# 1 Розв'язання задачі «Lesson»

Дану задачу можна розв'язати динамічним програмуванням.

Для початку, трохи «стиснемо» слово, замінивши послідовності однакових літер, що йдуть підряд, на одинарне входження цієї літери (наприклад, `aaabbcaad` → `abcad`). Зрозуміло, що відповідь від цього не зміниться, час роботи в найгіршому випадку — теж, але принаймні не треба буде виконувати гарантовано непотрібну роботу.

В якості серії підзадач аналогічної структури різних розмірів варто взяти таку: *«За яку мінімально можливу кількість кроків можна викреслити усі літери з підрядка* `a[i]..a[j]`*?»* (маються на увазі са́ме підрядки, інакше кажучи неперервні підпослідовності).

Потрібна задача буде однією із задач такої серії, а саме задачею розміру $(1, m)$, де $m$ — довжина вхідного рядка (після «стиснення»).

Тривіальні підзадачі, розв'язки котрих очевидні — це підзадачі розмірами $(i, i)$ (односимвольні підрядки викреслюються за один хід) та розмірами $(i, i{+}1)$ (двосимвольні підрядки, що складаються з різних символів, викреслюються за два ходи).

Позначимо шукану мінімальну кількість дій для викреслення підрядка `a[i]..a[j]` як $T_{ij}$. За $K_{ij}$, $K_{ij}^{(*)}$ та $K_{ij}^{(**)}$ позначимо певні допоміжні величини (котрі потрібні ).

Отже, оптимальні розв'язки для нетривіальних підзадач розміром $(i, j)$ достатньо шукати серед таких:

Спочатку викреслити підрядок `a[i]..a[k]`, а потім підрядок `a[k+1]..a[j]` (перебираємо усі $k$: $i \leqslant k < j$). Це дає формулу

$$K_{ij} = \min_{i \leqslant k < j} \{T_{ik} + T_{k+1,j}\}. \tag{1}$$

Якщо `a[i]` $\neq$ `a[j]`, то це і буде відповіддю підзадачі розміром $(i, j)$.

Але за умови `a[i]` $=$ `a[j]` (оскільки ми «стиснули» вхідне слово, це можливо лише при $j > i + 1$), з'являються додаткові можливості пришвидшити викреслювання:

1. Спочатку викреслити підрядок `a[i+1]..a[j-1]` (увесь крім крайніх літер), а потім (за один хід, бо вони однакові) крайні літери. Це дає формулу

$$K^* = T_{i+1,j-1} + 1. \tag{2}$$

2. Якщо виконується і умова `a[i]` $=$ `a[j]`, і крім того всере́дині підрядка `a[i]..a[j]` є ще й інші входження такої самої літери, як крайні ($\exists k : (i < k < j) \wedge$ (`a[i]` $=$ `a[k]` $=$ `a[j]`)), то додатково слід розглянути ще можливості викреслити спочатку підрядок `a[i+1]..a[k-1]`, потім підрядок `a[k+1]..a[j-1]`, а потім (за один хід, бо вони однакові) літери `a[i]`, `a[k]` та `a[j]`.

   Здавалося б, це повинно призвести до формули вигляду

$$K_{ij}^{(**)} = \min_{k:a[i]=a[k]=a[j]} \{T_{i+1,k-1} + T_{k+1,j-1} + 1\}. \tag{3}$$

Але більш тонкий аналіз показує, що це все-таки не зовсім правильно.

Формула (3) враховує ситуацію типу `a...a...a`, але не враховує ситуації типу `a...a...a...a`, `a...a...a...a...a` і так далі.

