Завдання XXVIII Всеукраїнської учнівської олімпіади з інформатики

2014/15 н. р.

Данило Мисак, Шаміль Ягіяєв

Зміст

Завдання першого туру

1.1. Мутація (Роман Рубаненко)

Умова

Вчені планети Олімпія кожен рік досліджують різноманітні мутації геномів примітивних організмів. Геном таких організмів може бути представлений як послідовність N невід'ємних цілих чисел, які занумеровані зліва направо від одиниці до N та не перевищують число N. Геноми підлягають постійним мутаціям. На кожному етапі мутації геном змінюється таким чином:

- на перше місце записується кількість одиниць у вхідному геномі;
- на друге місце записується кількість двійок у вхідному геномі;
- ...,
- на місце номер N записується кількість чисел, які дорівнюють N, у вхідному геномі.

Наприклад, геном $[1,\ 2,\ 3]$ з трьох чисел після мутації перетвориться на $[1,\ 1,\ 1]$ — по одній одиниці, двійці та трійці. Інші приклади:

- $[1, 2, 2, 3, 3, 3] \to [1, 2, 3, 0, 0, 0]$.
- $[7, 7, 7, 4, 7, 4, 4] \to [0, 0, 0, 3, 0, 0, 4]$.

Далі геном продовжує змінюватися за тим самим принципом.

Завдання. Напишіть програму `mutation`, яка за інформацією про початковий вигляд геному визначить його стан після K мутацій.

Вхідні дані. Перший рядок вхідного файла `mutation.dat` містить два цілих числа N і K ($1 \leqslant N \leqslant 10^5$, $1 \leqslant K \leqslant 10^9$), що задають початковий розмір геному та кількість мутацій, які геном переживе. Другий рядок містить N невід'ємних цілих чисел, що не перевищують N, — початковий вигляд геному.

Вихідні дані. У вихідний файл `mutation.sol` слід записати геном після K мутацій у тому ж форматі, що й у вхідному файлі: N чисел, розділені пропуском.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 40 % балів: $1 \leqslant N \leqslant 100$, $1 \leqslant K \leqslant 100$.
- 30 % балів: $1 \leqslant N \leqslant 1000$.
- 30 % балів: $1 \leqslant N \leqslant 10^5$.

Приклад вхідних та вихідних даних.

mutation.dat	mutation.sol
4 2 1 3 1 4	2 1 0 0

Пояснення. Спочатку $[1, 3, 1, 4]$ мутує в геном $[2, 0, 1, 1]$, який у свою чергу мутує в $[2, 1, 0, 0]$.

Розв'язання

Лема. Якщо геном містить принаймні одне додатне число, то через $O(\log\log N)$ мутацій він набуде вигляду $[1, 0, 0, \dots, 0]$ та перестане змінюватися.

Доведення. Розглянемо дві ітерації послідовних мутацій: $A \to B \to C$. Нехай $z(X)$ позначає кількість ненульових елементів у геномі X. Тоді $z(C)$ — кількість різних ненульових елементів B, тобто кількість різних частот ненульових чисел геному A. Щоб отримати $z(C) = P$ різних частот, потрібно мати хоча б $1 + 2 + \dots + P = P(P+1)/2$ ненульових елементів. Таким чином, $z(A) \geqslant P(P+1)/2 > z^2(C)/2$, звідки $z(C) < \sqrt{2z(A)}$. Оскільки $\sqrt{2z(A)} \leqslant z(A)$ при $A \geqslant 2$, кількість ненульових елементів у геномі кожні дві ітерації зменшується, і рано чи пізно в ньому залишиться тільки один ненульовий елемент. А тоді вже за два кроки геном набуде вигляду $[1, 0, 0, \dots, 0]$. Залишається оцінити кількість ітерацій, після яких це станеться.

Якщо після двох ітерацій кількість ненульових елементів стає меншою за $\sqrt{2z(A)}$, то після $2K$ ітерацій вона стане меншою за $\sqrt{2\sqrt{2\dots\sqrt{2\sqrt{2z(A)}}}}$, де кількість радикалів дорівнює K. Оскільки $\sqrt{2\sqrt{2\dots\sqrt{2\sqrt{2}}}} < 2$ (це нескладно довести за індукцією), $\sqrt{2\sqrt{2\dots\sqrt{2\sqrt{2z(A)}}}} < 2(z(A))^{1/2^K}$.

Маємо $z(A) \leqslant N$, тому після $K = O(\log\log N)$ ітерацій число $\frac{1}{2^K}$ стане меншим за $\frac{1}{\log N}$, а логарифм числа $(z(A))^{1/2^K}$ — меншим за одиницю. Залишається додати сталу кількість ітерацій, щоб дійти до фінального вигляду. ■

Моделювання однієї ітерації може бути просто реалізовано за лінійну складність відносно N з використанням допоміжного масиву. Умовна реалізація на псевдокоді може виглядати так:

```
b[] = [0, 0, ..., 0]
for i = 1 to N do
    b[a[i]] = b[a[i]] + 1
```

Після кожної ітерації треба перевірити, чи перетворився геном на $[1, 0, 0, \dots, 0]$. Якщо це так, то подальше виконання програми не має сенсу: геном залишатиметься незмінним. В результаті програма виконає $O(N \cdot \min(K, \log\log N))$ операцій.

1.2. Лінія (Данило Мисак)

Умова

Орися розставила на аркуші в клітинку N^2 літер у формі квадрата $N \times N$ і хоче викреслити однією лінією деякі літери у такий спосіб: вона починає викреслювати літери, починаючи з лівої верхньої букви, і веде лінію то праворуч, то вниз; останньою літерою вона викреслює праву нижню. Таким чином, дівчина викреслить рівно $2N - 1$ літеру. При цьому Орися хоче, щоб уздовж лінії, яку вона проведе, було записано певне чарівне слово.

Завдання. Напишіть програму `line`, яка для заданих розташування літер і чарівного слова визначить, у скільки різних способів Орися може його викреслити, та виведе відповідь за модулем простого числа $1\,000\,003$.

Вхідні дані. У першому рядку вхідного файла `line.dat` записано натуральне число N ($2 \leqslant N \leqslant 1000$) — довжину сторони квадрата з літер. У наступних N рядках записано по N малих літер латинської абетки (не обов'язково різних), що задають розташування літер. Пробілів між літерами немає. Далі записано чарівне слово, що складається з $2N - 1$ літери (усі — малі літери латинської абетки, не обов'язково різні).

