ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2014–2015 НАВЧАЛЬНОМУ РОЦІ

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

Закінчення, початок у №4, №5 за 2015 рік

V. ІДЕЇ РОЗВ'ЯЗАННЯ ЗАВДАНЬ III ЕТАПУ

1. Дивовижне число

Якщо число d має простий дільник, більший за 10, воно не може виявитися добутком цифр жодного натурального числа. В іншому разі d можна подати як добуток степенів простих чисел 2, 3, 5 і 7: $d=2^w\times3^x\times5^y\times7^z$, де степені w, x, y, z невід'ємні цілі. Тоді шукане число міститиме w двійок, x трійок, y п'ятірок, z сімок і $d-2w-3x-5y-7z$ одиниць та матиме вигляд 77...755...533...322...211...1 (якихось із цифр 7, 5, 3, 2 чи 1 може й не бути, якщо відповідна кількість дорівнює нулю). Справді: число повинно містити рівно z сімок і рівно y п'ятірок, адже інші ненульові цифри не дають множників 7 і 5. Якщо число містить цифру 4, то її можна замінити двома цифрами 22, збільшивши число, але при цьому не змінивши ні суми, ні добутку його цифр. Аналогічно цифру 6 можна замінити цифрами 321, цифру 8 — цифрами 22211, а 9 — 33111. Відповідно, в найбільшому числі, що задовольняє умову задачі, є сенс використовувати лише двійки і трійки, причому рівно w та x таких цифр відповідно. Вже маємо добуток цифр d, але сума на даний момент становить $2w+3x+5y+7z$. Єдиний спосіб отримати з неї суму d, не змінивши при цьому добутку, — додати рівно $d-2w-3x-5y-7z$ одиниць. Це число невід'ємне, адже сума довільного набору натуральних чисел, більших за 1, не може перевищувати добутку цих же чисел (пропонуємо довести це самостійно).

2. Черевички на підборах

Спершу числа в кожній з двох послідовностей потрібно впорядкувати за неспаданням, скориставшись одним з ефективних методів сортування. Далі, починаючи з перших елементів, потрібно рухатись зліва направо одночасно по двох послідовностях. На кожному кроці:

- Якщо поточний елемент першої послідовності менший за поточний елемент другої послідовності, перейти до наступного елемента першої послідовності.
- Якщо поточний елемент другої послідовності менший за поточний елемент першої послідовності, перейти до наступного елемента другої послідовності.
- Якщо поточні елементи послідовностей однакові, додати до лічильника-відповіді 1 і перейти до наступних елементів в обох послідовностях.
- Якщо хоча б в одній послідовності закінчилися елементи, вийти.

Цей процес подібний до етапу злиття відсортованих фрагментів масиву в алгоритмі сортування злиттям, який, до речі, і можна використати в цій задачі як ефективний метод сортування.

Додамо, що, коли значення самих чисел не перевищують 1000, можна скористатися іншим підходом: у k-ту комірку масиву a, $1\le k\le1000$, записати кількість чисел k у першій послідовності (це можна зробити за єдиний прохід) і аналогічно в k-ту комірку масиву b записати кількість чисел k у другій послідовності. Таким чином,

$$a_1+a_2+...+a_{1000}=b_1+b_2+...+b_{1000}=n.$$

Відповіддю у такому разі буде сума

$$\min\{a_1, b_1\}+\min\{a_2, b_2\}+...+\min\{a_{1000}, b_{1000}\},$$

де $\min\{x, y\}$ позначає менше з двох чисел x, y.

3. Два квадрати

Розв'язати задачу можна за допомогою кількох підходів. Один із них такий.

Спершу визначимо розташування одного квадрата:

1. Повним перебором знайдемо чорну клітинку A, і зліва, і зверху від якої клітинки або відсутні, або білі. Така клітинка обов'язково знайдеться, бо інакше ми могли б переходити по чорних клітинках ліворуч або вгору нескінченно довго. Якщо клітинок з такою властивістю більше ніж одна, нам потрібна найлівіша з них. Знайдена клітинка A обов'язково є лівою верхньою клітинкою принаймні одного з квадратів.

2. Порахуємо кількість чорних клітинок у тому ж рядку праворуч від знайденої і кількість чорних клітинок у тому ж стовпці під знайденою. Ці кількості можуть бути однаковими або різними, але в будь-якому разі менша з них задає довжину сторони квадрата з лівою верхньою вершиною в клітинці A.

3. За знайденою інформацією про ліву верхню клітинку і сторону квадрата повністю визначимо розташування квадрата.

