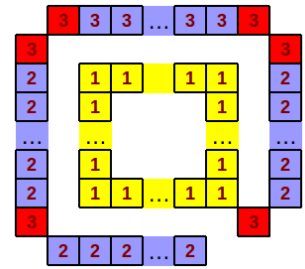


Розбір задачі А: Піксельний равлик

Піксельний равлик – це така фігура, яку дуже просто намалювати на аркуші в клітинку, і яка дуже схожа на лігатуру «at» («@»).

Знайдімо закономірність, тобто залежність кількості клітинок у піксельному равлику k -го порядку від параметра k . Клітинки, які знаходяться по периметру квадрата, позначимо цифрою «1» – на малюнку праворуч зафарбовані жовтим кольором. Клітинки, які знаходяться навпроти сторін квадрата, позначимо цифрою «2» – на малюнку праворуч зафарбовані синім кольором. Решту клітинок позначимо цифрою «3» – на малюнку праворуч зафарбовані червоним кольором.



Рахуємо сумарну кількість клітинок у піксельному равлику k -го порядку від параметру k :

- По периметру квадрата (жовті клітинки, позначені цифрою «1»): $s_1 = 4 \cdot k - 4$, наприклад, якщо $k = 2$, то $s_1 = 4 \cdot 2 - 4 = 4$;
якщо $k = 3$, то $s_1 = 4 \cdot 3 - 4 = 8$;
якщо $k = 4$, то $s_1 = 4 \cdot 4 - 4 = 12$.
Треба зазначити, що випадок, коли $k = 1$, треба розглянути окремо: $s_1 = 1$.
- Навпроти сторін квадрата (сині клітинки, позначені цифрою «2»): $s_2 = 4 \cdot k$.
- Решта клітинок (червоні клітинки, позначені цифрою «3»): $s_3 = 6$.
- Відповідь:
якщо $k = 1$, то $s = s_1 + s_2 + s_3 = 1 + 4 \cdot 1 + 6 = 11$,
якщо $k > 1$, то $s = s_1 + s_2 + s_3 = 4 \cdot k - 4 + 4 \cdot k + 6 = 8 \cdot k + 2$.

Інший підхід до розв'язку цієї задачі може бути таким. Можна помітити, що залежність кількості клітинок у піксельному равлику k -го порядку – це лінійна функція від k , тобто

$$s(k) = \alpha \cdot k + \beta$$

Візьмемо значення цієї функції при $k = 2$ та при $k = 3$ з умови задачі:

$$s(2) = \alpha \cdot 2 + \beta = 26$$

$$s(3) = \alpha \cdot k + \beta = 34$$

та розв'яжемо систему лінійних рівнянь: $\alpha = 8$, $\beta = 2$.

Приклад функції, яка розв'язує цю задачу, мовою програмування Pascal:

```
1 function solve(k : LongInt) : LongInt;  
2 begin  
3   if k = 1 then solve := 11 else solve := 8 * k + 2;  
4 end;
```

Приклад функції, яка розв'язує цю задачу, мовою програмування C++:

```
1 int solve(int k) {  
2   if (k == 1)  
3     return 11;  
4   return 8 * k + 2;  
5 }
```

Приклад функції, яка розв'язує цю задачу, мовою програмування Python3:

```
1 def solve(k):  
2   if k == 1:  
3     return 11  
4   return 8 * k + 2
```

Розбір задачі В: Нова Пошта

Є п'ять великогабаритних вантажів m_1, m_2, m_3, m_4, m_5 та дві автівки з вантажністю M_1 та M_2 . Треба розподілити вантажі по автівках.

Обмеження в цій задачі доволі маленькі, тому можна просто перебрати усі варіанти:

- $m_1 + m_2 + m_3 + m_4 + m_5 \leq M_1$ – всі вантажі можна доставити першою автівкою;
- $m_1 + m_2 + m_3 + m_4 + m_5 \leq M_2$ – всі вантажі можна доставити другою автівкою;
- $m_1 + m_2 + m_3 + m_4 + m_5 \geq M_1 + M_2$ – вантажі неможливо доставити;
- У першу автівку завантажимо один вантаж, а решту, тобто чотири вантажі – у другу автівку. Є п'ять способів такого розподілу вантажів.
- У другу автівку завантажимо один вантаж, а решту, тобто чотири вантажі – у першу автівку. Є п'ять способів такого розподілу вантажів.
- У першу автівку завантажимо два вантажі, а решту, тобто три вантажі – у другу автівку. Є десять способів такого розподілу вантажів.
- У другу автівку завантажимо два вантажі, а решту, тобто три вантажі – у першу автівку. Є десять способів такого розподілу вантажів.