Вихідні дані. Вихідний файл `line.sol` повинен містити єдине число — остачу від ділення кількості способів, у які Орися може викреслити чарівне слово, на число $1\,000\,003$.

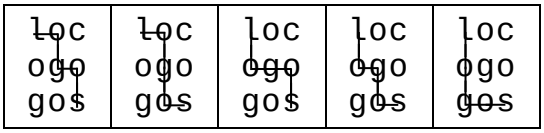
Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 25 % балів: $2 \leqslant N \leqslant 10$.
- 30 % балів: $10 < N \leqslant 100$.
- 45 % балів: $100 < N \leqslant 1000$.

Приклад вхідних та вихідних даних.

line.dat	line.sol
3 loc ogo gos logos	5

Пояснення. Є рівно 5 способів викреслити слово `logos`:



Розв'язання

Будемо вважати, що літери записано в комірках квадратної таблиці, а замість ліній розглядатимемо шляхи з лівої верхньої клітинки, що йдуть праворуч та вниз. *Гарним* шляхом з лівої верхньої клітинки в довільну клітинку з літерою називатимемо такий шлях, уздовж якого послідовність літер збігається з початком чарівного слова — тобто який теоретично можна продовжити до правильного шляху в праву нижню клітинку (але лише теоретично: якщо такого шляху насправді немає, бо внизу і справа не знайдеться потрібних літер, шлях з лівої верхньої клітинки в дану все одно вважаємо гарним).

Зауважимо, що якщо клітинка з літерою розташована на перетині i-го рядка та j-го стовпця, то шлях у неї з лівої верхньої клітинки матиме довжину $i + j - 1$ незалежно від того, як саме він проходить. Отже, в послідовності літер ця клітинка відповідатиме $(i + j - 1)$-й літері чарівного слова. Якщо літера, записана в даній комірці, відрізняється від літери з цим номером чарівного слова, кількість гарних шляхів у дану клітинку нульова, адже остання літера на шляху завжди буде неправильною. Якщо натомість літера в даній комірці збігається з $(i + j - 1)$-ю літерою чарівного слова, то кількість гарних шляхів у неї дорівнює сумі кількості гарних шляхів у клітинку-сусіда ліворуч від даної і кількості гарних шляхів у клітинку-сусіда вище від даної. Справді: будь-який гарний шлях входить у дану клітинку або зліва, або зверху, причому він мав бути гарним і на попередньому кроці. І навпаки: якщо є якийсь гарний шлях, що закінчується в сусідній зліва або зверху клітинці, то його можна продовжити на одну літеру вправо або вниз відповідно.

Окремо слід розглянути випадок клітинок на верхній та лівій межах поля. Для клітинок у верхньому рядку кількість гарних шляхів дорівнює просто кількості гарних шляхів у клітинку зліва від даної. Аналогічно для комірок лівого стовпця кількість гарних шляхів дорівнює кількості таких шляхів у клітинку зверху від даної. Звичайно, за умови, що в даній клітинці стоїть літера, яка збігається з відповідною буквою чарівного слова.

Фактично нас цікавить кількість гарних шляхів у праву нижню клітинку. Щоб знайти її, послідовно порахуємо цю кількість для всіх клітинок поля, починаючи з лівої верхньої та йдучи зліва направо зверху вниз. Для лівої верхньої клітинки кількість гарних шляхів дорівнює або 1, якщо літера в ній збігається з першою літерою чарівного слова, або 0, якщо не збігається. Далі кількості шляхів можна рахувати за описаною вище схемою, адже на момент, коли ми розглядаємо довільну комірку, клітинки зліва та зверху від неї, якщо такі є, вже було розглянуто раніше.

Таким чином, у задачі застосовано класичний підхід динамічного програмування. Щоб уникнути розгляду додаткових випадків — коли клітинка розташована у верхньому рядку або лівому стовпці — можна ввести фіктивний додатковий рядок зверху та стовпець зліва від поля, для всіх комірок яких кількості гарних шляхів покласти рівними нулям.

Не слід, звичайно, забувати, що всі операції виконуються за модулем числа, вказаного в умові.

1.3. Східний кросворд (Данило Мисак)

Умова

Марися любить розв'язувати східні кросворди. Так називається головоломка, в якій потрібно зафарбувати деякі клітинки прямокутника $N \times M$ таким чином, щоб на кожній з N вертикалей і на кожній з M горизонталей кількість зафарбованих клітинок дорівнювала деякому наперед визначеному записаному на полях для даної вертикалі чи горизонталі числу. На жаль, інколи укладачі кросвордів помиляються, і кросворд розв'язку не має. Дівчина не хотіла б марнувати свій час, розв'язуючи такі кросворди.

Завдання. Напишіть програму `puzzle`, яка для заданих величин N та M, а також $N + M$ чисел, записаних на полях кросворда, визначить, чи є даний кросворд розв'язним.

Вхідні дані. У першому рядку вхідного файла `puzzle.dat` записано число T, $1 \leqslant T \leqslant 5$, — кількість кросвордів для перевірки. Кожен кросворд подається трьома рядками: в першому рядку записано натуральні числа N та M, що не перевищують 10^5, — ширину та висоту кросворда; у другому рядку подано N цілих невід'ємних чисел — кількість зафарбованих клітинок на першій, другій, ..., N-й вертикалях відповідно; у третьому рядку подано M цілих невід'ємних чисел — кількість зафарбованих клітинок на першій, другій, ..., M-й горизонталях відповідно. Жодне з чисел у другому і третьому рядках не перевищує загальної кількості клітинок на відповідній вертикалі чи горизонталі.

Вихідні дані. Вихідний файл `puzzle.sol` повинен містити відповідь для кожного з кросвордів в окремому рядку: `1`, якщо кросворд розв'язний, або `0`, якщо ні. Відповіді потрібно подати в тому ж порядку, в якому у вхідному файлі подано самі кросворди.

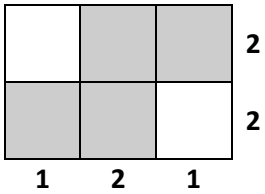
Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 15 % балів: жоден з розмірів (ні ширина, ні висота) жодного з кросвордів не перевищує 5.
- 35 % балів: жоден з розмірів жодного з кросвордів не перевищує 1000, причому хоча б один з розмірів хоча б одного з кросвордів більший за 5.
- 50 % балів: хоча б один з розмірів хоча б одного з кросвордів більший за 1000.

