;// Ініціалізація змінної,
      // в яку буде записано результат
 for (int i = 0; i < n; i++) // Ідемо від
  // перших цифр перестановки до останніх
 {
  int c; // Скільки цифр, менших за i-ту
  //цифру перестановки,траплялося раніше
  c = 0;          // Ініціалізація c
  for (int j = 0; j < p[i]; j++)
   if (taken[j])
    c++;
  result += (p[i] – c) * f[n – 1 – i];
  // Додаємо до номера кількість переста-
  // новок, у яких на i-му місці стоїть
  // менша цифра, а на попередніх місцях
  //  — ті самі цифри, що у даної
  // перестановки
  taken[p[i]] = true;//Актуалізація taken
```

```
  }
  return result;
}

// Функція getMoves приймає на вхід
// перестановку p, а також масив a, куди
// записує перестановки, що відповідають
// усім можливим ходам з позиції p;
// повертає кількість цих ходів
inline int getMoves(int p[n], int a[4][n])
{
 int zero;     // Розташування порожньої
    // клітини у позиції (перестановці) p
 zero = 0;        // Ініціалізація zero
 while (p[zero] != 0)// Визначення
  zero++;         // величини zero
 int cnt;     // Кількість можливих ходів
 cnt = 0;     // Ініціалізація cnt
 if (zero % sz != 0) // Якщо порожняя
  // клітинка не в лівому стовпці дошки,
 { // можемо пересунути на її місце
  // плитку зліва
  for (int i = 0; i < n; i++) // Спершу
   a[cnt][i] = p[i]; // копіюємо позицію
  a[cnt][zero] = a[cnt][zero - 1];
  a[cnt][zero - 1] = 0; // Пересуваємо
                // плитку
  cnt++; // Оновлюємо лічильник
 }
 if ((zero + 1) % sz != 0) // Якщо порожня
 // клітинка не у правому стовпці дошки,
 { // можемо пересунути на її місце
  // плитку справа
  for (int i = 0; i < n; i++) // Спершу
   a[cnt][i] = p[i]; // копіюємо позицію
  a[cnt][zero] = a[cnt][zero + 1];
  a[cnt][zero + 1] = 0; // Пересуваємо
                // плитку
  cnt++; // Оновлюємо лічильник
 }
 if (zero >= sz) // Якщо порожня клітинка
       // не у верхньому рядку дошки,
 { // можемо пересунути на її місце
  // плитку зверху
  for (int i = 0; i < n; i++) // Спершу
   a[cnt][i] = p[i]; // копіюємо позицію
  a[cnt][zero] = a[cnt][zero - sz];
  a[cnt][zero - sz] = 0; // Пересуваємо
                // плитку

  cnt++; // Оновлюємо лічильник
 }
 if (zero < n - sz) // Якщо порожняя
 // клітинка не у нижньому рядку дошки,
 {  // можемо пересунути на її місце
   // плитку знизу
  for (int i = 0; i < n; i++) // Спершу
   a[cnt][i] = p[i]; // копіюємо позицію
  a[cnt][zero] = a[cnt][zero + sz];
  a[cnt][zero + sz] = 0; // Пересуваємо
                // плитку
  cnt++; // Оновлюємо лічильник
 }
 return cnt;
}

int main()
{
 freopen(«eight.in», «r», stdin);
 freopen(«eight.out», «w», stdout);
 // Обчислюємо значення факторіалів:
 f[0] = 1;
 for (int i = 1; i < n; i++)
  f[i] = f[i - 1] * i;

 // Обчислення номера потрібної
 // позиції-перестановки:
 int targetIndex = getIndex(target);
 // Зчитування початкової позиції
 // як нульової позиції в черзі пошуку:
 for (int i = 0; i < n; i++)
  scanf(«%d», &q[0][i]);
 qm[0] = 0;
 qs = 0;
 qe = 1;

 // Ініціалізація m:
 for (int i = 0; i < mx; i++)
   m[i] = -1;
 m[getIndex(q[0])] = 0;

