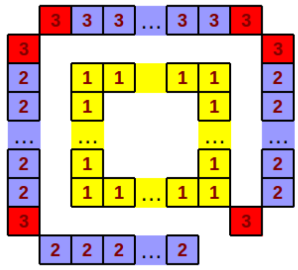
Розбір задачі A: Піксельний равлик



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 параметра 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'язую цю задачу, мовою програмування Pascal:

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

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

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

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

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

Розбір задачі B: 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:

- $m_1 + m_2 + m_3 + m_4 + m_5 \leqslant M_1$ — всі вантажі можна доставити першою автівкою;
- $m_1 + m_2 + m_3 + m_4 + m_5 \leqslant M_2$ — всі вантажі можна доставити другою автівкою;
- $m_1 + m_2 + m_3 + m_4 + m_5 \geqslant M_1 + M_2$ — 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:

```
for (int i = 1; i <= 30; i++) {
  string Vasyl = "Vasyl: ", Petro = "Petro: ";
  int t = i;
  for (int j = 0; j < 5; j++) {
    p[j] = t % 2;
    if (p[j] == 1)
       Vasyl = Vasyl + to_string(m[j]) + " ";
    else
       Petro = Petro + to_string(m[j]) + " ";
    t /= 2;
  }
  cout << Vasyl << endl << Petro << endl << "-----------\n";
}
```

Розбір задачі C: 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 елементу 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'язую цю задачу, мовою програмування C++:

```
typedef long long LL;
vector<LL> v;
vector<LL> ans[2]; // ans[0] - Vasyl's numbers, ans[1] - Peter's numbers;

void solve() {
  int n, k = 0;
  LL s = 0;
  cin >> n;
  v.resize(n);
  for (int i = 0; i < n; i++) {
    cin >> v[i];
    s += v[i];
  }
  for (int step = n - 1; step > 0; step--) {
    v.resize(step);
    LL sum = 0;
    for (int i = 0; i < step; i++) {
      cin >> v[i];
      sum += v[i];
    }
    ans[k].push_back(s - sum);
    s = sum;
    k = 1 - k;
  }
  ans[k].push_back(v[0]);
  sort(ans[0].begin(), ans[0].end());
  sort(ans[1].begin(), ans[1].end());
}
```

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).

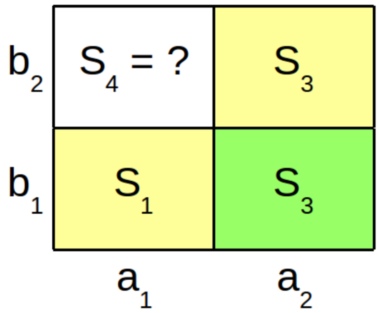


Рис. 1: Як обчислити S_4?

Проведемо нескладні обчислення:

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

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).

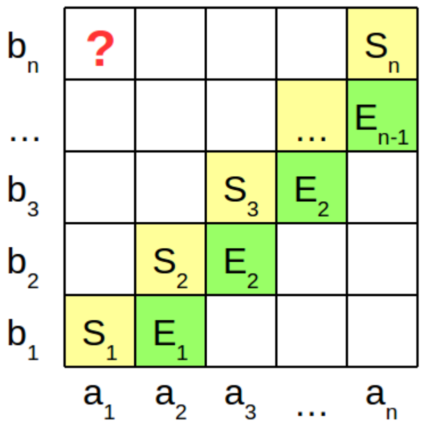


Рис. 2: Як обчислити площу ділянки у лівому верхньому куті?