Приклад вхідних та вихідних даних.

puzzle.dat	puzzle.sol
2 3 2 1 2 1 2 2 4 5 5 5 5 5 0 0 0 0 0	1 0

Пояснення. У першому випадку кросворд має, наприклад, таке розв'язання:



У другому випадку кросворд нерозв’язний: з одного боку, рядок з п’ятірок свідчить, що всі вертикалі повністю зафарбовано; з іншого боку, рядок з нулів означає, що жодна з горизонталей не містить жодної зафарбованої клітинки. Такого, очевидно, бути не може.

Розв'язання

Замість «кросворда» казатимемо «матриця», замість «зафарбованих клітин» — «одиниці», а замість «незафарбованих клітин» — нулі. Позначимо кількість одиниць в i-му стовпці через c_i, $1 \leqslant i \leqslant N$, а кількість одиниць в i-му рядку — через r_i, $1 \leqslant i \leqslant M$. Уважатимемо, що $\sum_{i=1}^{N} c_i = \sum_{i=1}^{M} r_i$ (обидві суми дорівнюють загальній кількості одиниць у матриці). Інакше можна відразу стверджувати, що потрібної матриці не існує.

Розглянемо набір з перших K стовпців. Для кожного з них ми знаємо, скільки в даному стовпці має бути одиниць, а отже, і загальну кількість одиниць у цих стовпцях: $\sum_{i=1}^{K} c_i$. З іншого боку, в i-му рядку в цих стовпцях буде не більше ніж $\min(r_i, K)$ одиниць, тому загальна кількість одиниць у цих стовпцях не може перевищувати $\sum_{i=1}^{M} \min(r_i, K)$. Звідси маємо необхідну умову для існування матриці чисел, що задовольняє умову задачі:

$$\sum_{i=1}^{K} c_i \leqslant \sum_{i=1}^{M} \min(r_i, K)\,, 1 \leqslant K \leqslant N.$$

Дану умову позначимо через (*).

Оскільки завжди без втрати загальності можна переставити будь-які стовпці місцями, можемо розставити стовпці в порядку спадання (незростання) кількості одиниць у них. Таким чином ми досягнемо того, що для довільного K сума $\sum_{i=1}^{K} c_i$ буде максимально можливою, а виведена вище умова необхідності — максимально суворою. При цьому для зручності вважатимемо, що рядки також впорядковані, але за зростанням (неспаданням) кількості одиниць, тобто r_M — максимальний рядок, r_{M-1} — другий за величиною і т. д.

Переформулюємо виведену умову без алгебри: для довільного K, $1 \leqslant K \leqslant N$, сума K найбільших кількостей по стовпцях не перевищує суму всіх кількостей по рядках, якщо всі ці кількості попередньо «обрізати» до K, тобто всі числа, більші за K, зменшити до K. Покажемо, що ця умова є й достатньою для існування відповідної матриці. Для цього заповнимо перший стовпець (у якому має бути найбільша кількість одиниць) так: поставимо одиниці в тих c_1 рядках, де загальна кількість одиниць має бути найбільшою. По-перше, це зробити можливо, адже якщо в (*) підставити $K = 1$, матимемо, що c_1 не перевищує кількості ненульових рядків. По-друге, коли ми це зробимо, то перейдемо до меншої підзадачі: сконструювати матрицю вже з $N - 1$ стовпцями з сумами $c'_1 = c_2$, $c'_2 = c_3$, ..., $c'_{N-1} = c_N$ і з M рядками з сумами $r'_1 = r_1$, $r'_2 = r_2$, ..., $r'_{M-c_1} = r_{M-c_1}$, $r'_{M-c_1+1} = r_{M-c_1+1} - 1$, $r'_{M-c_1+2} = r_{M-c_1+2} - 1$, ..., $r'_M = r_M - 1$ (при цьому впорядкованість r'_1, r'_2, ..., r'_M за неспаданням може порушитись, але це не впливає на правильність розв'язання). Якщо ми покажемо, що відповідні числа знову задовольняють (*), такі ж операції можна буде продовжити, поки ми не побудуємо матрицю повністю. Отже, нам треба довести таку нерівність:

$$\sum_{i=1}^{K} c'_i \leqslant \sum_{i=1}^{M} \min(r'_i, K)\,, 1 \leqslant K \leqslant N - 1.$$

Знайдемо таке найменше $C \geqslant 1$, що серед чисел r_1, r_2, ..., r_M таких, які більші або рівні C, буде не менше за c_1, а чисел, які більші або рівні $C + 1$, — не більше за c_1. Таке C обов'язково

знайдеться і буде не більшим за N, адже за умовою (*) при $K = 1$ чисел, не менших за 1, є не менше за c_1, а чисел, більших за N, бути не може в принципі.

Тоді для $K < C$ маємо:

$$\sum_{i=1}^{K} c'_i = \sum_{i=2}^{K+1} c_i \leqslant K \times c_1 = \sum_{i=M-c_1+1}^{M} \min(r_i - 1, K) = \sum_{i=M-c_1+1}^{M} \min(r'_i, K) \leqslant \sum_{i=1}^{M} \min(r'_i, K).$$

Водночас для $K \geqslant C$:

$$\begin{aligned}
\sum_{i=1}^{K} c'_i = \sum_{i=2}^{K+1} c_i = \sum_{i=1}^{K+1} c_i - c_1 \leqslant \sum_{i=1}^{M} \min(r_i, K+1) - c_1 = \\
= \sum_{i=1}^{M-c_1} \min(r_i, K+1) + \sum_{i=M-c_1+1}^{M} \min(r_i, K+1) - c_1 = \\
= \sum_{i=1}^{M-c_1} \min(r_i, K) + \left(\sum_{i=M-c_1+1}^{M} \min(r_i, K+1) - c_1 \right) = \\
= \sum_{i=1}^{M-c_1} \min(r_i, K) + \sum_{i=M-c_1+1}^{M} \min(r_i - 1, K) = \\
= \sum_{i=1}^{M-c_1} \min(r'_i, K) + \sum_{i=M-c_1+1}^{M} \min(r'_i, K) = \sum_{i=1}^{M} \min(r'_i, K).
\end{aligned}$$

Отже, залишається відсортувати підрахунком, а точніше — підрахувати індексні масиви для обох послідовностей: кількостей одиниць по стовпцях і по рядках. Якщо в масиві кількостей по стовпцях достатньо йти і щораз додавати до суми S_c нові елементи в порядку їх спадання, то для масиву кількостей по рядках потрібно зберігати кількість на поточний момент часу чисел, не менших за дане (оновлювати його можна, щораз віднімаючи кількість чисел, які дорівнюють даному), та на кожному кроці додавати цю кількість до суми S_r. Далі порівнювати S_c та S_r.