 while (qs < qe && m[targetIndex] == -1)
 // Поки в черзі є елементи та відповідь
 // ще не знайдено,
 { // розглядаємо чергову позицію
  int d = qm[qs] + 1;// Відстань у кроках
  // від початкової позиції до тих, у які
  // можна потрапити з поточної за 1 крок
  int moves[4][n], movesCnt =
  getMoves(q[qs], moves);
  // Можливі ходи з поточної позиції
  qs++; // «Видаляємо» з черги поточну
  // позицію, щоб на наступній ітерації
  // циклу розглянути нову
  for (int i = 0; i < movesCnt; i++)
  { // Для кожного можливого ходу
   int index = getIndex(moves[i]);
   // Рахуємо номер перестановки,
   // отриманої після ходу
   if (m[index] == -1)
   { // Якщо така позиція раніше не
    // траплялася, додаємо її до черги
    // та оновлюємо масиви m і qm
    m[index] = d;
    for (int j = 0; j < n; j++)
     q[qe][j] = moves[i][j];
    qm[qe] = d;
    qe++;
   }
  }
 }
 // Виведення відповіді, записаної
 // у відповідній комірці масиву m:
 printf(«%d\n», m[targetIndex]);
}
```

(Далі буде)

ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2016–2017 НАВЧАЛЬНОМУ РОЦІ

Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва.
Рибак Олександр Владиславович,
молодший науковий співробітник Інституту математики НАН України.
Рудик Олександр Борисович,
доцент Київського університету імені Бориса Грінченка.

Продовження, початок у №2 за 2017 рік

IV. УМОВИ ЗАВДАНЬ III ЕТАПУ

1. Сходові числа

Максимальна оцінка: 100 балів.
Обмеження на час: 1 сек.
Обмеження на пам'ять: 256 МБ.
Вхідний файл: numbers.in.
Вихідний файл: numbers.out.
Програма: numbers.*

Назвімо *сходовими числами* числа вигляду $a^{a^{\cdots^a}}$, де a — натуральне число, що входить у запис щонайменше двічі, а піднесення до степеня здійснюють «згори донизу». Наприклад, числа $3^3=27$ та

$$2^{2^{2^2}} = 2^{2^4} = 2^{16} = 65536$$ є сходовими.

Сходовим числом є й одиниця, бо $1=1^1$. А от, скажімо, числа 2, 3 і 5 сходовими не є, бо подати їх у потрібному вигляді неможливо.

Завдання. Установіть, скільки натуральних чисел, що не перевищують заданого натурального числа n, є сходовими.

Вхідні дані

У вхідному файлі записано натуральне число $n\le10^9$.

Вихідні дані

У вихідний файл виведіть відповідь — кількість сходових чисел у межах від 1 до n включно.

Приклад

numbers.in	numbers.out
5	2

2. Таблиця

Максимальна оцінка: 100 балів
Обмеження на час: 1 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: table.in
Вихідний файл: table.out
Програма: table.*

Задано квадратну таблицю $n\times n$, що складається з натуральних чисел. Розглядатимемо шляхи з лівої верхньої комірки таблиці у праву нижню, які довільним чином ідуть по комірках таблиці праворуч і вниз (але ніколи ліворуч або вгору). Уздовж кожного такого шляху розташовано $2n-1$ число. Якщо для певного шляху даний набір чисел є «симетричним», тобто читається у зворотному порядку так само, як і у прямому, називатимемо вибраний шлях *паліндромічним*.

Завдання. За заданою таблицею встановіть загальну кількість на ній паліндромічних шляхів.

Вхідні дані

У першому рядку вхідного файлу вказано розмір таблиці — ціле число n: $2\le n\le100$. У наступних n рядках файлу записано по n натуральних чисел, що не перевищують 10 000 і задають таблицю.

Вихідні дані

Вихідний файл повинен містити єдине число: остачу від ділення на 101 кількості паліндромічних шляхів на заданій таблиці.

Приклади

№	table.in	table.out
1	3 7 10 5 5 8 10 8 5 7	4
2	6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	50

Коментар. У першому прикладі є чотири паліндромічних шляхи:
1) праворуч — праворуч — униз — униз (7–10–5–10–7);
2) праворуч — униз — праворуч — униз (7–10–8–10–7);
3) униз — праворуч — униз — праворуч (7–5–8–5–7);
4) униз — униз — праворуч — праворуч (7–5–8–5–7).

У другому прикладі кожен з 252 можливих шляхів є паліндромічним, тож за умовою слід вивести остачу від ділення 252 на 101 — число 50.

3. Ламана

Максимальна оцінка: 100 балів
Обмеження на час: 1 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: polyline.in
Вихідний файл: polyline.out
Програма: polyline.*

Завдання. На площині позначено кілька точок. Якщо це можливо, побудуйте замкнену ламану без самоперетинів, що проходить через усі ці точки і не має відмінних від них вершин.

Вхідні дані

У першому рядку вхідного файлу задано кількість точок n, $3\le n<2017$. У n наступних рядках вказано по дві координати кожної точки: абсцису й ординату відповідно. Точок, що повторюються, нема-

є. Усі координати — цілі числа, що за модулем не перевищують 10 000.

Вихідні дані

Якщо побудувати потрібну замкнену ламану неможливо, слід вивести 0. В іншому разі треба вивести порядок, у якому задані точки розташовані на ламаній, тобто порядок обходу всіх цих точок, починаючи з довільного місця і в довільному напрямку (або за, або проти годинникової стрілки). Нумерація точок починається з одиниці. Ламаних, що задовольняють умову, може бути кілька. Достатньо знайти хоча б одну.

Приклад

polyline.in	polyline.out
7 1 0 -1 -1 1 1 -1 1 0 0 1 -1 0 -1	2 4 3 1 6 5 7

4. Дільники

Максимальна оцінка: 100 балів
Обмеження на час: 0,05 сек.
Обмеження на пам'ять: 32 МБ
Вхідний файл: divisors.in
Вихідний файл: divisors.out
Програма: divisors.*

Завдання. Пер Ґюнт, герой п'єси норвезького письменника Генріка Ібсена, потрапив у полон короля тролів. Щоб визволитися, Пер Ґюнт повинен відповісти на запитання: скількома способами дану кількість однакових діамантів можна розділити на одну чи кілька частин так, щоб кількість діамантів у кожній з них була однаковою?

Вхідні дані

Єдиний рядок файлу містить десятковий запис натурального числа n, що містить не більше, ніж 32 цифри й не має багатоцифрових (більше ніж 1 цифра у десятковому записі) простих дільників.

В 1/3 тестів це число не перевищує 10^9, у 2/3 тестів це число не перевищує 10^{18}.

Вихідні дані

Єдиний рядок файлу має містити десятковий запис кількості натуральних дільників числа n з вхідного файлу.

Приклад

divisors.in	divisors.out
12	6

5. Печери

Максимальна оцінка: 100 балів
Обмеження на час: 7 сек.
Обмеження на пам'ять: 128 МБ
Вхідний файл: caves.in
Вихідний файл: caves.out
Програма: caves.*

Пер Ґюнт, герой п'єси норвезького письменника Генріка Ібсена, потрапив у полон короля тролів. Щоб визволитися, Пер Ґюнт повинен перемогти у грі, яку король пропонує своїм бранцям. Правила гри такі:

- є 10 печер , деякі з яких сполучені між собою;
- бранець грає протягом m днів:
- у перший день він може вибрати собі довільну печеру;
- кожного наступного дня він може або залишитися у печері, у якій перебуває, або перейти у будь-яку іншу печеру, до якої є хід з поточної печери;
- кожного дня король винагороджує бранця певною наперед обумовленою кількістю золотих монет, яка залежить і від печери, і від номера дня;
- якщо за всю гру бранець накопичить кількість монет, виражену одним з l «королівських чисел», то бранця навіки залишають у полоні — він програв. Інакше його відпускають на волю з виграшем — усією накопиченою за час гри кількістю монет.

Завдання. Визначити, N_w, p_{max}, p_{min} — відповідно кількість різних виграшів, найбільший і найменший виграші.

Вхідні дані

Перший рядок файлу містить десяткові записи натуральних чисел m, l, де $m \le 60$.

Кожний з наступних 10 рядків містить по 10 цілих чисел — нулів або одиниць. При $1 \le j, k \le 10$: якщо k число $(j+1)$ рядка файлу дорівнює 1, то з j печери можна потрапити безпосередньо до k печери і навпаки. Інакше це зробити неможливо.

Кожний з наступних m рядків містить по 10 невід'ємних цілих чисел. При $1 \le j \le m$ і $1 \le k \le 10$: k число $(j+11)$ рядка — кількість монет, які отримує гравець як винагороду за перебування у j день у k печері.

Останній $(m+12)$ рядок містить у порядку спадання l різних цілих невід'ємних «королівських чисел», кожне з яких менше від 2^{64}. Останнє число у цьому рядку — нуль.

Вихідні дані

Єдиний рядок файлу містить десяткові записи трьох чисел: N_w, p_{max}, p_{min}. Якщо $N_w=0$, то $p_{max}=p_{min}=0$. Відомо, що $p_{max} < 2^{64}$, $l+N_w \le 10^6$.

Приклад

caves.in	caves.out
2 2 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 2 3 0 0 0 0 0 0 0 4 5 6 0 0 0 0 0 0 0 7 0	2 9 5

6. Розкладна послідовність

Максимальна оцінка: 100 балів
Обмеження на час: 1 сек.
Обмеження на пам'ять: 32 МБ
Вхідний файл: decomp.in
Вихідний файл: decomp.out
Програма: decomp.*

Кожний елемент послідовності $a_1, a_2, \ldots, a_n, \ldots$ є натуральним числом. Підпослідовність $a_{i_1}, a_{i_2}, \ldots, a_{i_n}, \ldots$ називатимемо *траєкторією*, якщо для кожного

натурального j справджується співвідношення $i_{j+1}=i_j+a_{i_j}$. Послідовність $a_1, a_2, \ldots, a_n, \ldots$ будемо називати *розкладною*, якщо її можна розділити на декілька траєкторій.

Завдання. Для періодичної послідовності $a_1, a_2, \ldots, a_n, \ldots$ визначити, чи є вона розкладною. Якщо це так, знайти кількість траєкторій, з яких складається задана послідовність.