Організувати перебір можна, наприклад, так. Нехай $p[1..5]$ – масив з п'яти елементів, кожен з яких може бути одиничкою або нулем. Якщо $p[i] = 1$, то вантаж масою m_i намагаємось завантажити у першу автівку, інакше – у другу. Згенеруємо усі можливі масиви p – кожному такому масиву відповідає двійковий запис числа від 0 до 31, наприклад:

$$10 = 01010_2 : p[1] = 0, p[2] = 1, p[3] = 0, p[4] = 1, p[5] = 0.$$

$$25 = 11001_2 : p[1] = 1, p[2] = 1, p[3] = 0, p[4] = 0, p[5] = 1.$$

Випадки, коли вантажі можна доставити однією автівкою або не можливо доставити, варто розглянути окремо. Перебір варіантів завантаження двох автівок можна реалізувати так:

```
1 for (int i = 1; i <= 30; i++) {
2     string Vasyl = "Vasyl: ", Petro = "Petro: ";
3     int t = i;
4     for (int j = 0; j < 5; j++) {
5         p[j] = t % 2;
6         if (p[j] == 1)
7             Vasyl = Vasyl + to_string(m[j]) + " ";
8         else
9             Petro = Petro + to_string(m[j]) + " ";
10        t /= 2;
11    }
12    cout << Vasyl << endl << Petro << endl << "-----\n";
13 }
```

Розбір задачі С: Тренування пам'яті

Є числовий масив $A[1..n]$ із n елементів. Відбувається n кроків алгоритму:

- На кожному кроці видаляється один з елементів масиву $A[1..n]$.
- Елементи, які видаляються на непарних кроках (першому, третьому, п'ятому, і так далі) забирає собі Василь.
- Елементи, які видаляються на парних кроках (другому, четвертому, шостому, і так далі) забирає собі Петро.

Треба з'ясувати, які елементи масиву забере Василь, а які – Петро.

Фактично, на кожному кроці алгоритму є два масиви, назвемо їх v та w , про які відомо, що

- $v.size() - w.size() = 1$, тобто в масиві v на один елемент більше ніж в масиві w .
- Якщо з масиву v видалити один елемент X , та, можливо, переставити певним чином елементи, то отримаємо масив w .

Треба знайти X .

Існує багато способів реалізувати один крок такого алгоритму. Наприклад, можна для кожного елемента з масиву w знайти такий самий елемент в масиві v . Але все набагато простіше:

$$X = \sum_{i=1}^{v.size()} v[i] - \sum_{i=1}^{w.size()} w[i]$$

Приклад функції, яка розв'язує цю задачу, мовою програмування C++:

```
1 typedef long long LL;
2 vector<LL> v;
3 vector<LL> ans[2]; // ans[0] - Vasyl's numbers, ans[1] - Peter's numbers;
4
5 void solve() {
6     int n, k = 0;
7     LL s = 0;
8     cin >> n;
9     v.resize(n);
10    for (int i = 0; i < n; i++) {
11        cin >> v[i];
12        s += v[i];
13    }
14    for (int step = n - 1; step > 0; step--) {
15        v.resize(step);
16        LL sum = 0;
17        for (int i = 0; i < step; i++) {
18            cin >> v[i];
19            sum += v[i];
20        }
21        ans[k].push_back(s - sum);
22        s = sum;
23        k = 1 - k;
24    }
25    ans[k].push_back(v[0]);
26    sort(ans[0].begin(), ans[0].end());
27    sort(ans[1].begin(), ans[1].end());
28 }
```

Розбір задачі D: Ділянки

Є квадратна ділянка, яку геодезисти розділили на n^2 прямокутних ділянок, провівши $(n - 1)$ вертикальних ліній та $(n - 1)$ горизонтальних ліній. Нам відомі площі ділянок на «головній діагоналі» та на «побічній діагоналі».