Тепер розглянемо такі варіанти:

- Якщо крім клітинок першого квадрата немає жодних інших зафарбованих клітин, другий квадрат, очевидно, має те саме розташування, що й перший.
- Якщо є хоча б одна чорна клітинка за межами першого квадрата, то за його межами є й принаймні одна вершина другого квадрата, яку можна визначити як клітинку, з двох непротилежних боків від якої клітинки або білі, або відсутні (тобто зліва і зверху, справа і зверху, справа і знизу або зліва і знизу). З цієї вершини виходить принаймні одна сторона, про яку напевне можна сказати, що вона не перетинає перший квадрат. За цими вер-

шиною і стороною можна повністю відновити розташування другого квадрата.

- Інакше, тобто якщо всі чорні клітинки лежать усередині першого квадрата, причому хоча б одна лежить строго всередині, зробимо таке. Розглянемо множину горизонталей та вертикалей, на яких лежить принаймні одна чорна клітинка, яка точно не належить першому квадрату. Включимо до цієї множини горизонталь, на якій лежить верхня сторона першого квадрата тоді й лише тоді, коли на наступній за нею горизонталі є хоча б одна чорна клітинка, що лежить строго всередині першого квадрата, але немає двох сусідніх таких клітинок. Аналогічно зробимо з лініями, на яких лежать нижня, ліва та права сторони першого квадрата. Тоді шуканий другий квадрат утворюватиметься крайніми горизонталями та крайніми вертикалями побудованої множини ліній.

Зауважимо, що з поданого розв'язання випливає, що можливе розташування квадратів завжди єдине.

4. Схил

Позначимо точки з цвяхами через $P_1, P_2, \ldots, P_n$. Згідно з умовою, при $i=1, 2, \ldots, n$ координатами точки P_i є x_i та y_i. Для вказаних точок запровадимо поняття порядку таким чином: будемо вважати, що точка P_i менша за точку P_j, якщо $y_i<y_j$, або якщо одночасно $y_i=y_j$ та $x_i<x_j$. Останнє будемо записувати як $P_i<P_j$. Легко пересвідчитися, що з висловлювань $P_i<P_j$ і $P_j<P_k$ випливає $P_i<P_k$.

Перенумеруємо дані точки в порядку зростання: $P_1<P_2<\ldots<P_n$. Це можна зробити за допомогою ефективного алгоритму впорядкування. Автор обрав метод злиття, як і в розв'язанні задачі «Істотні інверсії» 2013 року. Хоча для більшості учасників знайомішим буде швидкий метод Хоара.

Згідно з поданим принципом сортування, у впорядкованій послідовності $P_1, P_2, \ldots, P_n$ між двома точками з однаковою ординатою y можуть бути лише точки з цією самою ординатою. Тому впорядкована послідовність, $P_1, P_2, \ldots, P_n$ складається з блоків точок $B_1, B_2, \ldots, B_k$, де кожен блок містить точки з однаковою ординатою y, а в різних блоках точки мають різні ординати. У довільному блоці B_j точки розташовано за зростанням абсциси x.

Зауважимо, що у випадку $y_i<y_j<y_k$ пара точок P_i та P_k не може бути відповіддю:

- якщо точка P_j не лежить на прямій P_iP_k, то пара (P_i, P_j) або пара (P_j, P_k) дає менший, але також не нульовий кут нахилу до горизонталі;
- якщо точка (x_j, y_j) лежить на вказаній прямій, то пара (P_i,P_j) прийнятніша за пару (P_i, P_k) згідно з додатковими умовами вибору відповіді.

Тому достатньо переглянути лише *пари точок із сусідніх блоків*. На роль відповіді претендують такі пари P_i та P_j, де кожна точка є першою або останньою в своєму блоці. Тому всього потрібно порівняти не більше $4n$ пар. Для достатньо великих n це потребує меншої кількості операцій, ніж процес впорядкування точок.

Ефективність всього розв'язання збігається з ефективністю методу впорядкування і становить $O(n\log_2 n)$.

5. Каса

Довільному розташуванню дітей у черзі поставимо у відповідність послідовність чисел a_j:

- $a_j=-1$, якщо j-та дитина має банкноту 10 гривень;
- $a_j=+1$, якщо j-та дитина має банкноту 20 гривень.

Розглянемо суму $S_j=p+a_1+a_2+\ldots+a_j$, що є різницею між кількостями банкнот по 20 і 10 гривень після купівлі квитків першими j дітьми. *Для наочності* у прямокутній системі координат xOy розглянемо ламану з *послідовними* вершинами $A_j=(j, S_j)$ при $j=0, 1, 2, \ldots, m+n$. Ця ламана сполучає точку $A_0=(0, p)$ з точкою $A_{n+m}=(m+n, p+n-m)$ і проходить через точки A_1, $A_2, \ldots, A_{m+n-1}$. Називатимемо таку ламану траєкторією, яка відповідає даному способу розташування покупців у черзі (точніше, даному способу надходження банкнот у касу).