Вхідні дані

Перший рядок містить натуральне число n ($1 \le n \le 100000$) — кількість чисел у періоді послідовності. У наступному рядку через пробіли задані натуральні числа $a_1, a_2, \ldots, a_n$, які складають один період — послідовність має вигляд $a_1, a_2, \ldots, a_n, a_1, a_2, \ldots, a_n, a_1, a_2, \ldots, a_n, \ldots$ Усі числа a_i не перевищують 100 000.

Вихідні дані

Якщо послідовність $a_1, a_2, \ldots, a_n, a_1, a_2, \ldots, a_n, a_1, a_2, \ldots, a_n, \ldots$ є розкладною, вивести кількість її траєкторій. У протилежному випадку вивести –1.

Приклади

№	decomp.in	decomp.out
1	3 5 3 1	3
2	3 1 3 5	-1
3	9 1 2 3 4 5 6 7 8 9	5

Коментар

У першому випадку траєкторіями є підпослідовності ($a_1, a_6, a_7, a_{12}, a_{13}, \ldots$), ($a_2, a_5, a_8, a_{11}, a_{14}, \ldots$) та ($a_3, a_4, a_9, a_{10}, a_{15}, \ldots$).

У другому випадку послідовність не можна розкласти, бо a_5 має одночасно бути наступним елементом у траєкторії після a_2 та після a_4.

V. ІДЕЇ РОЗВ'ЯЗАННЯ ЗАВДАНЬ III ЕТАПУ

1. Сходові числа

За допомогою нескладного перебору (ще до складання програми) можна переконатися, що існує лише 11 сходових чисел, які не перевищують 10^9:

$1=1^1$	$256=4^4$	$823\,543=7^7$
$4=2^2$	$3125=5^5$	$16\,777\,216=8^8$
$16=2\hat{}(2\hat{}2)$	$46\,656=6^6$	$387\,420\,489=9^9$
$27=3^3$	$65\,536=2\hat{}(2\hat{}(2\hat{}2))$	

Далі залишається кожне з них порівняти із заданим числом n.

2. Таблиця

Для заданого числа d, $0 \le d \le n-1$, розглянемо дві множини комірок таблиці, у які можна потрапити за d ходів з крайньої верхньої лівої комірки (перша множина) та нижньої правої комірки (друга множина). Ці множини збігаються при $d=n-1$, утворюючи побічну діагональ. На рис. 1 зображено випадок $n=8$, $d=4$. Для кожної пари комірок A та B відповідно з першої і другої множини (при фіксованому d) підраховуємо кількість *пар* шляхів, один з яких іде з лівої верхньої комірки в A, а інший — із правої нижньої комірки в B і таких, що набір чисел уздовж них однаковий. Позначимо цю кількість шляхів як $f(A, B)$, а попередні до A та B комірки на шляху від крайніх клітин — як A_1, A_2 та B_1, B_2 відповідно. Якщо в комірках A та B записано однакові числа, то маємо:

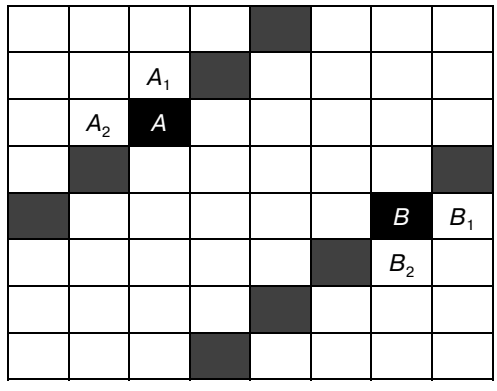

Рис. 1

$$f(A, B)=f(A_1, B_1)+f(A_1, B_2)+f(A_2, B_1)+f(A_2, B_2). \quad (*)$$

Інакше (якщо ці числа різні) $f(A, B)=0$. У випадку ж, коли хоча б одна з комірок A_i та/або B_j ($i=1$ чи 2; $j=1$ чи 2) відсутня, тобто розташована за межами таблиці, відповідне значення покладемо нульовим: $f(A_i, B_j)=0$. Окремо слід розглянути випадок $d=0$, коли A — ліва верхня комірка, а B — права нижня. Тоді $f(A, B)=1$, якщо в комірках записано однакові числа, інакше $f(A, B)=0$.

Використовуючи записані співвідношення, значення для всіх відповідних пар комірок можна обчислити за допомогою методу динамічного програмування, збільшуючи значення d від 0 до $n-1$. Відповіддю ж до задачі буде сума значень $f(A, A)$ для всіх комірок A з побічної діагоналі ($d=n-1$, див. рис. 2).

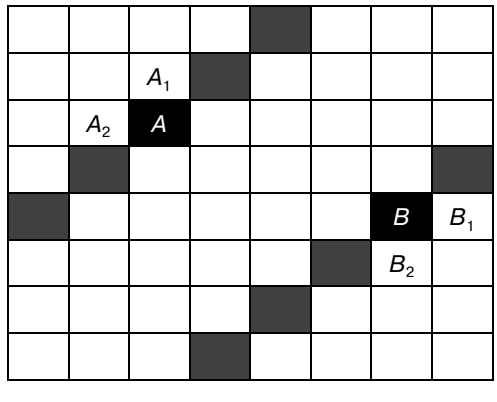

Рис. 2

В авторському розв'язанні використано альтернативний підхід до реалізації алгоритму — рекурсію із запам'ятовуванням. Часова складність обох підходів однакова — $O(n^3)$.

Не слід, звичайно, забувати, що після кожного підрахунку суми (*) цю суму потрібно замінити на остачу, яку вона дає від ділення на 101. Інакше дуже швидко значення будь-якого базового типу даних буде переповнено.

3. Ламана

Спершу побудуємо опуклу оболонку всіх точок. Потім відкинемо точки, які лежать на опуклій оболонці, і побудуємо опуклу оболонку точок, що залиши-

лися. Потім відкинемо ще й ті точки, які лежать на новій опуклій оболонці, й побудуємо опуклу оболонку решти точок і т. д. Зрештою отримаємо «мотрійку» з опуклих оболонок, *вкладених одна в одну*.

Назвемо ламану, яка має властивості, описані в умові задачі, *хорошою*. Зауважимо, що в нас є хороша ламана для тих точок, що лежать на зовнішній опуклій оболонці, — це сама опукла оболонка. Візьмемо на ній довільні дві сусідні точки A та B (вони не обов'язково є вершинами оболонки) та знайдемо точку другої оболонки (тобто тієї, яка стане зовнішньою, якщо першу зовнішню відкинути), яка лежить найближче до прямої AB. Нехай це точка P, а сусідніми з нею на другій оболонці є (з різних боків) точки Q та R. Тоді можна показати, що одна з пар відрізків AP і BQ, AQ і BP, AP і BR, AR і BP має таку властивість: відрізки не перетинаються між собою і з жодною з двох опуклих оболонок. Нехай, для визначеності, це пара відрізків AP і BQ. Тепер ми можемо побудувати хорошу ламану, яка міститиме всі точки, що потрапили на першу або на другу оболонку: це ламана, утворена з цих опуклих оболонок із відкинутими відрізками AB і PQ та доданими відрізками AP і BQ.

Аналогічно можемо побудувати хорошу ламану для всіх точок, що потрапили на одну з перших трьох оболонок, розглянувши дві довільні сусідні вершини A_1 та B_1 на другій оболонці, такі що $A_1B_1 \neq PQ$. Робитимемо таке й далі, доки не дійдемо до останньої, внутрішньої оболонки (на кожній наступній оболонці відкинуто буде два відрізки: один разом із відрізком попередньої оболонки, а один разом із відрізком наступної).

Внутрішня оболонка може бути виродженою. Якщо вона складається з єдиної точки, з'єднаємо цю точку з довільними сусідніми точками A_k та B_k попередньої оболонки такими, що відрізок A_kB_k не було відкинуто на попередньому кроці; після цього A_kB_k відкинемо. Якщо ж внутрішня оболонка складається з двох або більшої кількості точок, які лежать на одній прямій, візьмемо крайні з цих точок і позначимо їх як P_k та Q_k. Візьмемо також сусідні точки A_k та B_k попередньої оболонки такі, що відрізок A_kB_k не було відкинуто на попередньому кроці. Можна показати, що одна з пар відрізків A_kP_k і B_kQ_k, A_kQ_k і B_kP_k має таку властивість: відрізки не перетинаються між собою і з відрізком, який утворює внутрішню опуклу оболонку. Нехай, для визначеності, це пара відрізків A_kP_k і B_kQ_k. Проведемо ці відрізки, а відрізок A_kB_k відкинемо.

У такий спосіб ми побудували хорошу ламану, яка містить усі задані точки. Єдиний випадок, коли її побудувати не вдалося б, — якщо всі точки лежать на одній прямій (тоді для «внутрішньої» оболонки немає попередньої, з точками якої можна було б з'єднати її вершини).

Якщо для побудови всіх опуклих оболонок використовувати алгоритм Джарвіса, час виконання програми можна оцінити як $O(n^2)$.

4. Дільники

Число з вхідного файлу має такий вигляд:

$$2^{n_2} \cdot 3^{n_3} \cdot 5^{n_5} \cdot 7^{n_7},$$

де n_2, n_3, n_5, n_7 — невід'ємні цілі числа. Дільники такого числа мають такий вигляд:

$$2^{k_2} \cdot 3^{k_3} \cdot 5^{k_5} \cdot 7^{k_7},$$

де k_2, k_3, k_5, k_7 — невід'ємні цілі числа, що не перевищують відповідно n_2, n_3, n_5, n_7. При цьому k_2 може набувати (n_2+1) різних значень — нуль і n_2 натуральних значень: 1, 2, 3, ..., n_2. Аналогічно, k_3, k_5 та k_7 можуть набувати відповідно (n_3+1), (n_5+1) та (n_7+1) різних значень. Шукана кількість натуральних дільників дорівнює кількості різних наборів k_2, k_3, k_5, k_7, а саме добутку $(n_2+1) \cdot (n_3+1) \cdot (n_5+1) \cdot (n_7+1)$.

Невід'ємні числа n_2, n_3, n_5, n_7 можна знайти діленням з лишком. Вхідні дані забезпечують справдження нерівностей $n_2 \leq 100$, $n_3 \leq 100$, $n_5 \leq 100$, $n_7 \leq 100$.

Щоб набрати всі відведені бали, потрібно реалізувати подання числа з вхідного файлу масивом його цифр. Частковий випадок: основа системи числення 10^{16}. У цьому випадку достатньо використовувати 2 цифри, на які відвести по 64 біти .

5. Печери

Це завдання на відпрацювання навичок розв'язування типових комбінаторних задач з використанням рекурентних співвідношень і масивів. Основна ідея розв'язання така:

- перебираючи всі номери ходів у порядку зростання;
- для сталого номеру ходу перебираючи всі номери печер;
- на основі даних про те, які кількості монет можна отримати *після попереднього ходу в усі печери*, можна встановити, які кількості монет отримують *після поточного ходу у поточну печеру*.

Завершити розв'язання можна, долучивши ще один хід (скажімо, у тронний зал короля) з нульовим виграшем з усіх печер.

Щоб уникнути повторного розгляду кількостей монет, потрібно:

- або зберігати дані в упорядкованих (наприклад, за спаданням) масивах;
- або вести облік того, які кількості вже зустрілися.

Останній спосіб непридатний для повного розв'язання, бо вимагає 10^{18} бітів пам'яті.

Щоб зменшити кількість виконуваних дій при визначенні кількості монет після поточного ходу у поточну печеру потрібно аналізувати дані для всіх печер, з яких є хід у поточну печеру, *одночасно*. Останнє означає таке:

- спочатку оголосити перші елементи поточними для кожного з масивів для усіх печер, з яких є хід у поточну печеру;