Може виникнути бажання у випадку `a[i]` $=$ `a[j]` познаходити усі внутрішні входження цієї літери, і будувати стратегію так: спочатку повидаляти підслова що їх розділяють, а потім знищувати усі ці літери за один хід. Але така стратегія насправді теж неправильна.[1]

Тому формулу (3) треба модифікувати так, щоб з одного боку допускалася можливість видаляти за один хід більш ніж три входження літери, а з іншого не вимагалося знищувати зразу всі входження. Такою модифікацією виявляється

$$K_{ij}^{(**)} = \min_{k:a[i]=a[k]=a[j]} \{T_{ik} + T_{kj} - 1\}. \tag{4}$$

[1] Скажімо, у слові `abacadaea` вона спрацьовує (спочатку повидаляти окремі літери `b`, `c`, `d`, `e`, а потім, за один хід, усі літери `a`). А у слові `abcadaeafacba` оптимальна послідовність ходів така: спочатку видалити (єдину) літеру `d`, потім `e`, потім textttf, потім послідовність `aaaa` (котра на той момент опиниться в середи́ні слова), потім `cc`, `bb`, і, нарешті, крайні входження `aa`. Бачимо що тут треба видаляти за раз 4 входження `a`, але не всі 6 входжень.

Останнє "$-1$" означає поправку на те, що сума $T_{ik}+T_{kj}$ рахує вилучення літери `a[k]` двічі, тоді як її треба вилучити один раз.

Сама́ ж оптимальна відповідь рахується як

$$T_{ij}=\min(K_{ij},K^{*}_{ij},K^{**}_{ij}) \quad \begin{array}{l}\text{(рахуючи по тим величинам, котрі ма-}\\ \text{ють смисл для даного підслова)}\end{array} \tag{5}$$

Звичайно, реалізовувати формули (1)–(2), (4)–(5) через рекурсивні функції украй недоцільно. Натомість треба заповнювати табличку для $T_{ij}$, починаючи від головної діагоналі і рухаючись послідовно по паралельним головній діагоналі лініям, доки не дойдемо до кутового елемента $T_{1m}$.

Дана задача майже повністю відповідає задачі «Email» Всеукраїнської олімпіади 2000 р. Тож рекомендуємо звернутися також до її пояснень.

## 2 Розв'язання задачі «Net»

Цілком зрозуміло, що потрібно знайти час передавання повідомлення від кожного комп'ютера до кожного. Звичайно табличку цих часів оформлюють як двовимірний масив, номери рядки котрого відповідають номерам комп'ютера–джерела, стовпчики — комп'ютера–приймача.

Потім потрібно знайти в кожному рядку найбільше число (через який проміжок часу отримає повідомлення останній комп'ютер, якщо відправити повідомлення з даного), і вибрати серед усих знайдених чисел найменше.

Потрібно врахувати, що з деяких комп'ютерів на деякі повідомлення можуть взагалі не передаватися. Тому потрібно або зберігати окремо (булевий) масив можливостей такої передачі, і окремо (числовий) масив часу передачі, або ж ввести так звану «машинну $\infty$» — число, гарантовано більше за будь-яке «розумне» значення.

Отже, основна проблема — побудова матриці часів передавання повідомлень.

На нашу думку, найдоцільнішими є два способи.

Перший — багатократне (починаючи по черзі від кожного комп'ютера) застосування пошуку в ширину. Тобто, спочатку потрібно відмітити і запам'ятати всі комп'ютери, куди повідомлення може дійти від початкового за 1 хід. Потім шукаємо комп'ютери, куди повідомлення доходить з початкового за 2 ходи. Щоб повідомлення доходило до комп'ютера за 2 ходи, потрібно, щоб цей комп'ютер міг отримати його з якогось із тих комп'ютерів, до котрих воно доходить за 1 хід. Потім розглядаємо «сусідів» тих комп'ютерів, котрі отримують повідомлення за 2 ходи, і знаходимо перелік тих, котрі отримують повідомлення за 3 ходи, і так далі.