Кожна траєкторія складається з $m+n$ похилих відрізків, n з яких піднімаються вгору — їхній кут нахилу до осі абсцис Ox складає $+45^\circ$, а інші m відрізків опускаються донизу — їхній кут нахилу до осі абсцис Ox складає -45°. Якщо вказати номери тих відрізків, які піднімаються вгору, то тим самим траєкторію буде визначено повністю. Тому загальне число траєкторій — число послідовностей банкнот — дорівнює

$$C^n_{m+n}=\binom{m+n}{n}=\frac{(m+n)}{m!n!}.$$

Траєкторії, що відповідають тим способам розташування дітей, при яких жодна дитина не чекає здачі, не перетинають пряму $y=-1$ і не дотикаються до неї. Для того щоб підрахувати число таких траєкторій, поставимо у відповідність кожній *непридатній* траєкторії T, яка *перетинає або дотикається* до прямої $y=-1$, нову траєкторію T' за таким правилом: до *першої* точки дотику з прямою $y=-1$ траєкторія T' збігається з T, а далі T' є образом траєкторії T при симетрії відносно прямої $y=-1$. Інакше кажучи, абсциси вершин ламаної залишаються тими самими, а ординати зазнають таких змін:

$$y'=2\cdot(-1)-y=-2-y.$$

Всі траєкторії T' закінчуються в точці $A'_{m+n}=(m+n, m-n-p-2)$, яка є образом точки A_{m+n} при симетрії відносно прямої $y=-1$. Встановлена відповідність є взаємно однозначною. Кількість непридатних траєкторій дорівнює загальній кількості ламаних, які сполучають $A_0=(0, p)$ з A'_{m+n}. Якщо така ламана складається з x відрізків, що йдуть донизу, й y відрізків, що йдуть угору, то

$$x+y=m+n;$$
$$p+y-x=m-n-p-2,$$

звідки $y=m-p-1$. При невід'ємному y кількість траєкторій з A_0 в A'_{m+n} дорівнює

$$C^y_{x+y}=C^{m-p-1}_{m+n}=\binom{n+m}{m-p-1}.$$

Інакше, тобто при $m-p-1<0$, що еквівалентно $m\le p$, ця кількість дорівнює 0. Остаточно маємо:

- при $m\le p$ шукана кількість розташувань дорівнює

$$C^n_{n+m}=\binom{n+m}{n};$$

- при $m>p$ шукана кількість розташувань дорівнює

$$C_{n+m}^{n}-C_{n+m}^{m-p-1}=\binom{n+m}{m}-\binom{n+m}{m-p-1}.$$

Таким чином, задачу зведено до обчислення біномних коефіцієнтів:

$$C_{n+m}^{n}=\frac{(m+n)!}{m!n!}=\frac{(m+n)(m+n-1)\ldots(n+1)}{m!}=$$
$$=\frac{(m+n)(m+n-1)\ldots(m+1)}{n!},$$

де з останніх двох виразів бажано вибрати той, що має менший знаменник. Потім, використавши алгоритм Евкліда, скоротити на найбільші спільні дільники множники чисельника та знаменника, щоб *подати знаменник добутком одиниць* і знайти чисельник. При $r>0$ після кожного множення потрібно замінювати добуток на лишок від ділення на r, щоб не вийти не межі базового типу (int64 для Pascal).

6. Гра

Свого часу цю задачу для випадку $g=2$ і максимального номеру фішки 17 було запропоновано на ІІІ (міському) етапі олімпіади у 2004 році. Задачу з такими обмеженнями можна було розв'язати, використавши аналіз графа гри «з кінця». Але для запропонованих у даній умові обмежень щодо кількості фішок та часу виконання такий підхід не годиться для більшості тестів.

Будь-яка позиція розглядуваної гри є об'єднанням рядів послідовних фішок (без розриву). Інакше кажучи, гра з довільної позиції є сумою ігор з ряду послідовних фішок. Це явна вказівка на те, що повне розв'язання передбачає використання чисел Шпраґе — Гранді (оцінок позицій, що дорівнюють нулю для програшних позицій). В авторському розв'язанні їх знайдено для рядів послідовних фішок при зростанні довжини ряду:

- якщо довжина ряду не перевищує g, то вона і є числом Шпраґе — Гранді для позиції, що містить лише цей ряд;
- для решти позицій їхню оцінку потрібно знайти, керуючись тим, що:
  - *оцінка позиції — це найменше невід'ємне ціле число, відмінне від оцінок позицій, у які є хід з даної позиції;*
  - *оцінка суми позицій є побітною сумою* (без перенесень у старші розряди, див. операцію xor для мови Pascal) *оцінок позицій-доданків.*

Позиція є програшною тоді й лише тоді, коли її оцінка дорівнює нулю. Детальний і замкнений виклад теорії опубліковано, наприклад, на сайті http://kievoi.ippo.kubg.edu.ua/kievoi/lectures/spragrun.html.