- зі значень поточних елементів вибрати найбільше;
- у шуканий масив (значень кількостей монет, які можна отримати після поточного ходу у поточну печеру) внести суму вибраного найбільшого значення поточних елементів і винагороди за хід;
- для усіх масивів для печер, з яких є хід у поточну печеру, перейти до наступного елемента, якщо поточне значення збігається з вибраними максимальним.

Такий одночасний аналіз потрібно здійснювати до тих пір, поки не буде пройдено усі елементи усіх масивів, які аналізують. Цей аналіз є узагальненням принципу впорядкування лінійного масиву злиттям на (можливо) більшу кількість упорядкованих скла-

дових, але з перенесенням лише одного з однакових значень з цих складових у результат злиття.

6. Розкладна послідовність

Для розуміння зазначеного тут алгоритму спочатку доведемо дві теореми.

Теорема 1. *Нехай* $(a_1, a_2, ..., a_n)$ *— період послідовності, а* n *— довжина цього періоду. Доведемо, що послідовність розкладна тоді і лише тоді, коли числа* $a_1+1, a_2+2, ..., a_n+n$ *мають попарно різні лишки при діленні на* n.

Доведення. Спочатку розглянемо випадок існування натуральних чисел i та j ($1 \le i, j \le n$, $i \ne j$), для яких a_i+i та a_j+j дають однакові лишки при діленні на n. Маємо: $(a_i+i)-(a_j+j)=sn$, де s — деяке ціле число. Не обмежуючи загальності міркувань, вважатимемо, що s ? 0, інакше поміняємо значення i та j між собою. З рівності $(a_i+i)-(a_j+j)=sn$ отримуємо $i+a_i=j+sn+a_j$. Послідовність має період n, тому $a_j=a_{j+sn}$. Звідси $i+a_i=j+sn+a_{j+sn}$. Отже, після *різних* елементів a_i та a_{j+sn} у траєкторії має йти один і той самий елемент a_{i+a_i}, тому в даному випадку послідовність не можна розкласти на траєкторії.

Розглянемо випадок, коли всі числа вигляду a_i+i ($i=1, \ldots, n$) мають різні лишки при діленні на n. Для зручності продовжимо нашу послідовність $\{a_i\}$ ліворуч, щоб вона була нескінченною в обидва боки й мала період $(a_1, a_2, \ldots, a_n)$. Для всіх цілих k числа a_k+k відрізняються одне від одного. Справді:

- якщо числа i та j дають різні лишки від ділення на n, то різні лишки даватимуть і числа a_i+i та a_j+j;
- якщо різниця i та j кратна n, то на $(i-j)$ відрізнятимуться a_i+i та a_j+j, бо $a_i=a_j$ внаслідок того, що послідовність має період довжини n.

Покажемо, що для кожного цілого m існує таке ціле k, що $a_k+k=m$. Нехай $m=pn+r$, де $0 \le r < n$. Згідно з припущенням (про те, що всі числа вигляду a_i+i мають різні лишки при діленні на n для $i=1, \ldots, n$), існує натуральне j, для якого сума a_j+j має лишок r при діленні на n. Нехай $a_j+j=qn+r$. Тоді маємо:

$$a_{j+(p-q)n}+j+(p-q)n=a_j+j+(p-q)n=qn+r+(p-q)n=pn+r=m.$$

Покладемо $k=j+(p-q)n$. Усі суми вигляду a_k+k різні (доведено вище), тому $k=j+(p-q)n$ буде єдиним, при якому $a_k+k=m$. Для кожного m проведемо уявну стрілку від елемента a_m до a_{m+a_m}. Траєкторії складатимуться з елементів, сполучених стрілками. Твердження про існування і єдиність k має наочне тлумачення: до кожного елемента входить a_m входить одна й лише одна така стрілка. Отже, у розглянутому випадку послідовність розкладається на декілька нескінченних в обидва боки траєкторій, що не перетинаються. Для початкової послідовності потрібні траєкторії можна отримати виключенням тих елементів, які стоять перед a_1.

Теорема 2. *Для розкладної послідовності, що має період* $(a_1, a_2, ..., a_n)$, *кількість траєкторій дорівнює* $(a_1+a_2+ \ldots +a_n)/n$.

Доведення. (Запропоноване автором завдання учасникам перед апеляцією). Розглянемо числову пряму з відміченими на ній точками, що відповідають цілим числам. Для кожного натурального i сполучимо напрямленим відрізком точки i та $i+a_i$. Кожна траєкторія відповідає сукупності відрізків, які вкривають пряму в один шар: у такій сукупності кінець кожного відрізка є початком наступного. Тому кількість траєкторій дорівнює кількості шарів із відрізків, якими вкрита числова пряма.

Підрахуємо цю кількість в інший спосіб. Позначимо:

$$d=\max\{a_1+1, a_2+2, \ldots, a_n+n\}.$$

Усі числа, які не менші за d, не можуть бути першими у будь-якій траєкторії. Отже, всі траєкторії починаються лівіше точки d. Тому кожна траєкторія вкриває проміжок $[d, d+n)$ відрізками або їхніми частинами загальної довжини n.

Для кожного $k = 1, 2, \ldots, n$ розглянемо відрізки, ліві кінці яких розташовано у точках вигляду $mn+k$, де m — довільне ціле число. Всі такі відрізки, за побудовою, мають довжину a_k.

При $a_k=1$ рівно один відрізок вигляду $[mn+k, mn+k+1]$ перетинається з $[d, d+n)$. Отже, сумарна довжина перетину (для фіксованого k) дорівнює 1. Якщо a_k перевищує 1, то множину відрізків:

$$\{[mn+k, mn+k+a_k] \mid m \in \mathbb{Z}\}$$

можна представити як об'єднання множин:

$[mn+k, mn+k+1]$,
$[mn+k+1, mn+k+2]$,
…
$[mn+k+a_k-1, mn+k+a_k]$, де $m \in \mathbb{Z}$.

Кожна така множина дає перетин довжини 1, тому всі множини разом дають перетин довжини a_k.

Сумарна довжина перетинів з $[d, d+n)$ за всіма $k=0, 1, \ldots, n-1$, враховуючи $a_0=a_n$, дорівнює $a_1+a_2+ \ldots +a_n$. З іншого боку, кожна траєкторія повністю і без накладень покриває проміжок $[d, d+n)$. Тому сумарна довжина перетину дорівнює добутку n і кількості траєкторій. Звідси кількість траєкторій рівна $(a_1+a_2+ \ldots +a_n)/n$.

Доведення. (Запропоноване автором завдання вже після проведення апеляції). Розглянемо числову пряму з відміченими на ній точками, що відповідають натуральним числам. Для кожного натурального k побудуємо напівінтервал $[k, k+a_k)$. Тоді для кожної траєкторії $(a_{i_1}, a_{i_2}, \ldots, a_{i_n}, \ldots)$ індексам її елементів відповідають напівінтервали $[i_1, i_1+a_{i_1})$, $[i_2, i_2+a_{i_2})$, …, $[i_n, i_n+a_{i_n})$, … За означенням траєкторії, $i_n+a_{i_n}=i_{n+1}$ для всіх натуральних n. Тому вказані напівінтервали не перетинаються, а їх об'єднанням є промінь, який починається в точці i_1 та напрямлений у бік зростання чисел.

Покажемо, що кожна з траєкторій, на які розкладається послідовність, починається лівіше точки $m = \max\{a_1+1, a_2+2, \ldots, a_n+n\}$. Для цього достатньо довести, що жодне число, яке більше або дорівнює m, не може бути індексом першого елемента деякої траєкторії. Розглянемо довільне $k \ge m$. За умовою, знайдеться j ($1 \le j \le n$), для якого a_j+j дає той самий лишок при діленні на n, що і число k. За вибором k, справджується нерівність $k \ge a_j+j$. Тому існує таке $p \ge 0$, що

$k=a_j+j+pn$. Оскільки послідовність має період n, також має місце рівність $k=a_{j+pn}+j+pn$. Отже, у траєкторії, що містить a_k, попереднім до нього елементом є a_{j+pn}. Тому всі траєкторії починаються лівіше точки m.

Отже, напівінтервал $[m, m+n)$ вкритий напівінтервалами $[k, k+a_k)$ у s шарів, де s — кількість траєкторій. Тобто, сумарна довжина перетинів усіх $[k, k+a_k)$ з напівінтервалом $[m, m+n)$ становить sn. Виразимо значення цієї довжини через елементи послідовності. Для кожного $k=1, 2, \ldots, n$ розглянемо всі напівінтервали вигляду $[k+pn, k+pn+a_k)$, де p — ціле невід'ємне число. Множину таких напівінтервалів позначимо через C_k. Доведемо, що сумарна довжина перетинів усіх напівінтервалів множини C_k з напівінтервалом $[m, m+n)$ дорівнює a_k. Кожен $[k+pn, k+pn+a_k)$ є об'єднанням напівінтервалів $[k+pn, k+pn+1)$, $[k+pn+1, k+pn+2)$, …, $[k+pn+a_k-1, k+pn+a_k)$, які не перетинаються та мають довжину 1. Нехай D_l позначає множину всіх напівінтервалів вигляду $[l+pn, l+pn+1)$, де p — ціле невід'ємне число. Тоді напівінтервали множини C_k є об'єднанням відповідних напівінтервалів із множин D_k, D_{k+1}, …, D_{k+a_k-1}. Для кожної D_l (де $l=k, k+1, \ldots, k+a_k-1$) рівно один напівінтервал довжини 1 потрапляє у $[m, m+n)$. Тому сумарна довжина перетину $[m, m+n)$ із напівінтервалами множини C_k — це кількість відповідних множин D_l. Отже, ця величина дорівнює a_k.

Якщо взяти сумарну довжину перетинів по всіх $k=1, 2, \ldots, n$, матимемо $a_1+a_2+\ldots+a_n$. Отже, $sn=(a_1+a_2+\ldots+a_n)$, тобто $s=(a_1+a_2+\ldots+a_n)/n$.

Алгоритм розв'язання задачі

Перевірити, чи мають всі числа a_1+1, a_2+2, …, a_n+n різні лишки при діленні на n.

Якщо відповідь ствердна, вивести число $(a_1+a_2+\ldots+a_n)/n$, інакше — число -1.

Зауваження. Сума $a_1+a_2+\ldots+a_n$ може перевищити 2^{32}. Тому при використанні цілого типу, що займає 32 біти, потрібно одночасно зберігати та оновлювати частку й остачу від ділення часткової суми $a_1+a_2+\ldots+a_i$ на n. Альтернатива — використовувати 64-бітні числа.

Примітка. Послідовності зі згаданими властивостями називаються *сайд-свопами* (або *сайт-свопами*). Їх використовують для запису комбінацій у жонглюванні. Кожна траєкторія відповідає кидкам окремого предмета. Про застосування сайд-свопів див. відеолекцію за посиланням: www.youtube.com/watch?v=GidTh8Nvh4.

VI. АВТОРСЬКІ РОЗВ'ЯЗАННЯ ЗАВДАНЬ III ЕТАПУ

1. Сходові числа

