All-Ukrainian Collegiate Programming Contest 2021, II stage Editorial

Задача A. Adjacent Button Characters

Розбір підготувала: **Софія Мельник**

Спочатку варто для кожної букви знайти номер її рядку на клавіатурі. Це можна записати в окремий масив або меп.

Далі для кожного запиту потрібно порівняти чи номери рядків двох букв рівні, тоді вивести "`Yes`", інакше "`No`".

Задача B. Bad Checker Detected

Розбір підготувала: **Софія Мельник**

Для розв'язання задачі варто перебрати усі $L \leqslant X \leqslant R$ і перевірити для цього X чи відстань до найближчого щасливого числа ліворуч рівна відстані до найближчого щасливого числа праворуч.

При цьому, для швидкого знаходження щасливих чисел ліворуч та праворуч, будемо підтримувати ці два числа в змінних $left$ та $right$. При збільшенні X на 1 будемо змінювати $left$ та $right$. Якщо число X – щасливе, тоді в $left$ потрібно записати X, оскільки на цей момент це буде найближче щасливе число ліворуч. Також якщо X – щасливе, то при переході до $X + 1$ потрібно знайти нове найближче щасливе число праворуч. Для цього можна здвигати $right$ поки воно не стане щасливим. Оскільки $right$ буде тільки збільшуватись, отримаємо лінійне рішення.

Для того, аби швидко перевірити чи число є щасливим, можна використати математичні формули для знаходження кожної цифри в числі, а потім взяти суму перших трьох та останніх трьох і зрівняти їх.

Задача C. Conjuring Dark Energy

Розбір підготувала: **Софія Мельник**

Спочатку відсортуємо усі точки в порядку зростання координати x. Тоді якщо друга точка лежить нижче першої, відобразимо другу і третю точку відносно прямої y_1. Тепер маємо, що друга точка лежить вище і праворуч від першої. Залишилось розглянути три випадки розміщення третьої точки (вище другої, між першою і другою, нижче першою), щоб правильно порахувати відповідь.

Задача D. Digital Evolution Foundation

Розбір підготувала: **Вікторія Ковальова**

Розглянемо функції $f(k)$ і $g(k)$: $f(k)$ — кількість чисел з k цифр, перша цифра яких не нуль, які мають принаймні один нуль і два нулі не йдуть послідовно, $g(k)$ — кількість чисел з k цифр, які не мають нулів. $g(k) = 9^k$

Виразимо значення f рекурсивно: Для чисел, що рахує $f(k)$, є 9 варіантів вибору першої цифри. Якщо друга цифра не нуль, то існує $f(k-1)$ способів вибрати останні $k-1$ цифр. Інакше, друга цифра — нуль і існує $f(k-2)$ варіантів вибору останніх $k-2$ цифр.

Отже, $f(k) = 9 \cdot (f(k-1) + f(k-2)),$ при $2 \leqslant k$.

Знайдемо значення $f(k)$ та $g(k)$ для $k \leqslant 14$. Це можна зробити динамічно за $O(14)$.

Тепер будемо по черзі знаходити цифри n-го ідентіфікаційного номеру. Нехай ми знаходимо i-ту цифру. Спочатку візьмемо 0 (або 1, якщо це перша цифра чи попередня була 0) та будемо збільшувати її поки можливо. При збільшенні цифри з x до $x+1$ ми пропускаємо ідентіфикаційні номери, що мали x на i-й позиції. Кількість таких номерів:

При $x = 0$: $f(15-i)$ (якщо в суфіксі є нулі) $+ g(15-i)$ (якщо нема).

При $x \neq 0$:

номерів з $i+1$-ою цифрою не нуль, але з нулями в суфіксі — $f(15-i)$,

номерів з $i+1$-ою цифрою нуль — $f(14-i) + g(14-i)$ (їх потрібно враховувати, якщо i - не остання),

номерів без нулів в суфіксі — $g(15-i)$ (їх потрібно враховувати, якщо серед попередніх цифр були нулі).

Позначимо кількість вже пропущенних ідентіфикаційні номерів як m. Будемо збільшувати дану цифру, пока вона менша 9 та поки збільшення не призведе до $n \leqslant m$. Після того, як ми знайшли всі цифри числа, виведемо його, якщо $m + 1 = n$, та -1 в іншому випадку.

Ми перебираємо не більше 10 цифр для 15 позицій, тобто асимптотика $O(10 \cdot 15) = O(C)$.