VI. АВТОРСЬКІ РОЗВ'ЯЗАННЯ ЗАВДАНЬ ІІІ ЕТАПУ

1. Дивовижне число

```
/* GCC */
#include<stdio.h>
int main()
{
freopen("awesome.in", "r", stdin);
freopen("awesome.out", "w", stdout);
int d, s;
scanf("%d", &d);
  s = d; // s позначатиме, скільки залишилося
  // добрати, щоб вийшла правильна
  // сума цифр, а d — добуток
int digits[4] = {7, 5, 3, 2}; // Перебираємо
  // цифри саме в такому порядку
int counts[4] = {0, 0, 0, 0}; // Кількості відповідно
  // цифр 7, 5, 3, 2, які рахуємо далі
for(int i = 0; i <4; i++)
while(d % digits[i] == 0) // Поки
    // d ділиться на поточну цифру:
{
      d /= digits[i]; // Ділимо d на поточну цифру
s -= digits[i]; // Корегуємо значення суми цифр,
          // яку залишилося добрати
counts[i]++; // Збільшуємо лічильник
            // відповідної цифри на 1
if(d >1) // Якщо виявилося, що в числа
   // є простий дільник, більший за 7
printf("0");
else
{
for(int i = 0; i <4; i++)
     // Виводимо цифри 7, 5, 3, 2
     // відповідну кількість разів
for(int j = 0; j <counts[i];
              j++)
printf("%d", digits[i]);
for(int j = 0; j < s; j++)
printf("1"); // Доповнюємо число одиницями так,
       // щоб сума цифр вийшла правильною
}
printf("\n");
return 0;
}
```

2. Черевички на підборах

```
/* GCC */
#define maxN 200000 // Найбільше можливе
           // значення n
#include<stdio.h>
#include<algorithm> // Ця бібліотека
         // містить функцію сортування
int a[maxN], b[maxN]; // Перша та друга
// послідовності. Масиви оголошено поза
// тілом програми, щоб уникнути
// переповнення стека
int main()
{
freopen("heels.in", "r", stdin);
freopen("heels.out", "w", stdout);
int n;
// Зчитуємо дані:
scanf("%d", &n);
for(int i = 0; i < n; i++)
scanf("%d", &a[i]);
for(int i = 0; i < n; i++)
scanf("%d", &b[i]);
std::sort(a, a + n); // Сортуємо всі
         // члени першої послідовності
std::sort(b, b + n); // Сортуємо всі
         // члени другої послідовності
int answer = 0; // Лічильник-відповідь
int ac = 0, bc = 0; // Вказівники на
    // поточні елементи першої та другої
    // послідовностей відповідно
while(ac< n &&bc< n) // Поки ми не вийшли за
  // межі жодної з двох послідовностей:
```

```
{
// Якщо поточні члени різні, переходимо
    // до наступного в тій послідовності,
    // де член менший, а інакше додаємо до
    // лічильника-відповіді одиницю
    // і переходимо до наступних членів
    // в обох послідовностях:
if(a[ac] < b[bc])
ac++;
elseif(a[ac] > b[bc])
bc++;
else
{
answer++;
ac++;
bc++;
    }
  }
// Виводимо відповідь:
printf("%d\n", answer);
return0;
}
```