```
/* GCC */
#include <stdio.h>
int main()
{
 freopen(«numbers.in», «r», stdin);
 freopen(«numbers.out», «w», stdout);
 // Зчитування даних
 int n;
 scanf(«%d», &n);
 // Список усіх сходових чисел
 int numbers[11] = {1, 4, 16, 27, 256,
 3125, 46656, 65536, 823543, 16777216,
 387420489};
 // Порівняння кожного сходового числа
 // з n та обрахунок відповіді
 int ans = 0;
 for (int i = 0; i < 11; i++)
  if (numbers[i] <= n)
   ans++;
 // Виведення відповіді
 printf(«%d\n», ans);
}
```

2. Таблиця

```
/* GCC */
#define maxN 100 //Найбільше значення n
#define mod 101  // Число, за модулем якого
        // необхідно обчислити відповідь
#include <stdio.h>
int n, a[maxN][maxN];     // Вхідні дані
int cnt[maxN][maxN][maxN];// cnt[d][cl][cr]
// міститиме відповідь для пари комірок
// (cl,cr) з лівої та правої діагоналей d
// (нумерація діагоналей і комірок
// у програмі — з нуля)

// getValue обчислює (якщо цього ще не
// зроблено) та повертає значення комірки
// cnt[d][cl][cr]
int getValue(int d, int cl, int cr)
{
 if (cl < 0 || cl > d || cr < 0 || cr > d)
  return 0; // Якщо виходимо за межі
          // таблиці, повертаємо 0
 int v = cnt[d][cl][cr];
 if (v == -1) // Якщо значення комірки ще
   //не обчислено, обчислюємо його зараз
 {
  if (a[cl][d - cl] ==
  a[n - 1 - d + cr][n - 1 - cr])
  { // Якщо числа в комірках однакові
   if (d == 0) // Якщо це початкова
   // діагональ, то відповідь — 1
    v = 1;
   else // Інакше обчислюємо значення
   // згідно з рекурентним
   // співвідношенням (і за модулем 101)
    v = (getValue(d - 1, cl, cr)
      + getValue(d - 1, cl - 1, cr)
      + getValue(d - 1, cl, cr - 1)
      + getValue(d - 1, cl - 1, cr - 1))
      % mod;
  }
  else // Якщо числа в комірках різні,
  // відповідь для цієї пари комірок — 0
   v = 0;
  cnt[d][cl][cr] = v;
 }
 return v;
}

int main()
{
 freopen(«table.in», «r», stdin);
```

```
  freopen(«table.out», «w», stdout);