Ідея перша

Нехай є прямокутник, який складається з чотирьох ділянок – два стовпчики та два рядки. Відомі площі трьох ділянок з чотирьох. Навчимося знаходити невідому площу S_4 (див. Рис. 1).

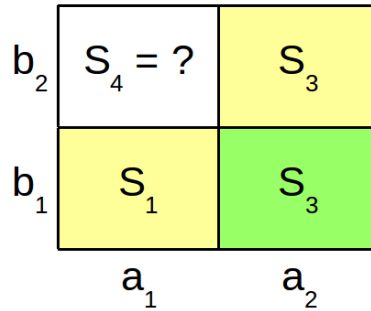


Рис. 1: Як обчислити S_4 ?

Проведемо нескладні обчислення:

$$\begin{aligned} S_1 &= a_1 \cdot b_1, \\ S_2 &= a_2 \cdot b_2, \\ S_3 &= a_2 \cdot b_1, \\ S_4 &= a_1 \cdot b_2 = \frac{a_1 \cdot b_1 \cdot a_2 \cdot b_2}{a_2 \cdot b_1} = \frac{S_1 \cdot S_2}{S_3} \end{aligned}$$

Таким чином, щоб знайти площу невідомої ділянки, достатньо перемножити площі на головній діагоналі, та поділити на площу ділянки під головною діагоналлю. Можна вивести аналогічну формулу для випадку, коли відомими є площі інших трьох ділянок з чотирьох. У принципі, ця ідея дає нам можливість обчислити площі усіх ділянок.

Ідея друга

Давайте обчислимо площу ділянки, що знаходиться у лівому верхньому куті (див. Рис. 2).

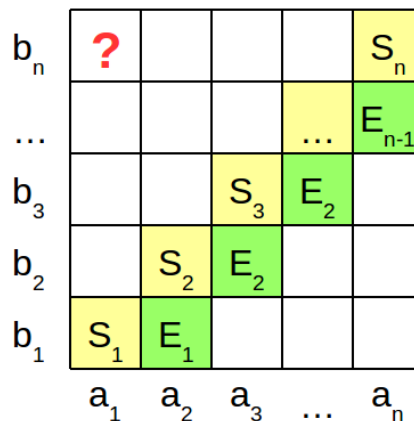


Рис. 2: Як обчислити площу ділянки у лівому верхньому куті?

Проведемо нескладні обчислення:

$$? = a_1 \cdot b_n = \frac{(a_1 \cdot a_2 \cdot \dots \cdot a_n) \cdot (b_1 \cdot b_2 \cdot \dots \cdot b_n)}{(a_2 \cdot a_3 \cdot \dots \cdot a_n) \cdot (b_1 \cdot b_2 \cdot \dots \cdot b_{n-1})} = \frac{S_1 \cdot S_2 \cdot \dots \cdot S_n}{E_1 \cdot E_2 \cdot \dots \cdot E_{n-1}}.$$

Ідея третя

Нехай $p > q$. По аналогії з другою ідеєю, площу ділянки, що знаходиться на перетині p -го стовпчика та q -го рядку, можна обчислити так:

$$\frac{S_q \cdot S_{q+1} \cdot \dots \cdot S_p}{E_q \cdot E_{q+1} \cdot \dots \cdot E_{p-1}}.$$

Нехай $p < q$. Користуючись першою ідеєю, можна обчислити площі ділянок на іншій «побічній діагоналі», ця діагональ складається з ділянок, що знаходяться на перетині i -го стовпчика та $(i + 1)$ -го рядка. Далі поміняємо місцями побічні діагоналі та поміняємо місцями значення p та q . Тепер відповідь можна обчислити за формулою у попередньому абзаці.