Проведемо нескладні обчислення:

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

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:

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

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:

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

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:

```
typedef long long LL;

map<LL, LL> Factorisation(LL x) {
  map<LL, LL> m;
  m.clear();
  LL d = 2;
  do {
    while (x % d == 0) {
      m[d] += 1;
      x /= d;
    }
    d++;
    if (d * 1LL * d > x && x != 1) {
      m[x] += 1;
      break;
    }
  } while (x != 1);
  return m;
}

map<LL, LL> mult(map<LL, LL> a, map<LL, LL> b) {
  for (auto x : b)
    a[x.first] += x.second;
  return a;
}

map<LL, LL> div(map<LL, LL> a, map<LL, LL> b) {
  for (auto x : b)
    a[x.first] -= x.second;
  return a;
}
```

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

Розбір задачі E: 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 і-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 $(20 \cdot i, 20 \cdot j), 1 \leqslant i, j \leqslant 4$. 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 $(20 \cdot i, 20 \cdot j), 1 \leqslant i, j \leqslant 5, i = 5 or j = 5$. 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.

Розбір задачі F: Галерея

Розглянемо сумарну кількість конфігурацій ваз, не враховуючи висоти (пронумеруймо їх 1, 2, 3), конфігурації будуть наступні: {1}, {2}, {3}, {1,2}, {1,3}, {2,3}. Тобто існує всього 5 конфігурацій. Конфігурації {1}, {2} та {3} нас не цікавлять, оскільки ці конфігурації не можуть відповідати вежам з максимальною висотою. Розглядатимемо лише конфігурації {1,2}, {1,3} та {2,3}. Помітимо, що кожна з цих конфігурацій – це конфігурація {1,2,3} без однієї з трьох ваз, відповідна висота вежі яка відповідає даним конфігураціям – висота вежі яка відповідає {1,2,3} мінус висота відсутньої вази. Така висота буде максимальною, якщо висота відсутньої вази мінімальна, тому відповідь – це a+b+c мінус висота найнижчої вази.

Розбір задачі G: Невільний песик

Помітимо, що бджоли не можуть дістатися песика тоді і тільки тоді, коли існує купол, по одну сторону якого бджоли, а по іншу — песик. Тому розв'язок полягає у тому, щоб для кожного купола перевірити, чи розділяє він песика і бджіл, якщо такий купол можливо знайти —, то виведемо `NO`, і купол з мінімальним індексом, що розділяє їх, інакше `YES`. Перевіряти чи лежить точка в куполі можна наступним чином: оскільки усі точки не нижче 0, то достатньо перевірити, чи відстань між центром купола і точкою не більша за радіус купола, для перевірки без використання дробових чисел — можна порівнювати квадрат радіуса і квадрат відстані. Відстань між двома точками (x_1, y_1), (x_2, y_2) можна полічити за формулою $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Розбір задачі H: Дороги Потоколяндії

Розглянемо шлях з міста 1 в місто n, є два випадки:

1. n – непарне, тоді n не зв'язане ребром з жодним містом крім себе, тобто дістатись 1 з нього не можливо бо неможливо зробити перехід по дорозі в місто n, або $n = 1$

2. n – парне, тоді з міст з непарними індексами існуватимуть дороги лише в міста з парними індексами, при чому для кожного непарного міста існуватиме лише одна дорога дотична до нього. Тому робитимемо наступну процедуру: для кожного непарного міста утотожнимо його з його парним сусідом по ребру (він завжди існуватиме), і розв'язуватимемо задачу лише для парних, пам'ятаючи з ким утотожнене місто 1. Оскільки ми розв'язуємо лише для парних - можна поділити всі індекси на 2 без втрати загальності. Тоді 1 перейде в 1 в наступному кроці.

   Виконуватимемо таке перетворення поки n не стане непарним, тоді можна з пункту 1 зрозуміти, що не існуватиме шляху з міста 1 в місто n

Розбір задачі I: Прямокутники всюди

1. Відповідь «`NO`» — тоді і тільки тоді, коли ($x_1 = 0$ і $x_2 = n$) або ($y_1 = 0$ і $y_2 = m$).

2. Відповідь «`NO`» — тоді, коли або один з прямокутників виконує умову з блока 1, або один з прямокутників торкається лівої і нижньої пари стінок, інший — верхньої і правої і ці два прямокутники перетинаються

3. Потрібно перевіряти для кожної пари умову блока 2, також якщо прямокутники не перетинаються, але обидва перетинаються з третім — то відповідь теж — «`NO`»

4. Можна побудувати граф між всіма точками та лініями перетину та запустити BFS з пари сторін (ліво, низ) і перевірити, чи він дійшов до пари сторін (право, верх), або стиснути координати і розв'язувати задачу на матриці $(2k, 2k)$ буквально для кожної клітинки перевіряючи чи можна в неї зайти.