  // Зчитування даних
  scanf(«%d», &n);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf(«%d», &a[i][j]);

  // Ініціалізація значеннями -1
  // («ще не обчислено») для рекурсії
  // з запам'ятовуванням
  for (int d = 0; d < n; d++)
    for (int i = 0; i <= d; i++)
      for (int j = 0; j <= d; j++)
        cnt[d][i][j] = -1;

  //Виклик рекурсії та підрахунок відповіді
  //(доданків не більше ніж 100 і кожен

  //не перевищує 100, тому про переповнення
  // на цьому етапі можна не турбуватися)
    int ans = 0;
    for (int i = 0; i < n; i++)
      ans += getValue(n - 1, i, i);

    // Виведення відповіді за модулем 101
    printf(«%d\n», ans % mod);
}
```

3. Ламана

```
/* GCC */
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <map>

using namespace std;

//Повертає довжину (зі знаком) векторного
// добутку векторів p0p1 і p0p2
inline int product(pair<int, int> p0,
pair<int, int> p1, pair<int, int> p2)
{
 return (p1.first - p0.first)
    * (p2.second - p0.second)
    - (p1.second - p0.second)
    * (p2.first - p0.first);
}

// Якщо n=0, повертає 0. Інакше повертає
// одиницю з тим знаком, який має n
inline int sign(int n)
{
 if (n > 0)
   return 1;
 else if (n < 0)
   return -1;
 else
   return 0;
}

// Розраховує коефіцієнти рівняння прямої,
// на якій лежать точки a1 та a2
inline void coeffs(
pair<int, int> a1,
pair<int, int> a2,
long long &ca,
long long &cb,
long long &cc)
{
 ca = a1.second - a2.second;
 cb = a2.first - a1.first;
 cc = a1.first * a2.second
   - a1.second * a2.first;
}

// Повертає відстань від точки p до прямої
// ax+by+c, збільшену в sqrt(a*a+b*b) разів
inline int distance(long long a,
     long long b, long long c,
          pair<int, int> p)
{
 return abs(a*p.first + b*p.second + c);
}

// Визначає, чи перетинаються відрізки
// a1a2 та b1b2 (крім як, можливо, у своїх
// крайніх точках)
inline bool intersect(
pair<int, int> a1, pair<int, int> a2,
pair<int, int> b1, pair<int, int> b2)
{
 long long ca, cb, cc, da, db, dc;
 coeffs(a1, a2, ca, cb, cc);
 coeffs(b1, b2, da, db, dc);
 if (ca * db == cb * da) // Якщо прямі,
 // що містять відрізки, паралельні
 {
  if (ca * dc != cc * da
   || cb * dc != cc * db)
    // Якщо прямі не збігаються, відрізки
    // перетинатися не можуть
    return false;
  else   // Якщо прямі збігаються,
       // здійснюємо подальші перевірки
   {
    int l1, m1, l2, m2;
    if (a1.first != a2.first)
    // Якщо пряма не є вертикальною,
    // проектуємо відрізки на вісь
    // абсцис
    {
     l1 = min(a1.first, a2.first),
     l2 = max(a1.first, a2.first);
     m1 = min(b1.first, b2.first),
     m2 = max(b1.first, b2.first);
    }
    else // Якщо пряма є вертикальною,
    // проектуємо відрізки на вісь
    // ординат
    {
     l1 = min(a1.second, a2.second),
     l2 = max(a1.second, a2.second);
     m1 = min(b1.second, b2.second),
     m2 = max(b1.second, b2.second);
    }
    return !(m2 <= l1 || m1 >= l2);
    // Перевіряємо, чи перетинаються
```

```
      // проекції відрізків
     }
    }
    else
    { // Якщо прямі, що містять відрізки,
     // не є паралельними, підставляємо
     // кінці відрізків у рівняння прямої
     // іншого відрізка, щоб дізнатися,
     // по який бік від неї вони лежать
     int pb1 = sign(ca * b1.first
              + cb * b1.second + cc);
     int pb2 = sign(ca * b2.first
              + cb * b2.second + cc);
     int pa1 = sign(da * a1.first
              + db * a1.second + dc);
     int pa2 = sign(da * a2.first
              + db * a2.second + dc);
     switch (pb1 * pb2)
     { // Якщо точки b1 та b2 лежать по
      // один бік від прямої a1a2,
      // відрізки не перетинаються
      case  1:  return false;

      // Якщо точки b1 та b2 лежать по
      // різні боки від прямої a1a2,
      // відрізки перетинаються тоді й
      // лише тоді, коли точки a1 та a2
      // не лежать по один бік від прямої
      // b1b2
      case -1: return pa1 * pa2 <= 0;

      // Якщо одна з точок b1 та b2 лежить
      // на прямій a1a2, відрізки
      // перетинаються тоді й лише тоді,
      // коли точки a1 та a2 лежать
      // по різні боки від прямої b1b2
      case 0: return pa1 * pa2 == -1;
     }
    }
}

// Повертає в порядку обходу протии
// годинникової стрілки індекси всіх точок
// (вершин та, якщо є, проміжних точок), що
// опинилися на опуклій оболонці множини
// точок c; надає параметру line значення
// true або false залежно від того, чи всі
// точки опуклої оболонки (і множини c)
// лежать на одній прямій
vector<int> hull(const
vector<pair<int, int> > &c, bool &line)
{
 int n = c.size();
 vector<int> result; // Сюди буде записано
            // точки опуклої оболонки
 // Шукаємо точку з найменшою ординатою,
 // а серед таких — з найменшою абсциссою,
 // та додаємо її як першу точку опуклої
 // оболонки
 int minInd = 0;
 for (int i = 1; i < n; i++)
 if (c[i].second <  c[minInd].second
  || c[i].second == c[minInd].second
  && c[i].first   < c[minInd].first)
   minInd = i;
 result.push_back(minInd);
 line = true; // Початкове значення
 if (n == 1)
  return result;

 // used[i] матиме значення true тоді й
 // лише тоді, коли точку i вже було
 // додано до опуклої оболонки, а також
 // i!=minInd
 vector<bool> used;
 used.resize(n, false);

 int prevInd = minInd; // Точка, яку було
        // додано до опуклої оболонки
        // на попередньому кроці
 while (true)
 {
  int curInd = (prevInd == minInd)
         ? (minInd + 1) % n : minInd;
  // Початкова вершина-кандидат
  // на додавання

  for (int i = 0; i < n; i++)
  // Перебираємо всі точки у пошуку
  // найбільш підходящого кандидата
  // на додавання
  {
   if (!used[i] && i != prevInd
     && i != minInd)
   // Ці перевірки потрібні, щоб
   // не додати неправильну точку
   // на тій самій прямій
   {
    int p = product(c[prevInd],
          c[curInd], c[i]);
    // Нас цікавитиме орієнтація кута,
    // тобто знак векторного добутку
    // Якщо кут іде проти

    // руху годинникової стрілки або
    //якщо точки лежать на одній прямій
    // і ми досі розглядаємо початкову
    // вершину-кандидата (потрібно, щоб
    // не пропустити точки, які лежать
    // строго правіше від minInd) або
    // якщо точки лежать на одній прямій,
    // але та, яку розглядаємо зараз,
    // лежить ближче до prevInd,
    // ніж попередній кандидат
    if (p < 0 ||
       p == 0 && curInd == minInd ||
       p == 0 &&
  abs(c[prevInd].first-c[i].first)+
  abs(c[prevInd].second -c[i].second)
 < abs(c[prevInd].first-c[curInd].first)
 + abs(c[prevInd].second-c[curInd].second))
      { // Оновлюємо вершину-кандидата
       curInd = i;
      }
    }
   }
   if (curInd == minInd)// Якщо ми прийшли
   //назад до початкової вершини, виходимо
```