3. Два квадрати

```
/* GCC */
#definemaxN1000 // Найбільше можливе
          // значення n
#include<stdio.h>
boola[maxN + 2][maxN + 2];
// Беремо аркуш з «полями» по одній клітин-
// ці в кожен бік (так зручніше буде реалі-
// зовувати різноманітні перевірки).
// a[i][j] == true означатиме,
// що клітинка зафарбована
intmain()
{
freopen("squares.in", "r", stdin);
freopen("squares.out", "w", stdout);
intn;
scanf("%d\n", &n);
// Заповнюємо «поля» навколо аркуша
  // значенням «незафарбовано»:
for(inti = 0; i < n + 2; i++)
    a[0][i] = a[i][0] = a[n + 1][i] =
        a[i][n + 1] = false;
// Зчитуємо сам аркуш:
for(inty = 1; y <= n; y++) // Рядки —
    // це координата y, а стовпці — x
for(intx = 1; x <= n; x++)
    {
charc;
scanf(x < n ? "%c "
: "%c\n", &c);
// Не забуваємо зчитувати пробіл або
      // перенесення рядка після кожного символу
a[x][y] = (c == '#');
       // Клітинка зафарбована,
       // якщо відповідний символ — #
}
// Шукаємо найлівішу з клітинок, що є
  // лівими верхніми вершинами квадратів:
ints1_left = 0, s1_top = 0; // У май-
  // бутньому — ліва та верхня межі пер-
  // шого квадрата (координати лівої вер-
  // хньої вершини). Нулі —індикатор то-
  // го, що клітинку поки що не знайдено
for(intx = 1; x <= n; x++) // Ідемо
{           // зліва направо:
for(inty = 1; y <= n; y++)
// Ідемо зверху вниз:
if(a[x][y] && ! a[x − 1][y]
&& ! a[x][y − 1])
      { // Якщо клітинку з відповід ними
        // властивостями знайде-но, запам'я-
        // товуємо її та виходимо з циклу:
s1_left = x;
        s1_top = y;
break;
      }
if(s1_left >0) // Якщо клітинку знайдено,
       // виходимо також і з зовнішнього циклу
break;
  }
// Визначаємо ліву або верхню сторону
  // квадрата з вершиною у знайденій
  // клітинці:
ints1_right = s1_left,
    s1_bottom = s1_top; // У майбутньому —
    // права та нижня межі першого квадрата
    // квадрата (координати правої нижньої
    // вершини)
while(a[s1_right + 1][s1_top] &&
      a[s1_left][s1_bottom + 1])
  { // Поки і праворуч, і вниз клітинки чорні,
    // продовжуємо «розширювати» квадрат:
s1_right++;
    s1_bottom++;
  }
// Визначаємо, чи є чорні клітинки строго
  // всередині та строго зовні квадрата:
boolinternal = false; // Чи є клітинки всередині
boolexternal = false; // Чи є клітинки зовні
for(intx = 1; x <= n; x++) // Проходимо
      // по всіх клітинках аркуша:
for(inty = 1; y <= n; y++)
if(a[x][y]) // Якщо клітинка
{       // чорна:
if(x > s1_left &&
          x < s1_right &&
          y > s1_top &&
          y < s1_bottom)
internal = true;
// Клітинка лежить строго всередині квадрата
elseif(x < s1_left ||
            x > s1_right ||
            y < s1_top ||
            y > s1_bottom)
external = true;
// Клітинка лежить строго зовні квадрата
}
ints2_left, s2_top,
    s2_right, s2_bottom; // У майбут-
    // ньому — ліва, верхня, права та
    // нижня межі другого квадрата
    // (координати відповідних двох
    // вершин цього квадрата)
if(external) // Якщо є зафарбовані
      // клітинки поза першим квадратом:
{
for(intx = 1; x <= n; x++)
    {
```

```
boolfound = false; // Чи знайдено вже клітинку,
        // що є вершиною другого квадрата, яка
        // лежить поза межами першого,
        // а отже, визначено розташу-
        // вання другого квадрата
for(inty = 1; y <= n; y++)
if(a[x][y] && (x < s1_left
                ‖ x > s1_right
                ‖ y < s1_top
                ‖ y > s1_bottom))
        { // Якщо ми натрапили на чорну клітинку
          // поза межами першого квадрата:
boolisTopLeft =
                ! a[x – 1][y] &&
                ! a[x][y – 1];
// Чи є вона лівою
                // верхньою вершиною
boolisBottomRight =
                ! a[x + 1][y] &&
                ! a[x][y + 1];
// Чи є вона правою нижньою вершиною
          // Решту випадків можна не розглядати,
          // так само як не слід хвилюватися через
          // те, що зліва чи зверху від лівої верхньої
         // вершини або справа чи знизу від правої
         // нижньої вершини буде зафарбована
        // клітинка першого квадрата,
        // яка завадить розпізнати вершину
if(isTopLeft ‖
isBottomRight)
            {
// Ініціалізуємо
                // межі другого
                // квадрата:
s2_left =
                 s2_right = x;
                s2_top =
                 s2_bottom = y;
found = true;
            }
// Ідемо вздовж сторони
        // другого квадрата, що точно не перетинає-
        // ться (і не дотикається) до першого
        // квадрата, поки не дійдемо до незафар-
       // бованої клітинки. Паралельно розширює-
        // мо і другий вимір квадрата:
if(isTopLeft)
while(x < s1_left
        ? a[s2_left][s2_bottom + 1]
        : a[s2_right + 1][s2_top])
            {
              s2_right++;
              s2_bottom++;
            }
elseif(isBottomRight)
while(x > s1_right
        ? a[s2_right][s2_top – 1]
        : a[s2_left – 1][s2_bottom])
            {
              s2_left—;
              s2_top—;
            }
if(found) // Якщо роз ташування другого
          // квадрата визначено, виходимо з циклу
break;
          }
if(found) // Якщо розташування вже визначено,
          // виходимо із зовнішнього циклу
break;
      }
   }
elseif(internal) // Якщо нема зафар-
{ // бованих клітинок поза межами пер-
    // шого квадрата, але є такі клітини
    // строго всередині нього:
    // Визначаємо ліву, верхню, праву та
    // нижню межу зафарбованих клітинок,
    // що містяться строго всередині
    // першого квадрата:
s2_left = s1_right; // Ініціаліза-
s2_top = s1_bottom; // ція значень
s2_right = s1_left;
    s2_bottom = s1_top;
// Проходимо по всіх внутрішніх
    // клітинках квадрата:
for(intx = s1_left + 1;
        x <= s1_right - 1; x++)
for(inty = s1_top + 1;
        y <= s1_bottom – 1; y++)
if(a[x][y])
        { // Якщо клітинка чорна,
          // оновлюємо межі:
if(x < s2_left)
s2_left = x;
if(y < s2_top)
s2_top = y;
if(x > s2_right)
s2_right = x;
if(y > s2_bottom)
s2_bottom = y;
        }
// Включаємо верхню межу першого квадрата
      // до другого квадрата, якщо на наступній
     // горизонталі є хоча б одна чорна клітинка,
     // але немає двох сусідніх:
if(s2_top == s1_top + 1) // Якщо на наступній
{ // горизонталі є чорна клітинка:
s2_top = s1_top; // Відразу включаємо межу,
      // однак далі, можливо, виключимо:
for(intx = s1_left + 1;
          x < s1_right – 1; x++)
if(a[x][s1_top + 1] &&
           a[x + 1][s1_top + 1])
         { // Якщо знайшлися дві сусідні чорні
           // клітинки, виключаємо межу:
s2_top = s1_top + 1;
break;
        }
    }
// Аналогічно для нижньої межі:
if(s2_bottom == s1_bottom – 1)
    {
      s2_bottom = s1_bottom;
for(intx = s1_left + 1;
           x < s1_right – 1; x++)
if(a[x][s1_bottom – 1] &&
          a[x + 1][s1_bottom – 1])
         {
           s2_bottom =
            s1_bottom – 1;
```

```
break;
            }
        }
// Аналогічно для лівої межі:
if(s2_left == s1_left + 1)
    {
      s2_left = s1_left;
for(inty = s1_top + 1;
            y < s1_bottom – 1; y++)
if(a[s1_left + 1][y] &&
            a[s1_left + 1][y + 1])
          {
            s2_left = s1_left + 1;
break;
          }
    }
// Аналогічно для правої межі:
if(s2_right == s1_right – 1)
    {
      s2_right = s1_right;
for(inty = s1_top + 1;
            y < s1_bottom – 1; y++)
if(a[s1_right – 1][y] &&
            a[s1_right – 1][y + 1])
          {
            s2_right =
             s1_right – 1;
break;
          }
    }
  } else// Якщо крім клітинок першого
{ // квадрата немає ніяких зафарбованих
     // клітин: просто копіюємо перший
     // квадрат у другий:
s2_left = s1_left;
    s2_top = s1_top;
    s2_right = s1_right;
    s2_bottom = s1_bottom;
  }
into[2][4] = { // Рядки чисел, які потрібно
  // вивести в лексикографічному порядку
{s1_left, s1_top,
     s1_right, s1_bottom},
    {s2_left, s2_top,
     s2_right, s2_bottom}
  };
booldirectOrder = true; // Виводити
     // рядки у прямому порядку (true)
     // чи в оберненому (false)
  // Визначаємо, в якому ж порядку
  // потрібно вивести рядки:
for(inti = 0; i <4; i++)
if(o[0][i] > o[1][i]) // Рядки
    // треба вивести в оберненому порядку
{
directOrder = false;
break;
    } elseif(o[0][i] < o[1][i])
// Рядки треба вивести у прямому порядку
break;
for(introw = 0; row<2; row++)
// Номер рядка
for(inti = 0; i <4; i++)
// Номер числа в рядку
      // Виводимо число з відповідного рядка
    // залежно від потрібного порядку рядків:
printf(i <3 ? «%d « : «%d\n»,
          o[directOrder ? row
              : 1 – row][i]);
return0;
}
```