1.4. Дерева (Роман Фурко)

Умова

У столиці країни Олімпія було визначено територію для будівництва нового Олімпійського парку. Границі території мають форму опуклого багатокутника. Ідея дизайнера парку полягає в тому, щоб засадити деревами певну кількість зелених зон, які мають на карті форму круга: кожну зелену зону можна задати координатами центра та її радіусом.

Отже, дизайнер доручив посадити дерева у точках, що мають цілі координати та розташовані в межах парку (можливо, на його границі) та хоча б однієї з зелених зон (можливо, на межі). Якщо певні зелені зони перетинаються, тобто одна й та сама точка на карті належить двом чи більшій кількості зелених зон, у цьому місці тим не менш можна посадити лише одне дерево.

Завдання. Напишіть програму `trees`, що за даними про координати вершин многокутника, який задає територію парку, а також за даними про координати центрів та радіуси зелених зон визначить кількість дерев, які має бути посаджено.

Вхідні дані. У першому рядку вхідного файла `trees.dat` вказано натуральне число N ($3 \leqslant N \leqslant 10^5$) — кількість вершин багатокутника, що задає територію парку. Наступні N рядків містять по два цілих числа — відповідно абсциси та ординати вершин у порядку обходу за чи проти годинникової стрілки. Наступний рядок містить ціле число M ($1 \leqslant M \leqslant 50\,000$) — кількість зелених зон. Далі йдуть M рядків, кожен з яких містить по три цілих числа: абсцису та ординату центра зеленої зони, а також її радіус. Усі координати, задані у вхідному файлі, є цілими числами в межах від -10^9 до 10^9 включно. Радіуси зелених зон цілі додатні, сума всіх радіусів не перевищує 10^5. Зверніть увагу, що деякі зелені зони можуть цілком лежати всередині інших зон; крім того, окремі зелені зони можуть лежати поза многокутником. Різні зони можуть мати спільні центри або й узагалі збігатися.

Вихідні дані. Єдиний рядок вихідного файла `trees.sol` повинен містити єдине ціле число — кількість дерев, що будуть посаджені за дорученням дизайнера.

Оцінювання. Набір тестів складається з блоків, для яких додатково виконуються такі умови:

- 60 % балів: $N \leqslant 100$. Зокрема, серед даного набору є такі групи тестів, що перетинаються:
  - 20 % балів (від загальної кількості): парк має форму прямокутника, сторони якого паралельні осям координат.
  - 30 % балів (від загальної кількості): $N \times M \times R^2 \leqslant 10^7$, де через R позначено найбільший з радіусів.
  - 25 % балів (від загальної кількості): існує квадрат зі сторонами довжини 2015, паралельними до осей координат, у якому або на межах якого повністю лежить територія парку.
- 40 % балів: на вхідні дані не накладено додаткових обмежень.

Приклад вхідних та вихідних даних.

trees.dat	trees.sol
3 0 0 15 0 15 15 3 3 3 4 5 5 7 15 15 1	73

Розв'язання

Спочатку розглянемо задачу ефективного знаходження перетину опуклого многокутника та вертикальної прямої. Очевидно, що кількість точок перетину не перевищує 2, якщо тільки пряма не містить вертикальну сторону многокутника. Розіб'ємо многокутник на верхній і нижній ланцюги: знаходимо найлівішу (а з усіх таких — найнижчу) та найправішу (з усіх таких — найвищу) вершину многокутника. Проводимо пряму між цими точками. Всі вершини многокутника, що лежать вище від проведеної прямої, належать верхньому ланцюгу, а решта — нижньому. Після такого поділу в кожному ланцюзі вершини впорядковані за абсцисою. Це дозволяє знаходити точки перетину вертикальної прямої з опуклим многокутником за $O(\log N)$ операцій за допомогою двійкового пошуку.

Тепер, оскільки круг довільного натурального радіуса R перетинається рівно з $2R + 1$ цілочисельною вертикальною прямою, загальна кількість таких прямих, що перетинаються принаймні з одним кругом, не перевищує $2S + M = O(S)$, де S — сумарний радіус усіх M кругів. Розглянемо відповідні $2S + M$ відрізків перетину та розташуємо їх у порядку зростання абсциси, а при однаковій абсцисі — за зростанням ординати нижнього кінця. Для кожного відрізка за $O(\log N)$, скориставшись описаним у попередньому абзаці методом, знайдемо кількість цілих точок на відрізку, що лежать усередині многокутника. При цьому, щоб не рахувати по кілька разів точки, які лежать водночас на кількох відрізках, для кожної нової абсциси пам'ятатимемо ординату, на якій закінчився попередній розглянутий відрізок і починаючи з якої, відповідно, потрібно продовжувати рахувати точки. Сумарна знайдена кількість точок і буде відповіддю.

Складність алгоритму — $O(S(\log S + \log N) + N)$ або, якщо скористатися впорядкованістю відрізків і замість двійкового пошуку йти по ланцюгах лінійно, $O(S \log S + N)$.

Альтернативні розв'язання. Розв'язок за $O(S(\log S + N))$ зі знаходженням кількості точок відрізка, що лежать усередині многокутника, за лінійний від кількості вершин многокутника час набирає 60 % балів. Якщо перебирати всі точки всередині кругів та наївно перевіряти, чи лежать вони в багатокутнику, можна набрати 30 % балів. А якщо перебирати всі точки всередині многокутника та перевіряти, чи лежать вони в межах принаймні одного круга, вдасться заробити лише 20 % балів.

Завдання другого туру

2.1. Шоколад (Віталій Герасимів)

Умова

У Марічки та Зеника є прямокутна плитка смачного українського шоколаду. Плитка складається з однакових квадратиків, кожен з яких може бути або чорним, або білим.