Особливо наголошуємо, що при застосуванні пошуку в ширину *спочатку* знаходяться *всі* комп'ютери, котрі можуть отримати повідомлення за 1 хід, *потім* усі, що отримують повідомлення за 2 ходи, і так далі. Тому, отримане значення кількості ходів *відразу є остато́чним*, тобто не може бути ситуації, коли спочатку знайшли якийсь один маршрут передавання повідомлень від початкового комп'ютера, а потім — оптимальніший (коротший).

Алгоритм пошуку в ширину широко відомий і викладений майже в усіх книгах, де згадується алгоритмічний підхід до теорії графів. Найкращим викладенням ми вважаємо [Лип] (гл. 2.1–2.3).

Іншим доцільним засобом розв'язання даної задачі є алгоритм Флойда–Уоршалла:

```
for k:=1 to n do begin
  for i:=1 to n do
    for j:=1 to n do begin
      if D[i,k]+D[k,j]<D[i,j] then
        D[i,j]:=D[i,k]+D[k,j];
    end;
end
```

Тут `D` — масив розміром $N\times N$. Спочатку він задає вказані в умові зв'язки́: по головній діагоналі (елементи `D[1,1]`, `D[2,2]`, …, `D[N,N]`) стоять нулі, у елементах, котрі відповідають парам комп'ютерів, між котрими є зв'язок, стоять одиниці, у решті елементів — машинні нескінченості. По мірі виконання алгоритму, у цьому ж масиві будується потрібна матриця найкоротших часів передавання повідомлень.

Детальніше про алгоритм Флойда–Уоршалла можна дізнатися у багатьох джерелах, зокрема, [КЛР, гл. 26.2] та [Лип, гл. 3.5]).

Порівнюючи ці підходи (багатократний пошук в ширину чи алгоритм Флойда–Уоршалла), бачимо що в умовах даної задачі пошук в ширину в середньому працює трохи швидше. Але на вхідних даних, де число $M$ (кількість ребер) дуже велике, швидшим виявляється алгоритм Флойда–Уоршалла. Крім того, алгоритм Флойда–Уоршалла можна було б застосувати навіть за умови, якби для кожного зв'язку між комп'ютерами задавався свій час передавання повідомлення (тоді як пошук в ширину в цьому випадку незастосовний).

# 3 Розв'язання задачі «Way»

Звичайно ж, ця задача предбачає не перебір усіх таких шляхів, а підрахунок їхньої кількості засобами комбінаторики.

Очевидно, перенесення системи координат в іншу точку, повороти на 90° і дзеркальні відображення відносно горизонталі та відносно вертикалі не змінюють кількості шляхів. Тому перемістимо початок координат у початкову точку, і надалі вважатимемо $x_2 \geqslant x_1$ (позначимо $x_2 - x_1 = a$), $y_2 \geqslant y_1$ (позначимо $y_2 - y_1 = b$), причому $a \geqslant b$.

Отже, рухатися потрібно праворуч–угору, причому більше праворуч ніж угору. Очевидно, що мінімальні шляхи такого руху як правило повинні містити ходи праворуч та праворуч–угору. Але крім того вони можуть містити і ходи праворуч–униз, бо послідовність ходів праворуч–угору та праворуч–униз дає той самий результат, що й два ходи праворуч.

Виділимо область, у котрій лежать усі найкоротші шляхи (див. мал. 1). Це перетин секторів, один з котрих іде з початкової вершини, інший — з кінцевої. Сектор початкової вершини у стовпчику номер $i$ займає клітинки від $(-i)$ до $i$, сектор кінцевої у стовпчику $a-k$ займає клітинки від $(b-k)$ до $(b+k)$. Отже, їхній перетин по стовпчику номер $i$ містить клітинки від $\max(-i, b-(a-i))$ до $\min(i, b+(a-i))$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 5 | | | | | | 1 | 6 | |
| 4 | | | | | 1 | 5 | 21 | 77 |
| 3 | | | | 1 | 4 | 15 | 50 | |
| 2 | | | 1 | 3 | 10 | 30 | | |
| 1 | | 1 | 2 | 6 | 16 | | | |
| 0 | 1 | 1 | 3 | 7 | | | | |
| -1 | | 1 | 2 | | | | | |

Малюнок 1: Область допустимих шляхів та кількості шляхів при $a = 7$, $b = 4$.