4. Схил

```
{ Free Pascal }
var x,y: array[1..100000] of integer;
  {Координати точок з цвяхами.}
  xc,yc:array[1..65536] of integer;
  {Запасні масиви, які використовуються
   при сортуванні методом злиття.}
  x1,y1,x2,y2:integer; {Поточне значення
   пари точок з найменшим схилом.}
  i,j,i2,j2:longint; {Індекси для
   циклів.}
  n:longint; {Кількість точок з цвяхами.}
{Впорядковування точок за зростанням
 y-координати, а при рівних y-координатах —
 за зростанням x-координати. Сортування
 виконується методом злиття. Координати
 точок, що сортуються, розташовані в
 масивах x та y.}
PROCEDURE Sort;
 var i:longint; {Номер елемента, після
     якого починається перший з двох
     відрізків, які зливаються.}
    j,k:longint; {Індекси елементів, що
     порівнюються у процесі злиття
     відрізків.}
    kb:longint; {Верхня межа для k.}
    m:longint; {Довжина відрізків, які
     зливаються.}
 Begin
  m:=1; {Початкова довжина відрізків, що
   зливається, встановлюється рівною 1.}
  {Цикл повторюватиметься, поки довжина
   відрізків, що зливаються, менша за
   довжини масивів.}
  while (m<n) do begin
   i:=0; {Встановлюємо, що перший
    відрізок розташований одразу після
    0-ої комірки. Тобто цей відрізок
    починається з елемента, який має
    індекс 1.}
   while (i+m<n) do begin
    {Копіюємо перший з відрізків, що
     зливаються, до масивів xc та yc.}
    for j:=1 to m do begin
     xc[j]:=x[i+j];
     yc[j]:=y[i+j];
    end;
    j:=1; {Встановлюємо значення
     біжучого індекса у першому
     відрізку.}
    k:=i+m+1; {Встановлюємо значення
     біжучого індекса у другому
     відрізку.}
    {Визначаємо верхню межу для k,
     рівну min(i+m*2,n).}
    kb:=i+m*2;
    if (kb>n) then kb:=n;
    {Повторюємо порівняння, поки не
```

```
      закінчиться один з відрізків, що
      зливаються. Після кожного
      порівняння менший елемент
      переносимо у основний масив, а
      відповідний біжучий індекс
      збільшуємо на 1.}
     while ((j<=m)and(k<=kb)) do begin
      if ((yc[j]<y[k])or((yc[j]=y[k])
        and(xc[j]<=x[k]))) then begin
       x[j+k-m-1]:=xc[j];
       y[j+k-m-1]:=yc[j];
       Inc(j);
      end;
      if (j<=m) then
       if ((yc[j]>y[k])or
         ((yc[j]=y[k])and
         (xc[j]>x[k]))) then begin
        x[j+k-m-1]:=x[k];
        y[j+k-m-1]:=y[k];
        Inc(k);
       end;
     end;
     {Якщо у першому відрізку залишилися
      елементи, переносимо їх до
      основного масиву. Якщо елементи
      залишилися у другому відрізку,
      нічого робити не потрібно. :) У
      згаданому випадку елементи другого
      відрізка вже знаходяться на
      потрібних місцях після закінчення
      циклу.}
     for j:=j to m do begin
      x[j+k-m-1]:=xc[j];
      y[j+k-m-1]:=yc[j];
     end;
     i:=i+m*2; {Переходимо до наступних
      двох відрізків.}
    end;
    m:=m*2; {Подвоюємо довжину відрізків,
     які зливаються.}
   end;
 End;
{Порівняння схилів, заданих парою точок
 (x1,y1), (x2,y2) та парою (x[i],y[i]),
 (x[j],y[j]). Якщо друга пара дає більш
 пологий схил, то точкам першої пари
 присвоюється значення точок другої пари.}
PROCEDURE Compare(var x1,y1,x2,y2:integer;
            i,j:longint);
 var p,q:longint; {Змінні, потрібні для
     того, щоб добутки різниць координат
     мали тип longint, а не integer. У
     протилежному випадку можлива
     помилка через перевищення
     максимально допустимого значення,
     бо вказаний добуток може сягати
     900000000.}
 Begin
  p:=abs(x2-x1);
  q:=abs(y2-y1);
  if (p*abs(y[i]-y[j])<q*abs(x[i]-x[j]))
  then begin
   x1:=x[i];
   y1:=y[i];
   x2:=x[j];
   y2:=y[j];
  end;
 End;
BEGIN
 {Відкриття файлів для вводу та виводу.}
 assign(input,'slope.in');
 assign(output,'slope.out');
 reset(input);
 rewrite(output);
 {Ввід вхідних даних.}
 read(n);
 for i:=1 to n do read(x[i],y[i]);
 Sort; {Сортування точок.}
 i:=1; j:=1;
 {Пошук найправішої точки, яка має таку ж
  у-координату, що й (x[i],y[i]).}
 while ((j<n)and(y[j+1]=y[i])) do Inc(j);
 if (j=n) then writeln(0); {Якщо всі точки
  лежать на одній горизонталі, на вивід
  подається число 0.}
 if (j<n) then begin
  {Присвоєння точкам (x1,y1) та (x2,y2)
   значень, які претендують на те, щоб
   задавати найпологіший схил.}
  x1:=x[i]; y1:=y[i];
  x2:=x[j+1]; y2:=y[j+1];
  while (j<n) do begin
   i2:=j+1; j2:=i2;
   {Пошук найправішої точки, яка має
    таку ж у-координату, що й
    (x[i2],y[i2]).}
   while ((j2<n)and(y[j2+1]=y[i2])) do Inc(j2);
   {Порівняння пари точок (x1,y1) та
    (x2,y2) з іншими претендентами.}
   Compare(x1,y1,x2,y2,i,i2);
   Compare(x1,y1,x2,y2,i,j2);
   Compare(x1,y1,x2,y2,j,i2);
   Compare(x1,y1,x2,y2,j,j2);
   i:=i2; j:=j2;
  end;
  write(x1,' ',y1,' ',x2,' ',y2); {Вивід
   найкращої пари точок.}
 end;
 {Закриття файлів вводу та виводу.}
 close(input); close(output);
END.
```