Марічка проведе довільну (можливо, нульову) кількість повних горизонтальних (зліва направо) розрізів плитки, тоді як Зеник — довільну (можливо, нульову) кількість повних вертикальних (зверху вниз) розрізів. Зауважте, що розрізи можна проводити тільки між квадратиками плитки, а також що Зеник і Марічка можуть зробити різну кількість розрізів. Після проведення всіх розрізів плитка розпадеться на певну кількість прямокутних шматків, які наші герої певним чином розділять між собою.

Оскільки Зеник дуже любить чорний шоколад, він хоче, щоб після всіх розрізів кількість цілих шматків, які складаються виключно з чорних квадратиків, була якомога більшою. Марічка хоче цю саму кількість зробити якомога меншою. Зауважте, що героїв не цікавить, якого саме розміру будуть шматочки: головне — їхня кількість.

Щоб усе було чесно, Андрій окремо та незалежно запитає Марічку та Зеника, в яких саме місцях потрібно провести розрізи, а потім сам замість них проведе ці розрізи.

Завдання. Напишіть програму `chocolate`, яка за заданими розмірами плитки, а також типом кожного її одиничного квадратика визначить, скільки після всіх розрізань утвориться шматочків, що складаються виключно з чорних квадратиків, — за умови оптимальних дій обох героїв.

Вхідні дані. Перший рядок вхідного файла `chocolate.dat` містить два цілих числа N і M ($2 \leqslant N \leqslant 1000$, $2 \leqslant M \leqslant 1000$): висоту та ширину плитки відповідно. Далі йдуть N рядків по M символів у кожному, які позначають тип відповідного одиничного квадратика шоколадки. Символ B позначає чорний квадратик, а символ W — білий.

Вихідні дані. Єдиний рядок вихідного файла `chocolate.sol` повинен містити одне ціле число — кількість шматків, які складатимуться виключно з квадратиків чорного шоколаду.

Оцінювання. Набір тестів складається з 2 блоків:

- 40 % балів: $N \leqslant 8$, $M \leqslant 8$.
- 60 % балів: на вхідні дані не накладають додаткових обмежень.

Приклад вхідних та вихідних даних.

chocolate.dat	chocolate.sol
3 4 BWBB BBBW BBBW	2

Розв’язання

Зенику вигідно провести всі $M - 1$ вертикальних розрізів, а Марічці — не проводити жодного горизонтального, адже додаткові розрізи точно не зменшать кількість плиток, що складаються виключно з чорних квадратиків. Отже, відповідь — кількість стовпців, що повністю складаються з чорного шоколаду. Її можна знайти за один лінійний прохід вхідних даних.

2.2. Гурток (Сергій Оришич)

Умова

У деякій школі країни Олімпія проводиться гурток з інформатики, в якому займаються N досвідчених програмістів та N новачків. Тренер будує заняття, орієнтуючись на роботу в парі досвідченого учня та новачка. Провівши тренування, тренер визначив ефективність співпраці i-го досвідченого програміста з j-м новачком, що виражається числом a_{ij}. Загальна ефективність роботи гуртка дорівнює сумарному показнику ефективності співпраці для всіх N пар за умови, що кожен учень працюватиме в парі, причому тільки в одній. Тренер хоче періодично проводити ротації пар, тому його цікавить питання: чи при довільному розбитті на пари ефективність роботи гуртка буде однаковою.

Завдання. Напишіть програму `section`, що за інформацією про ефективність співпраці кожної пари з досвідченого учня та новачка визначатиме, чи ефективність роботи гуртка відрізнятиметься залежно від того, як учнів розбито на пари.

Вхідні дані. Перший рядок вхідного файла `section.dat` містить єдине натуральне число K ($1 \leqslant K \leqslant 50$) — кількість тестів у файлі. Далі йде опис K різних гуртків: в окремому рядку записано натуральне число N ($2 \leqslant N \leqslant 100$) — кількість досвідчених програмістів та новачків у гуртку; потім іде N рядків по N цілих чисел через пропуск: j-те число в i-му рядку дорівнює a_{ij} ($0 \leqslant a_{ij} \leqslant 20\,000$) — ефективність роботи в парі i-го досвідченого програміста з j-м новачком.

Вихідні дані. Вихідний файл `section.sol` має містити K чисел, записаних по одному в рядку: i-те з них ($1 \leqslant i \leqslant K$) — сумарна ефективність i-го гуртка, якщо вона не залежить від розподілу учнів на пари, або `-1` в іншому випадку.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 30 % балів: $2 \leqslant N \leqslant 10$ для всіх гуртків.
- 20 % балів: $10 < N \leqslant 50$ для всіх гуртків.
- 50 % балів: $50 < N \leqslant 100$ для всіх гуртків.

Приклад вхідних та вихідних даних.

section.dat	section.sol
2 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 4 1 2 3 4 5 6 7 8 8 10 11 12 13 14 15 16	34 -1

Розв'язання

Віднімемо від кожного числа таблиці $N \times N$ перше число того ж рядка. Після таких операцій, наприклад, перший приклад з умови набуде такого вигляду:

```
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
```

Другий приклад з умови стане таким:

```
0 1 2 3
0 1 2 3
0 2 3 4
0 1 2 3
```

Необхідною і достатньою умовою для того, щоб ефективність гуртка завжди була однаковою, є збіг усіх рядків отриманої матриці. Справді: якщо всі рядки однакові, сумарна ефективність завжди дорівнюватиме сумі чисел одного рядка нової матриці та всіх перших чисел рядків початкової (у цьому нескладно переконатися, розписавши числа початкової таблиці як суми початкового першого числа рядка та нового поточного). З іншого боку, нехай деякі два числа у різних рядках нової матриці різні. Це не можуть бути числа в першому стовпці, адже всі числа в першому стовпці — нулі. Нехай це числа в j-му стовпці в рядках i_1 та i_2. Розіб'ємо учнів на пари довільним чином так, щоб i_1-й досвідчений програміст працював у парі з першим новачком, а i_2-й — з j-м. Запам'ятаємо сумарну ефективність гуртка. Тепер перерозіб'ємо ці дві пари: i_1-го досвідченого програміста поставимо працювати з j-м новачком, а i_2-го — з першим новачком. Оскільки відповідні числа були різними, сумарна ефективність цих двох пар зміниться, а отже, зміниться і загальна ефективність гуртка. Таким чином, вона дійсно залежить від розбиття учнів на пари.

Умову рівності всіх рядків можна перевірити за один лінійний прохід вхідних даних: запам'ятовуємо різниці для першого рядка і кожен наступний порівнюємо з ним.

