Розділ 1. Дніпропетровськ'2006

1.1 Завдання першого туру

1.1.1 «Дільники»

Розв'язок 1 Найпростіший розв'язок задачі — генерація факторіалу $N!$ та перевірка чи ділять його націло числа від 1 до $N!$. Недоліками такого розв'язку є те, що $N!$ при $N = 45$ набуває великих значень, та такий алгоритм працює занадто довго.

Розв'язок 2 Кращий розв'язок можна побачити, якщо розглянути структуру простих дільників факторіалу $N! = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots p_t^{k_t}$. Тут $p_1 \ldots p_t$ — прості дільники $N!$, $k_1 \ldots k_t$ — кратності цих дільників. Якщо відомі кратності дільників, то відповідь задачі може бути отримана за такою комбінаторною формулою:

$$x = (1 + k_1) \cdot (1 + k_2) \cdot \ldots \cdot (1 + k_t)$$

Отримати ці дільники можна порахувавши кількість простих дільників всіх множників, що складають факторіал від 2 до N.

У найгіршому випадку складність такого розв'язку складе $O(N^2)$, але можна реалізувати достатньо ефективні алгоритми реалізації пошуку кількості простих дільників.

Розв'язок 3 Як завжди у задачах з невеликою кількістю різних вхідних даних, відповіді можна обчислити та внести в масив констант.

1.1.2 «Екзамен»

Задача базується на класичній задачі про найбільше паросполучення у двудольному графі, з тією лише різницею, що потрібно побудувати лексикографічно найменше паросполучення. Розглянемо задачу у саме такому, графовому, формулюванні.

Нехай деяким чином отримано найбільше паросполучення. Знайдемо вершину з найменшим з номерів, що не увійшли до нього. Побудуємо ланцюг, який починається з цієї вершини, та йде почергово по ненасичених та насичених ребрах, та закінчується у вершині, що має більший номер ніж початкова. Тоді, після інвертації цього шляху, буде отримано паросполучення, що є лексикографічно меншим ніж до перетворення. Операцію потрібно виконувати до тих пір, доки такі ланцюги існуватимуть.

Доведемо, що у паросполучення, в якому немає таких ланцюгів є лексикографічно найменшим. Припустимо, що це не так. Нехай, існує паросполучення найбільшої потужності A, та паросполучення такої ж потужності B, яке лексикографічно менше за A. При цьому, вершина з номером v_0, це найменша вершина яка насичена у B, та не насичена в A. Всупереч твердженню побудуємо ланцюг в A таким чином. Почнемо з ребра, яке відповідає насиченому ребру що інцедентне v_0 у графі B. Ми перейдемо до вершини v_1, яка має інцедентне насичене ребро до v_2 (інакше це був би подовжуючий ланцюг, що суперечить умові), яке буде наступним у ланцюзі. Якщо $v_2 > v_0$, то ланцюг побудований. Інакше, інснує насичене ребро, що інцедентне v_2, та потрібно йти за ним. Цей процес зупиниться, тому що до жодної вершини ми не можемо потрапити два рази.

1.2 Завдання другого туру

1.2.1 «Пошук строки»

Розв'язок 1 Найпростіший розв'язок задачі — зберігання у буфері підрідку, у якому може бути знайдена оптимальна відповідь, та при додаванні нової літери зміщати буфер, та шукати у ньому рядок найбільшої довжини, який не містить однакових символів. Ця операція лінійна по кількості символів у файлі, але константа занадто велика.

Розв'язок 2 Кращий розв'язок задачі в процесі обробки вхідної послідовності запом'ятовує для кожного з 26 символів номер у послідовності коли він зустрівся останній раз l_c, та два лічильника: поточний номер символу у послідовності i, та поточний розмір підпослідовності без повторень m. На кожному кроці алгоритм виконує наступні дії:

- Зчитує наступний символ c з файлу, додає його до буферу, та додає одиницю до i.
- Перевіряє коли в останній раз зустрівся символ c, та якщо $i - l_c > w$ збільшує w на одиницю. Інакше $w = i - l_c$.
- Якщо поточна довжина більша за найкращу знайдену до цього, то поточна довжина, то поточна підпослідовність, яка відновляється з буферу, зберігаються.

Цей розв'язок також має лінійний порядок часу, але його константа відчутно менша за попередню.

В обох випадках можна також використовувати спеціальні засоби мов програмування для збільшення швидкості роботи з файлами.

1.2.2 «Щасливий білет»

Задача розв'язується за допомогою пошука з поверненням (backtracking). Для зручності, вираз генерується за допомогою польського інверсного запису (ПОЛІЗ, reverse polish notation (RPN)), який нагадує стек.

На кожному кроці рекурсія розгалуджується на такі варіанти:

- У стек записується число яке складається з ще не оброблених 1, 2,... цифр.
- Якщо в стекі є хоча б два числа, до стека додаються операціїї $+, -, \times$ та / (у випадку якщо ділення ціле), та операція відразу ж виконується.
- Умовою виходу з рекурсії є ситуація, коли всі цифри з номеру білета вже оброблено, та в стекі залишилося лише одне число. Коли ця умова виконана обчислений вираз порівнюється з числом k, та $-k$ (це єдине що потрібно зробити для врахування унарного мінуса).

Хоча алгоритм і перебірний, але за умови маленького розміру вхідних даних його виконання практично миттєве.