5. Каса

```
{ Free Pascal }
vari,m,n,p: longint;
o: text;    d,r: int64;
a,b: array[1..32665] of word;
functiongcd(a,b: word): word;
varx,y,z: word;    BEGIN
if (a=0) or (b=0) then gcd:=abs(a+b)
else     begin
x:=abs(a); y:=abs(b);
repeat z:=x mod y;  x:=y;  y:=z
until z=0;
gcd:=x              end END;
function c(k,m: integer):  int64;
varl,j: integer; d: int64;BEGIN
if (k<0) or (m<k) then c:=0 else
if (k=0) or (m=k) then c:=1 else  begin
```

```
if k<m–k then l:=k
else l:=m–k;
for j:=1 to l do    begin
a[j]:=m–j+1; b[j]:=j  end;
for j:=2 to l do    begin
i:=0; repeatinc(i);
d:=gcd(a[i],b[j]);
if d>1 then      begin
a[i]:=a[i] div d; b[j]:=b[j] div d end
until 1=b[j]      end;
d:=a[1];
if r=0
then for j:=2 to l do d:= d*a[j]
else for j:=2 to l do d:=(d*a[j]) mod r;
c:=d                    end END;
                    BEGIN
assign(o,'cash.in');
reset(o);
read (o,m,n,p,r);
close(o);
assign(o,'cash.out');
rewrite(o);
d:=c(m,m+n)–c(m–1–p,n+m);
while d<0 do d:=d+r;
writeln(o,d);
close(o)                END.
```