2.3. Перестановка (Данило Мисак)

Умова

Уявімо, що на дошці записано деяку перестановку чисел від 1 до N, а в Петриковому зошиті — кілька впорядкованих пар чисел, причому кожне число в межах від 1 до N і числа в парі не можуть бути однаковими. Якщо на дошці поряд стоять два числа, які в тому ж порядку утворюють одну з пар у зошиті, будь-яке з цих двох чисел на дошці Петрик може витерти. При цьому Петрик вважає, що чи́сла, які розділяло витерте число, тепер стоять поряд.

Завдання. Напишіть програму `permutation`, що для заданого набору пар чисел, виписаних у Петриковому зошиті, будує приклад початкової перестановки чисел, яку хлопець зможе шляхом послідовних витирань звести до єдиного (довільного) числа, або встановлює, що такої перестановки не існує.

Вхідні дані. У першому рядку вхідного файла `permutation.dat` записано натуральні числа N та M ($2 \leqslant N \leqslant 10^5$, $2 \leqslant M \leqslant 10^5$), де M — кількість упорядкованих пар, записаних у зошиті Петрика. Наступні M рядків містять по два числа — елементи відповідних пар. Кожне число натуральне, не перевищує N та відмінне від іншого числа в парі. Серед заданих упорядкованих пар немає однакових.

Вихідні дані. У перший рядок вихідного файла `permutation.sol` слід вивести будь-який приклад перестановки чисел від 1 до N, яку Петрику вдасться звести до одного числа, а у другий рядок потрібно вивести послідовність з $N - 1$ числа, які одне за одним витиратиме Петрик. У випадку, якщо потрібної перестановки не існує, у вихідний файл треба вивести єдине число `0`.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 20 % балів: $2 \leqslant N \leqslant 5$.
- 35 % балів: $5 < N \leqslant 1000$.
- 45 % балів: $1000 < N \leqslant 10^5$.

Приклад вхідних та вихідних даних.

permutation.dat	permutation.sol
4 4 1 4 4 3 1 2 3 4	1 3 4 2 3 4 1

Пояснення. У перестановці $1\ 3\ 4\ 2$ можемо витерти трійку, бо маємо пару $3\ 4$. Далі послідовність чисел на дошці стає такою: $1\ 4\ 2$. Тепер можемо витерти четвірку, бо маємо пару $1\ 4$. На дошці залишається пара чисел $1\ 2$. Оскільки ця пара також є в Петриковому зошиті, можемо витерти, наприклад, одиницю, залишивши на дошці єдине число (в даному випадку — 2).

Розв'язання

Розглядатимемо неорієнтований граф на N вершинах, ребро між вершинами v та w якого проведено тоді й лише тоді, коли серед записаних у вхідному файлі пар є або (v, w), або (w, v), або обидві. Задачу можна розв'язати, скориставшись пошуком у ширину чи пошуком у глибину на такому графі. При цьому можна використати два підходи: або жадібний (з пошуком у ширину чи в глибину), або рекурсивний (тут підійде лише пошук у глибину).

Розв'язання за допомогою **жадібного підходу** таке: спочатку маємо послідовність з одного-єдиного (довільного) числа a, після чого додаємо до неї числа в тому порядку, в якому проходимо їх пошуком у ширину чи в глибину (початкова вершина пошуку — число a). Кожне нове число ми відразу ж розміщуємо в послідовності або безпосередньо зліва, або безпосередньо справа від «суміжного» числа, з якого ми до даного числа дійшли. Зліва чи справа розміщувати число, визначається тим, у якому порядку дані два числа утворювали пару у вхідному файлі. Якщо граф зв'язний, утворена в кінці послідовність буде перестановкою чисел від 1 до N і, здійснивши операції у зворотному порядку, зможемо витерти з неї всі числа, крім останнього (числа a). А якщо граф незв'язний, то потрібної перестановки існувати не може. У цьому випадку утворена послідовність міститиме менше за N елементів, що означатиме, що у вихідний файл слід вивести нуль.

Якщо при додаванні кожного нового числа перебудовувати всю послідовність (принаймні з місця, куди додано число), матимемо складність $O(N^2)$. Замість цього можна зберігати не саму поточну послідовність, а індексний масив, у якому для кожного числа вказано число зліва від нього та справа від нього (або індикатор того, що зліва чи справа від числа немає іншого числа). Така структура подібна до двозв'язного списку. Для того щоб її модифікувати, додаючи нове число, не потрібно робити пошук: достатньо взяти інформацію з комірки індексного масиву, що відповідає числу, поряд з яким буде додано нове. Залежно від реалізації цей підхід вимагає $O(N + M)$ або $O(M)$ часу.

Інший спосіб розв'язати задачу полягає в **рекурсивному нарощуванні** масивів з перестановкою та порядком витирання чисел для піддерев. Почавши з довільної вершини-числа, робимо дії в такому порядку: рекурсивно заглиблюємось в усі вершини, що відповідають числам, які стоять у парах ліворуч від даного; додаємо до порядку витирання чисел усі ті дочірні вершини, в які ми заглиблювались, але в зворотному порядку; додаємо до перестановки-результату поточне число; рекурсивно заглиблюємось в усі вершини, що відповідають числам, які стоять у парах праворуч від даного, і відразу після заглиблення в кожну нову вершину і повернення звідти додаємо цю вершину до порядку витирання чисел. Якщо в результаті до перестановки додано всі N чисел, маємо відповідь. Інакше граф незв'язний, а у вихідний файл треба вивести нуль. Як і у випадку з жадібним підходом, час виконання алгоритму складає $O(N + M)$ або $O(M)$.

Зауважимо, що через обмеження, яке накладає компілятор GCC на деяких архітектурах, програма, що використовує рекурсію, може не витримати великого рівня заглиблення на одному-двох тестах. Це стосується як рекурсивного підходу, так і жадібного з використанням пошуку в глибину.

Насамкінець пропонуємо читачу самостійно розв’язати пов’язану задачу, з якою довелося зіткнутися автору при підготовці чекера: перевірити правильність виведення програми учасника за час $O(M)$.