```
      break;

    // Додаємо знайдену точку
    // до опуклої оболонки
    used[curInd] = true;
    result.push_back(curInd);
    prevInd = curInd;
  }

  // Перебираючи по три послідовних точки,
  // перевіряємо, чи всі точки опуклої
  //  оболонки лежать на одній прямій
  for (int i = 0; i < result.size(); i++)
    if (product(
      c[result[i]],
      c[result[(i + 1) % n]],
      c[result[(i + 2) % n]]) != 0)
    {
      line = false;
      break;
    }
  return result;
}

// Додає до answer усі точки з прямої hull
// у порядку від точки з індексом pp до
// точки з індексом pq


void levelLine(const
vector<pair<int,int> > &hull, int pp, int pq,
vector<pair<int,int> > &answer)
{
  if (pp <= pq)
    for (int i = pp; i <= pq; i++)
      answer.push_back(hull[i]);
  else
    for (int i = pp; i >= pq; i—)
      answer.push_back(hull[i]);
}

// Додає до answer усі точки з опуклої
// оболонки hull у порядку від точки з
// індексом pp до точки з індексом pq,
// крім самої точки pq;
// параметру a надає значення індекса
// недоданої точки (тобто pq),
// а параметру b — значення індекса
// останньої доданої точки
void levelHull(
const vector<pair<int, int> > &hull,
int pp, int pq, int &a, int &b,
vector<pair<int, int> > &answer)
{
  int n = hull.size();
  if ((pp + 1) % n == pq)
  // Якщо точка pq йде відразу за
  // точкою pp, точки потрібно додати
  // у порядку зменшення індексу
  {
    for (int i = pp + (pq > 0 ? n : 0);
        i >= pq + 1; i—)
      answer.push_back(hull[i % n]);
    b = (pq + 1) % n;
    a = pq;
  }
  else // Інакше точки потрібно додати
  {    // у порядку збільшення індексу
    for (int i = pp + (pp > 0 ? 0 : n);
            i <= pq + n - 1; i++)
      answer.push_back(hull[i % n]);
    b = (pq + n - 1) % n;
    a = pq;
  }
}

// Рекурсивна функція, що додає до answer
//точки з індексами від pp до pq опуклої
// оболонки hulls[l] та заглиблюється;
// у lastLine приймає показник того,
// чи є опукла оболонка найглибшого рівня
// прямою (або точкою)
void level(const
vector<vector<pair<int,int>>> &hulls,
int l, int pp, int pq,
bool lastLine, vector<pair<int, int>>
&answer)
{
  int a, b;
  if (l == hulls.size() - 1)
    // Якщо ми розглядаємо
  { //опуклу оболонку найглибшого рівня
    if (lastLine) // Якщо ця оболонка є
      // прямою, виводимо відрізок
      levelLine(hulls[l], pp, pq, answer);
    else //Якщо ця оболонка не є прямою,
    {    //виводимо цю оболонку
      levelHull(hulls[l], pp, pq, a, b,
                    answer);
      answer.push_back(hulls[l][a]);
      // Виводимо останню точку
    }
  }
  else
  if (l == hulls.size() - 2 && lastLine)
  // Якщо ми розглядаємо оболонку
  // передостаннього рівня, а
  // остання оболонка — пряма
  {// Виводимо оболонку
   // (крім останньої точки)
    levelHull(hulls[l], pp, pq, a, b,
                  answer);
    int lnmax = hulls[l + 1].size() - 1;
    // Індекс останньої точки на
    // прямій найглибшого рівня
    // Перевіряємо, при якому з двох
    // варіантів сполучення точок і
    // відповідно порядку обходу точок на
    // прямій не буде жодного перетину,
    // та заглиблюємось у відповідний
    //  варіант
    if  (!intersect(
    hulls[l][a], hulls[l + 1][0],
    hulls[l][b], hulls[l + 1][lnmax]) &&
    !intersect(
    hulls[l][a],    hulls[l + 1][0],
    hulls[l + 1][0], hulls[l + 1][lnmax]) &&
    !intersect(
```

```
  hulls[l + 1][0], hulls[l + 1][lnmax],
  hulls[l][b], hulls[l + 1][lnmax]))
  {
   level(hulls, l + 1, lnmax, 0,
          lastLine, answer);
  }
  else

  {
   level(hulls, l + 1, 0, lnmax,
          lastLine, answer);
  }

  answer.push_back(hulls[l][a]);
  // Виводимо останню точку оболонки
  // поточного рівня
}
else// Якщо існує опукла оболонка
{  // наступного рівня і вона не є прямою
  levelHull(hulls[l],pp,pq, a, b, answer);
  // Виводимо опуклу оболонку поточного
  // рівня (крім останньої точки)
  int nsize = hulls[l + 1].size();
  // Розмір оболонки наступного рівня
  // Шукаємо найближчу до прямої ab
  // точку на оболонці наступного рівня
  long long ca, cb, cc;
  coeffs(hulls[l][a], hulls[l][b],
                ca, cb, cc);
  int mini = 0, mind = distance
  (ca, cb, cc, hulls[l + 1][mini]);
  // Поточна найближча точка та
  // (масштабована) відстань до неї
  for (int i = 1; i < nsize; i++)
   {
    int cur = distance(ca, cb, cc,
            hulls[l + 1][i]);
    if (cur < mind)
    {
     mind = cur;
     mini = i;
    }
   }
   // np — найближча точка, а nq та nr
   // — її сусіди, яких вибираємо так,
   // щоб трійка точок nr, np, nq йшла
   // на опуклій оболонці наступного
   // рівня у тому самому порядку, в
   // якому на поточній оболонці йде
   // пара точок a, b (за або проти
   // годинникової стрілки)
   int np = mini, nq, nr;
   if (b == (a + 1) % hulls[l].size())
   {
    nq = (np + 1) % nsize;
    nr = (np - 1 + nsize) % nsize;
   }
   else
   {
    nq = (np - 1 + nsize) % nsize;
    nr = (np + 1) % nsize;
   }
   //Вибираємо той із двох варіантів
   //сполучення точок (і відповідно
   //порядку обходу наступної оболонки),
   //що не призводить до перетинів, та
   //заглиблюємось у відповідний варіант
   if (!intersect(
    hulls[l][a], hulls[l + 1][np],
    hulls[l][b], hulls[l + 1][nq]))
     level(hulls, l + 1, nq, np,
          lastLine, answer);
   else
     level(hulls, l + 1, np, nr,
          lastLine, answer);

   answer.push_back(hulls[l][a]);
   // Виводимо останню точку оболонки
   // поточного рівня
  }
}

int main()
{
 freopen(«polyline.in», «r», stdin);
 freopen(«polyline.out», «w», stdout);

 vector<pair<int, int> > curc;
 // У curc зберігатимуться усі точки,
 // ще не додані до опуклих оболонок
 // map<pair<int, int>, int> m;
 // Індексний масив, що дозволятиме за
 // точкою відновити її номер (починаючи
 // з одиниці) у початковому наборі
 // Зчитування даних
 int n;
 scanf(«%d», &n);
 for (int i = 0; i < n; i++)
 {
  int x, y;
  scanf(«%d%d», &x, &y);
  curc.push_back(make_pair(x, y));
  m[make_pair(x, y)] = i + 1;
 }

 vector<vector<pair<int, int> > > hulls;
  // Масив опуклих оболонок в порядку від
  // зовнішньої до внутрішньої
 bool line; // Після формування всіх
  // опуклих оболонок показуватиме, чи
  // внутрішня оболонка виявилась прямою
  // (або точкою)
  // Формуємо опуклі оболонки
 while (curc.size() > 0)
 {// Поки в наборі залишаються точки
  vector<int> h = hull(curc, line);
   // Опукла оболонка поточного набору
   // (містить індекси точок)
  vector<pair<int, int> > addc, newc;
   // Опукла оболонка поточного набору
   // (містить самі точки) і точки, які
   //  до оболонки не увійшли
  vector<bool> used;   // used[k] матиме
  // значення true тоді й лише тоді, коли
  // точка з індексом k увійшла до опуклої
  // оболонки
  used.resize(curc.size(), false);
   // Ініціалізація масиву used
```