6. Гра

```
{ Free Pascal }
const n_=255; l_=255;
type ab = array[0..n_] of byte;
   aw = array[0..n_] of word;
var a,b: aw;
  {кінці послідовностей фішок без розривів}
c,{оцінки послідовностей фішок без розривів}
d:ab; {лишки від ділення на 2 кількостей
      послідовностей фішок без розривів}
yet: boolean;    {чи був запис у рядку?}
  o: text;
  g,       {найбільша кількість фішок,
           яку можна взяти за 1 хід}
sg: byte;          {оцінка позиції}
   l,    {кількість послідовостей фішок}
   m,   {найбільша довжина послідовності}
  i,j,k,n: word;
  s: set of byte; {множина оцінок позицій,
у які є хід з позиції — послідовності фішок}
                    BEGIN
assign(o,'game.in');
reset(o);
read (o,g);
k:=0;
l:=1;
read(o,a[1]);
b[1]:=a[1];
while not seekeof(o) do begin
inc(k);
read(o,j);
if a[l]+k=j then b[l]:=j
else begin
        k:=0;
inc(l);
        a[l]:=j;
        b[l]:=j end end;
close(o);
assign(o,'game.out');
rewrite(o);
m:=b[1]–a[1]+1;
for j:=2 to l do begin
 k:=b[j]–a[j]+1;
if m<k then m:=k end;
for j:=1 to m do d[j]:=0;
for j:=1 to l do begin
i:=b[j]–a[j]+1;
d[i]:=(d[i]+1) mod 2 end;
{Оцінки послідовностей фішок без пропусків}
for j:=0 to g do c[j]:=j;
for j:=g+1 to m do begin
s:=[];
for k:=1 to g do begin
include(s,c[j–k]);
for i:=1 to ((j–k) div 2) do
include(s,c[i] xor c[j–i–k]) end;
k:=0;  while k in s do inc(k);
c[j]:=k                end;
          {Оцінювання позиції}
sg:=0;
for j:=1 to m do
if d[j]=1 then sg:=sg xor c[j];
            {Запис відповіді}
if sg=0 then for j:=1 to g do writeln(o)
else
for j:=1 to g do begin
yet:=false;
for k:=1 to l do if b[k]–a[k]+1>=j then
for n:=a[k] to b[k]–j+1 do begin
if (sg xor c[b[k]–a[k]+1]
xor c[n–a[k]]
xor c[b[k]–(n+j–1)])=0 then begin
if yet then write(o,' ');
write(o,n);
yet:=true end; end;
writeln(o)              end;
close(o)              END.
```

★ ★ ★

Підписано до друку 15.09.2015 р. Формат 60x84 1/8. Папір офсет. Друк офсет. Умовн. друк. арк. 5,88.
Умовн. фарбо-відб. 11,76. Обл.-вид. арк. 8,54. Видавець: ФО-П Вероцький С.В. Зам. №15–127.
Віддруковано у друкарні видавництва «Фенікс». Свід. ДК 271 від 7.12.2000 р.
E-mail: csf22101@ukr.net, www.csf221.wordpress.com, www.facebook.com/csfmagazine.

Передплату на наш журнал можна оформити у будь-якому відділенні зв'язку. Наш індекс 74248