2.4. Рядковий автомат (Данило Мисак)

Умова

Рядковий автомат «Гомер-2015» приймає на вхід рядок символів і повертає деякий рядок як результат. В автомата є певний набір інструкцій — список правил заміни. Кожне правило складається з підрядка, який потрібно замінити, і рядка, на який цей підрядок потрібно замінити. Автомат послідовно робить такі операції: шукає першу в списку заміну таку, що поточний рядок містить відповідний підрядок, міняє перше входження цього підрядка на відповідний рядок-заміну і повторює все спочатку. Коли жоден підрядок у поточному рядку не міститься, автомат повертає цей рядок як результат. Список замін автомата називається *програмою*. Кількість правил замін називається *розміром програми*. Кількість зроблених замін на конкретних вхідних даних називається *кількістю операцій*.

Розглянемо приклад, коли на вхід автомат приймає рядок `abcbca`, а його програма розміру 2 має такий вигляд: 1) `bac` → `c` і 2) `bc` → `cba`. Порядок дій автомата такий:

1. Поточним рядком є `abcbca`.
2. Оскільки `abcbca` не містить підрядка `bac` з першого правила, автомат переходить до другого правила.
3. Оскільки `abcbca` містить підрядок `bc` з другого правила, автомат заміняє перше входження цього підрядка на `cba` і повертається на початок.
4. Поточним рядком є `acbabca`.
5. Оскільки `acbabca` не містить підрядка `bac` з першого правила, автомат переходить до другого правила.
6. Оскільки `acbabca` містить підрядок `bc` з другого правила, автомат заміняє перше (і єдине) входження цього підрядка на `cba` і повертається на початок.
7. Поточним рядком є `acbacbaa`.
8. Оскільки `acbacbaa` містить підрядок `bac` з першого правила, автомат заміняє перше (і єдине) входження цього підрядка на `c` і повертається на початок.
9. Поточним рядком є `accbaa`.
10. Оскільки `accbaa` не містить підрядка `bac` з першого правила, автомат переходить до другого правила.
11. Оскільки `accbaa` не містить підрядка `bc` з другого правила, автомат намагається перейти до наступного правила.
12. Оскільки правила закінчилися, автомат повертає рядок `accbaa` як результат.

Таким чином, з рядка `abcbca` автомат утворив `accbaa`. Кількість операцій у даному випадку дорівнює 3: замінили `bc` на `cba`, потім ще раз `bc` на `cba`, далі `bac` на `c`.

Завдання. Напишіть набір програм для рядкового автомата, що вирішують різноманітні задачі перетворення рядків (див. нижче). Для кожної підзадачі ви здаєте лише текст програми рядкового автомата. За кожну підзадачу ви отримуєте або повний бал за відповідну підзадачу, якщо програма пройшла всі її тести, або нуль, якщо хоча б один тест даної підзадачі не пройдено.

Формат програми. У кожному рядку файла міститься відповідне правило заміни: спочатку підрядок, який треба знайти, потім один пробіл, потім рядок, на який потрібно замінити підрядок. Якщо рядок заміни порожній, пробіл необов'язковий (інакше кажучи, у відповідному правилі можна вказати лише рядок, який потрібно видалити, і після нього пробіл не ставити). Між будь-якими двома сусідніми правилами для читабельності за бажанням можна вставити один порожній рядок. Після останнього правила можна як ставити, так і не ставити перенесення рядка.

Обмеження. У кожному правилі довжина як підрядка пошуку, так і рядка заміни не повинна перевищувати 10 символів (але в сумі вони можуть бути довшими); при цьому рядок заміни може бути порожнім, а підрядок пошуку — ні. Автомат завжди може оперувати такими символами: цифри, малі та великі латинські літери (причому малі та великі літери в розумінні автомата — різні символи), а також символ «плюс» (+). Ви можете користуватися всіма цими символами як у підрядках пошуку, так і в рядках заміни в усіх підзадачах. Розмір однієї програми не може перевищувати 100. Для жодних вхідних даних кількість операцій не повинна перевищувати 1000. Означення розміру програми та кількості операцій див. вище.

Перевірка за допомогою емулятора. Ви можете завантажити та скомпілювати спеціальну утиліту[1], що емулює рядковий автомат та надає допоміжну інформацію про перебіг виконання вашої програми. Щоб використати емулятор, розташуйте його в окремому каталозі, створіть у цьому каталозі файл `program.txt` з текстом програми, а також файл `input.txt`, у який введіть вхідний рядок для вашої програми (можна додати або не додавати перенесення рядка в кінці файла), та запустіть програму-емулятор. Вона створить такі файли:

`!output.txt`: вихідний рядок автомата (якщо програма чи вхідні дані некоректні або кількість операцій перевищує обмеження, файл не буде створено / буде видалено).

`!debug.txt`: у кожному рядку цього файла вказано стан рядка після відповідної кількості операцій (якщо програма чи вхідні дані некоректні, цей файл не буде створено / буде видалено; якщо кількість операцій перевищує обмеження, файл міститиме стан рядка тільки до моменту, коли було перевищено обмеження).

`!info.txt`: у першому рядку цього файла вказано, чи є коректними програма та вхідні дані (а якщо вони некоректні, то чому); якщо програма та вхідні дані коректні, у другому рядку вказано кількість операцій виконаної програми; у третьому рядку вказано розмір програми; у четвертому рядку вказано довжину найдовшого рядка пошуку та довжину найдовшого рядка заміни.

Додатково емулятору можна передавати до чотирьох параметрів командного рядка: обмеження на кількість операцій, на розмір програми, на довжину рядка пошуку та на довжину рядка заміни. Наприклад, якщо запустити емулятор командою `auto 2000 100 15`, обмеженням на кількість операцій буде встановлено 2000 (замість 1000); обмеження на розмір програми буде стандартним (100); обмеження на довжину рядка пошуку — 15; невказане в параметрах командного рядка обмеження на довжину рядка заміни також залишиться стандартним (10).

[1] На самій олімпіаді учасникам надавали вже скомпільований варіант.

Підзадача 1 (5 балів). На вході — рядок з цифр довжини від 1 до 20 включно. На виході — рядок із тих самих цифр у порядку від найменших до найбільших. Приклад: `9101` → `0119`.

Підзадача 2 (15 балів). На вході — рядок з N символів `I` (велика латинська літера «і»), $1 \leqslant N < 1000$. На виході — десятковий запис числа N. Приклад: `IIIIIIIIIII` → `12`. Відомо, що число N не містить у своєму десятковому записі нулів.

Підзадача 3 (10 балів). На вході — рядок з N символів `I` (велика латинська літера «і»), $1 \leqslant N \leqslant 1000$. На виході — десятковий запис числа N. Приклад: `IIIIIIIII` → `10`.