Ідея четверта

Відповідь треба фактаризувати, тобто представити у вигляді:

$$Ans = p_1^{s_1} \cdot p_2^{s_2} \cdot \dots \cdot p_k^{s_k},$$

де p_i – прості числа, а s_i – цілі числа ($s_i \neq 0$). Можна одразу факторизувати вхідні дані, причому факторизацію числа зберігати в асоціативному масиві (map в C++). Також доцільно реалізувати операції множення та ділення факторизованих чисел

Факторизація, множення та ділення факторизацій:

```
1 typedef long long LL;
2
3 map<LL, LL> Factorisation(LL x) {
4     map<LL, LL> m;
5     m.clear();
6     LL d = 2;
7     do {
8         while (x % d == 0) {
9             m[d] += 1;
10            x /= d;
11        }
12        d++;
13        if (d * 1LL * d > x && x != 1) {
14            m[x] += 1;
15            break;
16        }
17    } while (x != 1);
18    return m;
19 }
20
21 map<LL, LL> mult(map<LL, LL> a, map<LL, LL> b) {
22     for (auto x : b)
23         a[x.first] += x.second;
24     return a;
25 }
26
27 map<LL, LL> div(map<LL, LL> a, map<LL, LL> b) {
28     for (auto x : b)
29         a[x.first] -= x.second;
30     return a;
31 }
```

Використовуючи ці чотири прості ідеї, нескладно отримати повне рішення цієї задачі.

Розбір задачі Е: Квадрат чи прямокутник

Є квадратна дошка розміром 100×100 клітинок. Рядки дошки пронумеровані цілими числами від 1 до 100 зверху донизу, а стовпчики – цілими числами від 1 до 100 зліва направо. Відповідно, клітинка, яка знаходиться у лівому верхньому куті дошки, має координати $(1, 1)$; клітинка, яка знаходиться у правому нижньому куті, має координати $(100, 100)$; а клітинка, яка знаходиться на перетині i -го рядка та j -го стовпчика, має координати (i, j) . На дошці зафарбовано прямокутну ділянку, треба з'ясувати, чи є зафарбована ділянка квадратом.

В умові зазначено, що площа зафарбованої ділянки займає не менш ніж 4% від площі всієї дошки – це не менш ніж 400 клітинок, тобто, якщо зафарбована ділянка – квадрат, то його сторона не менша за 20.

Спочатку зробимо 16 запитів про клітинки з координатами $(20 \cdot i, 20 \cdot j)$, $1 \leq i, j \leq 4$. Тепер є два випадки:

1. На всі 16 запитів Еолімп відповів «outside». Тоді, якщо зафарбована ділянка – квадрат, то його сторона може дорівнювати тільки 20, причому він буде торкатися або нижньої сторони поля, або правої. Зробимо ще 9 запитів про клітинки з координатами $(20 \cdot i, 20 \cdot j)$, $1 \leq i, j \leq 5$, $i = 5 \text{ or } j = 5$. Виникає ще два випадки:
 - (а) На всі 9 запитів Еолімп відповів «outside». Тоді, очевидно, що зафарбована ділянка – прямокутник.
 - (б) Хоча б на один з 9 запитів Еолімп відповів «inside». Нехай цей запит був про клітинку $(20 \cdot i_0, 20 \cdot j_0)$. Тоді бінарним пошуком за $\log(20) = 5$ запитів можна знайти таке мінімальне j , що клітинка $(20 \cdot i_0, j)$ зафарбована. Тепер залишилось перевірити, що клітинка $(20 \cdot i_0, j + 19)$ також зафарбована, а клітинка $(20 \cdot i_0, j + 20)$ не зафарбована. Якщо це так, то відповідь – квадрат, інакше – прямокутник.

Усього ми зробили $16 + 9 + 5 + 2 = 32$ запити.

2. Хоча б на один з 16 запитів Еолімп відповів «inside». Тоді двома бінарними пошуками за $2 \cdot \log(20) = 10$ запитів можна знайти мінімальні i та j такі, що клітинка (i, j) належить зафарбованій ділянці. Далі, ще одним бінарним пошуком за $\log(20) = 5$ запитів можна знайти таке максимальне d , що клітинка $(i + d, j + d)$ також зафарбована. Залишилось зробити усього два запити: $(i + d + 1, j + d)$ та $(i + d, j + d + 1)$. Якщо обидві ці клітинки не зафарбовані, то відповідь – квадрат, інакше – прямокутник.

Усього ми зробили $16 + 10 + 5 + 2 = 33$ запити.

Автор першого туру – Микола Арзубов.