```
    for (int i = 0; i < h.size(); i++)
     {
      addc.push_back(curc[h[i]]);
      used[h[i]] = true;
     }
    for (int i = 0; i < curc.size(); i++)
     if (!used[i])
      newc.push_back(curc[i]);

    hulls.push_back(addc);
    curc = newc;
  }
  if (hulls.size() == 1 && line)
  // Якщо всі точки лежать на одній прямій,
  // потрібної ламаної не існує
    printf(«0»);
  else // Якщо точки не лежать на одній
  {   // прямій, відповідь існує
    vector<pair<int, int> > answer;
    level(hulls, 0, 0, 1, line, answer);
    // Визначаємо та записуємо відповідь
    // у масив answer
    for (int i = 0; i < n; i++)
     {// Виводимо відповідь
      if (i > 0)
       printf(« «);
      printf(«%d», m[answer[i]]);
     }
  }
  printf(«\n»);
}
```

4. Дільники

```
{ Free Pascal }
const n_=4;
    p:array[1..n_] of byte=(2,3,5,7);
var c: char; o:text;
 a: array[0..1,1..31] of byte;
answer: qword;
b,i0,i1,j,k,l,n,npj: byte;
BEGIN
assign(o,'divisors.in');
reset(o);
i0:=0;
i1:=1;
 n:=0;
while not seekeoln(o) do begin
 inc(n);
 read(o,c);
 case c of
 '0': a[i0,n]:=0;
 '1': a[i0,n]:=1;
 '2': a[i0,n]:=2;
 '3': a[i0,n]:=3;
 '4': a[i0,n]:=4;
 '5': a[i0,n]:=5;
 '6': a[i0,n]:=6;
 '7': a[i0,n]:=7;
 '8': a[i0,n]:=8;
 '9': a[i0,n]:=9;
 end                end;
close(o);
assign(o,'divisors.out');
rewrite(o);
answer:=1;
for j:=1 to n_ do              begin
 npj:=0;
 b  :=0;
 while  0=b do                 begin
  if  (a[i0,1]<p[j]) and (1<n) then begin
   b:=a[i0,1]*10+ a[i0,2];
   k:=2                        end
                    else begin
   b:=a[i0,1];
   k:=1                        end;
  for l:=k to n do             begin
   a[i1,l-k+1]:=b div p[j];
   b:=10*(b mod p[j])+a[i0,l+1]    end;
  if b=0 then begin
   n :=n-k+1;
   i0:=1-i0;
   i1:=1-i1;
   a[i0,n+1]:=0;
   inc(npj)   end              end;
 answer:=answer*(npj+1)            end;

writeln(o,answer);
close(o)                       END.
```

5. Печери

```
{ Free Pascal }
const n=10;  nb_=1000000;
var a: array[1..n,1..n] of boolean;

b: array[0..1,1..n*nb_] of qword;  {масиви
      кількостей монет після попереднього
            і після поточного ходів}

nb: array[0..1,0..n]  of longint;
  {вказівники на масив b - де закінчуються
дані про кількості монет після попереднього
і після поточного ходів для певної печери}

jc: array[1..n] of longint; {поточні номери
 кількостей монет після попереднього ходу,
 які враховують для знаходження кількості
          монет після поточного ходу}

c: array[1..n] of byte;     {номери печер,
        з яких є хід у поточну печеру}

king: array[1..nb_] of qword;
              {«королівські числа»}
o: text;              {вхідний файл}
d,max: qword;
l,jl,jct: longint;
            {номери у масивах b, nb}
i0,          {після попереднього ходу}
i1,            {після поточного ходу}
i,j,k,nc,m,nm: byte;
procedure step; var l: byte; jct: longint;
                       BEGIN
nb[i1,j]:=nb[i1,j-1];
repeat
if nc=1 then                   begin
 for jct:=jc[1] to nb[i0,c[1]] do begin
 inc (nb[i1,j]);
 b[i1,nb[i1,j]]:=d+b[i0,jct]      end;
 nc:=0                       end
```

```
        else              begin
 max:=0;
 for l:=1 to nc do
 if  max<=b[i0,jc[l]]
 then max:=b[i0,jc[l]];
 inc (nb[i1,j]);
 b[i1,nb[i1,j]]:=d+max;
 l:=0;
 repeat
  inc(l);
  if max=b[i0,jc[l]]
  then inc(jc[l]);
  if nb[i0,c[l]]<jc[l] then     begin
    c[l]:= c[nc];
   jc[l]:=jc[nc];
   dec(l);
   dec(nc)                  end;
 until (l=nc)                 end;
until nc=0                    END;
                        BEGIN
assign(o,'caves.in');
 reset(o);
readln(o,m,l);
for j:=1 to n do          begin
 for k:=1 to n do         begin
  read(o,d);
  a[j,k]:=(d=1)           end;
            readln(o) end;
for j:=1 to n do begin
 read(o,b[0,j]);
  nb[0,j]:=j     end;
readln(o);
nb[0,0]:=0;
nb[1,0]:=0;
i0:=0;
i1:=1;
for k:=2 to m do    begin
 for j:=1 to n do   begin
  read(o,d);
  nc:=0;
  for i:=1 to n do
  if a[j,i] then   begin
   inc(nc);
   jc[nc]:=nb[i0,i-1]+1;
    c[nc]:=i       end;
 step;
 i:=i;
          end;
 readln(o);
 i0:=1-i0;
 i1:=1-i1          end;
for jl:=1 to l do read(o,king[jl]);
readln(o);
 close(o);
assign(o,'caves.out');
 rewrite(o);
 j:=1;
nc:=n;
for k:=1 to n do    begin
 c[k]:=k;
 jc[k]:=nb[i0,k-1]+1 end;
d:=0;
step;
jc[1]:=1;
jct:=1;
repeat
 while (king[jct]>b[i1,jc[1]]) and (jct<l)
                  do inc(jct);
 if   (king[jct]=b[i1,jc[1]]) then begin
  b[i1,jc[1]]:=0;
  inc(jc[1])                  end;
 while (king[jct]<b[i1,jc[1]])
     and (jc[1]<nb[i1,1]) do inc(jc[1]);
 if   (king[jct]=b[i1,jc[1]]) then begin
  b[i1,jc[1]]:=0;
  inc(jc[1])                  end;
until (jct>=l) or (jc[1]>nb[i1,1]);
d:=0;
for jct:=1 to nb[i1,1] do
if (0<b[i1,jct]) then inc(d);
write(o,d);
if (d=0) then writeln(' 0 0') else begin
 jct:=1;
 while 0=b[i1,jct] do inc(jct);
 write(o,' ',b[i1,jct]);
 jct:=nb[i1,1];
 while 0=b[i1,jct] do dec(jct);
 write(o,' ',b[i1,jct]);
 writeln(o)                   end;
close(o);
END.
```

6. Розкладна послідовність

```
{ Free Pascal }
var r: array[0..100000] of boolean;
{Чи дане число має вигляд (a_j+j) mod n?}

j, n: longint;
a, s: qword;
  o: text;
BEGIN
assign(o,'decomp.in');
reset(o);
s:=0;
readln(o,n);
for j:=0 to n-1 do r[j]:=false;
for j:=1 to n do    begin
 read(o,a);
 s:=s+a;
 r[(a+j) mod n]:=true end;
close(o);
assign(o,'decomp.out');
rewrite(o);
j:=0;
while r[j] and (j<=n-1) do inc(j);
if (j<n) then writeln(o,'-1')
      else writeln(o,s div n);
close(o);
END.
```