Позначимо за $T(m, n)$ кількість способів переміститися на $m$ клітинок праворуч і на $n$ клітинок угору. Виразимо $T(m, n)$ через ту саму функцію $T$, але від менших аргументів. Якщо $n = m = 0$, то $T(m, n) = 1$. Для інших клітинок вказаної області, останній хід оптимального маршруту можна зробити з однієї з попередніх: зліва–згори, зліва або зліва–знизу. Нема ні можливості прийти одночасно з кількох із цих напрямків, ні можливості прийти по якомусь іншому напрямку крім цих (так, щоб це був мінімальний шлях). Тому, кількість способів $T(m, n)$ дорівнює сумі $T(m-1, n+1)$, $T(m-1, n)$ та $T(m-1, n-1)$:

$$T(m,n) = T(m-1,n+1) + T(m-1,n) + T(m-1,n-1) \quad \begin{array}{l}\text{(рахуючи по тим із клі-}\\ \text{тинок, котрі входять до}\\ \text{виділеної області)}\end{array} . \qquad (6)$$

Отже, лишається запрограмувати цю формулу. Звичайно ж, її треба не програмувати рекурсивною функцією, а нарощувати в табличці значення від $m = n = 0$ до потрібних $a$ та $b$.

Як неважко переконатися при безпосередньому програмуванні, при вказаних в умові технічних обмеженнях ($|x_1|, |x_2|, |y_1|, |y_2| \leqslant 100$) значення далеко вибиваються за діапазон як `longint`, так і `QWord`. Використання типів з плаваючою точкою *не* призводить до повністю правильного розв'язання задачі, бо потрібно знайти *точну* кількість шляхів. Отже, потрібно реалізувати так звану довгу арифметику. Це означає, що числа необхідно подавати рядками або масивами (кожна цифра в окремому елементі), а арифметичні дії (в даному випадку лише додавання) писати самому, бажано оформлюючи як підпрограми.

Оскільки довгі числа займають багато пам'яті, табличка кількостей шляхів одночасно для усіх чисел з виділеної області досить велика. Тому при бажанні можна застосувати такий прийом: оскільки фомула (6) для побудови чергових значень використовує тільки значення попереднього стовпчика, достатньо в кожен момент зберігати лише два сусідніх стовпчики.

# 4 Розв'язання задачі «Digits»

Нам треба знайти число, кратне $K$, серед чисел

$$\begin{array}{rrrrrrrrr}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
11 & 22 & 33 & 44 & 55 & 66 & 77 & 88 & 99 \\
111 & 222 & 333 & 444 & 555 & 666 & 777 & 888 & 999 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
11\dots1 & 22\dots2 & 33\dots3 & 44\dots4 & 55\dots5 & 66\dots6 & 77\dots7 & 88\dots8 & 99\dots9 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots
\end{array} \tag{7}$$

Можна безпосередньо брати усі ці числа і перевіряти їхню подільність. Але цілком очевидні два недоліки такого способу. По-перше, ці числа дуже швидко вийдуть за межі допустимого діапазону і типу `longint`, і типу `QWord`. Можна було б застосувати довгу арифметику, але це досить складно для програмування, призводить до дуже повільної програми, і при всьому цьому ще й посилює другу проблему. Адже потрібно якось розпізнавати ситуацію, коли взагалі ніяке число вказаного вигляду не буде кратним вказаному $K$. Можна поставити яке-небудь евристичне обмеження, наприклад, вважати, що якщо не знайдено такого числа з кількістю цифр до 1000, то вважати що такого числа не існує. Але ж хотілось би мати чіткий алгоритм, відповіді котрого були б гарантовано правильними...