5. Можна як і в блоці 4 побудувати матрицю і завдяки двовимірним префікс-сумам дізнатись, які клітинки доступні. Далі все як в блоці 4.

6. Дістатись з точки $(0, 0)$ в точку (n, m) не можливо тоді і тільки тоді, коли між верхньою і правою парою стінок кімнати, та лівою і нижньою парою можливо пройти лише по прямокутниках. Будемо перевіряти це наступним чином: додамо як стартові точки BFS в чергу усі прямокутники які дотикаються однієї пари стінок і за $O(n^2)$ перевіримо чи можливо дійти до іншої пари стінок, фактично ми перевірятимемо чи існує послідовність прямокутників така, що шлях між парами стінок проходить через ці прямокутники і лише через них, якщо два прямокутники мають спільну точку, то з будь-якої точки одного дійти до будь-якої точки іншого. Прошу помітити, що тримати матрицю суміжностей між прямокутниками не обов'язково, можна для кожного прямокутника коли в BFS настає його черга - перевіряти усі інші прямокутники на те, чи мають вони спільну точку.

Тут можна почитати про алгоритм bfs: https://algoua.com/algorithms/graphs/bfs/
Тут можна почитати про двовимірні префікс суми: https://usaco.guide/silver/more-prefix-sums?lang=cpp

Розбір задачі J: Добуток

1. Будемо перебирати перший мінімум, другий мінімум і розширяти відрізок, сумарно це все працюватиме не довше $O(n^3)$.

2. Будемо перебирати початок відрізка, далі будемо розширювати його вправо підтримуючи в std::multiset множину елементів, для кожної правої границі візьмемо з мультисета два мінімуми та порахуємо добуток. Таким чином ми переберемо кожен відрізок і порахуємо відповідь за $O(n^2 \cdot log(n))$.

3. Будемо робити все як в блоці 2, але мінімуми підтримуватимемо парою чисел.

4. Цей блок був створений для різноманітних евристик (переборів з відсіканням, чи чогось подібного).

5. Будемо перебирати другий мінімум. Для кожного числа порахуємо перше менше нього справа і зліва від нього - це можна зробити стеком. Потім для кожного другого мінімуму ми можемо взяти як перший мінімум або перше менше зліва, або справа, обидва взяти не можемо, візьмемо одне, а потім друге, опишемо процес з першим меншим зліва, справа буде симетрично. Нехай зараз ми розглядаємо i - як другий мінімум. l - індекс першого меншого зліва, r - справа. На цей час ми визначились, що точно візьмемо i та l, а отже відрізок $a[l, i]$. Тепер ми хочемо його максимально розширити так, щоб значення двох мінімумів не змінилось. Розширити вправо можна лише до r виключно, оскільки з визначення $a_i > a_r$. Розширити вліво можна до першого меншого за a_i виключно, серед тих, які лежать лівіше від позиції i, це ми можемо робити тримаючи останні дві позиції для кожного значення яке можуть приймати елементи масиву і будемо завдяки цьому шукати лівий кінець відрізка за $O(\sqrt{n})$ і тоді загальна складність алгоритму буде $O(n \cdot \sqrt{n})$.

6. Робитимемо все як в блоці 5, але пошук лівої границі можемо робити бін пошуком і sparse table, або деревом відрізків з каскадним спуском. Це працюватиме за $O(n \cdot log(n))$.

Тут можна почитати як шукати найближчий менший елемент: https://bit.ly/3XIVqIj
Тут можна почитати про sparse table: bit.ly/3RcNFrm0
Тут можна почитати про дерево відрізків з каскадним спуском: bit.ly/3kKQ28J0
Автор другого туру — Владислав Денисюк.