Підзадача 4 (20 балів). На вході — десятковий запис числа N, $1 \leqslant N \leqslant 1000$. На виході — рядок з N символів `I` (велика латинська літера «і»). Приклад: `10` → `IIIIIIIII`.

Підзадача 5 (20 балів). На вході — рядок з малих літер a, b та c довжини від 1 до 20 включно. На виході — рядок з тих же літер, але великих і записаних у зворотному порядку. Приклад: `abac` → CABA.

Підзадача 6 (15 балів). На вході — десятковий запис двох натуральних чисел, що не перевищують 100; числа розділено символом +. На виході — десятковий запис суми відповідних чисел. Приклад: `4+19` → `23`.

Підзадача 7 (15 балів). На вході — десятковий запис двох натуральних чисел, що не перевищують 1000; числа розділено символом +. На виході — десятковий запис суми відповідних чисел. Приклад: `204+119` → `323`.

Розв'язання

Зауважимо, що, як і в більшості інших задач, подані тут ідеї не є єдино правильним підходом до відповідних підзадач. Усі або майже всі підзадачі мають багато різних розв'язків.

Підзадача 1. Слід узяти всі пари сусідніх цифр, що йдуть у порядку зменшення, та поміняти їх порядок на протилежний: 10 міняємо на 01, 20 на 02, ..., 98 — на 89. Порядок, у якому записано правила, ролі не грає.

Підзадача 2. Заміняємо блоки по 10 літер I на деяку іншу літеру: скажімо, J. Після таких замін у кінці рядка залишиться така кількість літер I, що дорівнює останній цифрі шуканого числа. Відповідно, заміняємо дев'ять літер I на дев'ятку, вісім — на вісімку, ..., одну — на одиницю. Отримали правильну останню цифру, зліва від якої кількість літер J дорівнює кількості повних десятків у шуканому числі. Тоді можна з літерами J провести ту саму операцію: замінити блоки з 10 таких літер, скажімо, на K та «згорнути» ті літери J, що залишилися, у цифру. Ця цифра і є розрядом десятків шуканого числа. Залишається «згорнути» сотні: їх (тобто літер K) може залишитися не більше за 9.

Підзадача 3. Дещо модифікуємо попередній алгоритм: замінятимемо блоки по 10 літер I на J0, після чого відразу 0J та 0I — на J та I відповідно. Таким чином дістанемо лише один нуль у кінці рядка, причому тоді й лише тоді, коли всі літери I розбилися на групи по 10 без залишку. Крім того, як і раніше, дев'ять I замінимо на дев'ятку, вісім I на вісімку і т. д. Далі десять J замінимо на K0, проведемо аналогічні операції з J і K. Нарешті, десять K замінимо на десятку, дев'ять K на дев'ятку, ..., одну K — на одиницю.

Підзадача 4. Замінимо цифру 0 на A, 1 — на AI, 2 — на AII, ..., 9 — на A і дев'ять літер I. Тепер якщо після деякої літери I стоїть у сумі n літер A, то дана літера позначає розряд 10^n. Тому, замінюючи послідовно фрагмент IA на A і десять літер I, у підсумку отримаємо якраз таку кількість літер I, яка дорівнює початковому числу[2]. При цьому зліва від них стоятимуть усі наявні літери A. Залишається замінити A на порожній рядок. Єдине: замінити IA на A і десять I нам не дозволяють обмеження на розмір рядка заміни. Але з цієї ситуації нескладно вийти, замінивши натомість IA на AB, а першим правилом у програмі — B на десять I.

Підзадача 5. Спочатку встановимо індикатор початку рядка: замінимо всі a на 0a, b на 0b, c на 0c, а всі a0 на a, b0 на b, а c0 — на c. Після таких операцій у нас залишиться єдиний нуль на першій позиції[3]. Тепер будемо робити великими літери, що опинилися на початку рядка, тобто замінювати 0a на 0A, 0b на 0B, 0c на 0C. Водночас будемо переставляти всі 9 пар з великих та малих літер, коли велика стоїть перед малою. Так ми спочатку пересунемо першу літеру в кінець, потім друга літера стане на передостаннє місце і т. д. Насамкінець лишається замінити 0 на порожній рядок. Єдина проблема тут така, що при початковій зміні, наприклад, символу a на 0a послідовність операцій буде зациклюватися. Тому мінятимемо a на 0x, b на 0y, c на 0z, після чого ототожнюватимемо x із a, y із b, а z — із c.

[2] Але див. зауваження нижче.
[3] Але див. зауваження нижче.

Підзадачі 6 і 7. Ідея базується на розкладі обох доданків у відповідну кількість літер I, після чого літери (яких тепер якраз сумарна кількість) збираються назад у число. Основна проблема полягає в тому, що після того, як ми зберемо остаточне число, рядковий автомат знов захоче розкладати числа в літери I. Щоб цього не сталося, ми маємо використати єдину відмінність між вхідними та потенційними вихідними даними: наявність плюса. Локальність цього плюса можна побороти таким чином: замінюватимемо, скажімо, не 4 на AIIII, а 4+ на pAIIII+ та аналогічно +4 на +AIIIIp. Тоді за допомогою символу p ми зможемо пропагувати зміни (а точніше, уникати зайвих змін) також і значно лівіше та правіше від плюса. Після цього плюс та символ p прибираються та йде зворотний процес. Проблеми з розміром рядка заміни вирішуємо так само, як і в четвертому пункті. Цілком імовірно, що кількість виконуваних операцій при цьому вийде за допустиму межу. Тоді можна оптимізувати програму: додатково до заміни IA на AB (див. четверту підзадачу) введемо заміни IIA на ABB, IIIA на ABBB і т. д. Щоб перша частина алгоритму (зведення до одиниць) не конфліктувала з другою (зведення до десяткового запису), можемо в першій частині за одиниці брати літери I, далі міняти їх на J, а в другій частині одиницями вважати вже літери J, а I ніяк не зачіпати. Щоб оптимізувати кількість операцій, потрібно ввести не лише заміну I на J, але й, наприклад, II на JJ, III на JJJ і т. д. Насамкінець зауважимо, що в третьому пункті ми працювали лише з числами в межах до 1000, а в даній підзадачі сума може вийти більшою. Отже, потрібно доповнити відповідні правила для коректної роботи з числами до 2000.