Нам потрібне не значення частки від ділення, а лише кратність. Інакше кажучи, потрібно з'ясовувати, чи дорівнює 0 залишок від ділення чисел вказаного вигляду на $K$. Як відомо, для «модулярної арифметики», або «арифметики залишків», мають місце співвідношення

$$(a+b) \bmod p = ((a \bmod p) + (b \bmod p)) \bmod p; \tag{8}$$

$$(a \cdot b) \bmod p = ((a \bmod p) \cdot (b \bmod p)) \bmod p. \tag{9}$$

Тому, залишки від ділення чисел з таблички (7) рахувати насправді набагато легше:

1. Залишок від ділення числá першого рядка і першого стовпчика відомий — це одиничка;
2. інші чи́сла першого стовпчика отримуються з числá першого стовпчика попереднього рядка як $N_m = 10N_{m-1} + 1$, тож відповідно обчислюємо і залишки:

$$r_m = (10r_{m-1}+1) \bmod K; \tag{10}$$

3. числа з не-перших стовпчиків отримуються з числá першого стовпчика того самого рядка шляхом домноження на значення відповідної цифри (2, 3, ..., 9).

Тепер уже неважко перейти і до другого питання — як з'ясовувати, що залишок ніколи не буде рівним 0?

Можна спробувати довести приблизно таке твердження: залишок при діленні чисел вказаного вигляду на $K$ ніколи не дорівнюватиме 0 тоді й тільки тоді, коли $K$ кратне 10, 16 або 25. Але ж зовсім неочевидно (і навіть невідомо), чи це твердження правильне (особливо та частина, що залишок ніколи не дорівнюватиме 0 *лише* для таких чисел)...

Тому розглянемо спосіб, менш красивий з суто математичної точки зору, але більш красивий з алгоритмічної. Коли ми перейшли до модулярної арифметики, ми зробили перелік усіх можливих значень скінченим. Тому неминуче, рано чи пізно, в залишках першого стовпчика таблиці (7) мусить повторитися значення, котре вже зустрічалося десь вище в цьому ж стовпчику. Але ж кожен наступний залишок цього стопчика цілком визначається через попередній за формулою (10); отже, тільки-но повториться хоча б одне значення, негайно почнеться точне циклічне повторення усієї послідовності значень. І якщо до початку такого циклу не було жодного нульового залишку, то його не буде взагалі.

Тут *не* стверджується, ніби повторитися має обов'язково значення *1-ого* рядка (контрприклад $1 \bmod 16 = 1$, $11 \bmod 16 = 11$, $111 \bmod 16 = 15$, $1111 \bmod 16 = 7$, $11111 \bmod 16 = 7$, $111111 \bmod 16 = 7$, ...). Не стверджується також, ніби обов'язково мусить настати ситуація, коли всі підряд залишки починаючи з якогось однакові (контрприклад $1 \bmod 30 = 1$, $11 \bmod 30 = 11$, $111 \bmod 30 = 21$, $1111 \bmod 30 = 1$, $11111 \bmod 30 = 11$, $111111 \bmod 30 = 21$, ...).

Зате абсолютно точно відомо, що обов'язково повториться *якесь* із значень, що раніше зустрічалися. Цих значень небагато (`1..K-1`), тож можна завести булевий масив розміром $K$, елементи котрого будуть відповідати твердженню «залишок такий-то вже зустрічався». І тільки-но отримуємо залишок, котрий уже зустрічався, кажемо що нульових залишків нема взагалі.

# 5 Розв'язання задачі «Lamps»

# 6 Посилання на літературу

[КЛР] — Т. Кормен, Ч. Лейзерсон, Р. Ривест. «Алгоритмы: построение и анализ». М., МЦНМО, 1999 (+ дополнительные тиражи следующих лет) — 960 стр.

[Лип] — В. Липский, «Кобинаторика для программистов». М., Мир, 1988 — 212 стр.