Задача E. Experiments For Generalization

Розбір підготував: **Петро Тарнавський**

Якщо X непарне, тоді одне з двох простих чисел рівне 2, а інше $X - 2$. Отже, непарне число можна подати як суму двох простих чисел тоді й лише тоді коли $X-2$ — непарне просте число. Отже, для того, щоб порахувати кількість непарних чисел X на проміжку [A, B], нам потрібно порахувати кількість непарних простих чисел на проміжку [A - 2, B - 2], а це можна зробити блоковим решетом Ератосфена.

Для випадку парних X (проблема Гольдбаха) у 2012 році було перевірено, що кожне парне натуральне число більше за 2 і менше за $4 \cdot 10^{18}$ можна подати, як суму двох простих чисел. Отже, нам підходять всі парні числа крім 2.

Задача F. Funny Gnomish Hockey

Розбір підготував: **Ігор Баренблат**

Позначимо через p кількість шайб на полі, а через n та m розміри поля.

Помітимо, що при $min(n, m) > 1$ та $p \geqslant 3$ відповідь "`yes`". Доведення цього предстaлено нижче (див. Лемму 4). Для випадків $min(n, m) = 1$ та/або $p \leqslant 2$ можна скористатись пошуком по всім можливим станам поля з складністю $O(C^p_{n \cdot m} \cdot p \cdot (n + m))$. Також розв'язати ці випадки можна аналітично зі складністю $O(n \cdot m)$.

Лемма 1. При фіксованих n, m, p та фіксованому розміщенні елементу «g» таких, що $min(n, m) = 2$ та $p = 2$, існує універсальна стратегія досягнення позитивного результату яка не залежить від початкового розміщення шайб на полі.

Не обмежуючи загальності вважатимемо що $n = 2$.

Побудуємо стратегію досягнення позитивного результату незалежно від розміщення елементу «g»:

- Штовхаємо шайбу1 вверх, шайбу2 вниз, шайбу1 вліво, шайбу2 вліво. Тепер обидві шайби знаходяться у першому стовпчику. Вважатимемо що шайба1 знаходиться у лівому верхньому кутку поля, шайба2 у лівому нижньому кутку поля.

- Виконаємо $(m-1)$ разів наступну комбінацію кроків: штовхаємо шайбу2 вправо, шайбу2 вверх, шайбу2 вліво, шайбу2 вниз, шайбу2 вверх, шайбу1 вниз, змінюємо номери шайб (шайба1 стає шайбою2, а шайба2 стає шайбою1).

Лемма 2. При фіксованих n, m, p та фіксованому розміщенні елементу «g» таких, що $min(n, m) \geqslant 3$ та $p = 3$, існує універсальна стратегія досягнення позитивного результату яка не залежить від початкового розміщення шайб на полі.

А) Елемент «g» розміщено на границі поля. Не обмежуючи загальності вважатимемо що елемент «g» розміщено у першому рядку поля. Побудуємо стратегію досягнення позитивного результату:

- Тричі виконаємо наступну комбінацію кроків: штовхаємо шайбу1 вниз, шайбу2 вниз, шайбу3 вниз. Тричі виконаємо наступну комбінацію кроків: обираємо найвищу, а серед таких найлівішу, шайбу серед тих, що ще не були обрані за цим принципом, і штовхаємо її спочатку вліво а потім вверх. Тепер всі три шайби знаходяться у перших трьох клітинках першого стовпчика поля. Вважатимемо що шайба1 знаходиться у лівому верхньому кутку поля, шайба2 у другій клітинці першого стовпчика поля, шайба3 у третій клітинці першого стовпчика поля.

- Виконаємо $(m-1)$ разів наступну комбінацію кроків: штовхаємо шайбу2 вправо, шайбу2 вверх, шайбу2 вліво, шайбу1 вниз, змінюємо номери шайб з номерами 1 та 2 (шайба1 стає шайбою2, а шайба2 стає шайбою1).

Б) Елемент «g» розміщено не на границі поля. Побудуємо стратегію досягнення позитивного результату:

- Тричі виконаємо наступну комбінацію кроків: штовхаємо шайбу1 вниз, шайбу2 вниз, шайбу3 вниз. Тричі виконаємо наступну комбінацію кроків: обираємо найвищу, а серед таких найлівішу, шайбу серед тих, що ще не були обрані за цим принципом, і штовхаємо її спочатку вліво а потім вверх. Тепер всі три шайби знаходяться у перших трьох клітинках першого стовпчика поля. Вважатимемо що шайба1 знаходиться у лівому верхньому кутку поля, шайба2 у другій клітинці першого стовпчика поля, шайба3 у третій клітинці першого стовпчика поля.
- Штовхаємо шайбу3 вниз, шайбу3 вправо, шайбу3 вверх, шайбу3 вліво, шайбу3 вниз.
- Допоки шайба3 знаходиться з елементом «g» у різних рядках виконуємо наступну комбінацію кроків: штовхаємо шайбу2 вправо, шайбу2 вверх, шайбу2 вліво, шайбу2 вниз, змінюємо номери шайб з номерами 2 та 3 (шайба2 стає шайбою3, а шайба3 стає шайбою2).
- Штовхаємо шайбу1 вправо, шайбу2 вправо, шайбу2 вверх.
- Виконаємо $(m-3)$ разів наступну комбінацію кроків: допоки шайба2 знаходиться у різних рядках з шайбою3 штовхаємо шайбу1 вліво, шайбу1 вниз, шайбу1 вправо, шайбу1 вверх, змінюємо номери шайб з номерами 1 та 2 (шайба1 стає шайбою2, а шайба2 стає шайбою1); коли шайба2 опинилась у одному рядку з шайбою3 штовхаємо шайбу1 вверх, шайбу2 вліво, шайбу3 вниз, шайбу3 вправо, шайбу3 вверх, змінюємо номери шайб з номерами 2 та 3 (шайба2 стає шайбою3, а шайба3 стає шайбою2).

Лемма 3. При існуванні універсальної стратегії S досягнення позитивного результату на полі фіксованого розміру з фіксованим розміщенням елементу «g» яке містить p шайб, завжди існує універсальна стратегія S' на полі такого ж розміру з таким самим розміщенням елементу «g» що містить p' шайб, де $p \leqslant p'$.

Доведемо для деякого фіксованого p' такого, що $p \leqslant p'$. Для початку оберемо будь-який набір що містить p шайб — позначимо ці шайби «видимими», відповідно всі інші $(p'-p)$ шайб позначимо «невидимими». Будемо виконувати кроки стратегії S один за одним, вважаючи що поточний набір видимих шайб є набором шайб для виконання плану стратегії — таким чином ми побудуємо стратегію S'. На кожному кроці стратегії ми обиратимемо деяку видиму шайбу та штовхатимемо її у певному напрямку. Якщо ця шайба вдаряється у стіну або у іншу видиму шайбу то очевидно інваріант стратегії S' валідний. Якщо шайба вдаряється у деяку невидиму шайбу то «змінимо їх видимості» — та видима шайба що зупиняється при зіткненні стає невидимою, а шайба що починає рухатись після зіткнення стає видимою. Очевидно що в цьому випадку інваріант стратегії S' теж валідний.

Лемма 4. При фіксованих n, m, p та фіксованому розміщенні елементу «g» таких, що $min(n, m) > 1$ та $p \geqslant 3$ існує універсальна стратегія досягнення позитивного результату яка не залежить від початкового розміщення шайб на полі.

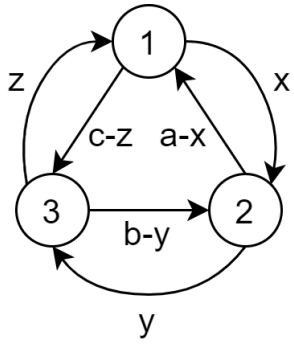
З Лемм 1, 2, 3 очевидно.

Задача G. General Highway Inspection

Розбір підготував: **Матвій Асландуков**

Для зручності позначимо міста Doublein, Longdon та Longlongdon як перше, друге та третє місто відповідно. Також будемо вважати, що існує рівно a доріг між першим та другим містом, b доріг між другим та третім, та c доріг між третім та першим (для цього у вхідних даних треба поміняти числа b та c місцями).

Розглянемо будь-який шлях, який починається та завершується в першому місті, та проходить через кожну з $(a+b+c)$ доріг рівно один раз. Нехай у цьому шляху було рівно x переходів від першого міста до другого, y — від другого до третього, та z — від третього до першого. Оскільки шлях проходив через кожну дорогу рівно один раз, то на ньому також було рівно $(a-x)$ переходів від другого міста до першого, $(b-y)$ — від третього до другого, та $(c-z)$ — від першого до третього (див. рис. 1).



Оскільки наш шлях є циклом, то для будь-якого міста сумарна кількість виходів з нього дорівнює сумарній кількості входів до нього. А тому мають місце наступні рівності:

1. $x + (c - z) = (a - x) + z$;
2. $y + (a - x) = (b - y) + x$;
3. $z + (b - y) = (c - z) + y$,

З першого та другого рівнянь отримуємо $z = \frac{2x+c-a}{2}$, $y = \frac{2x+b-a}{2}$. Таким чином, знаючи лише тільки x, можна однозначно визначити кількість проходів по кожному типу доріг в обидві сторони.

Подивимось, як можна знайти кількість шляхів для заданих x, y, z. Будемо вважати, що усі дороги одного типу еквівалентні між собою (в кінці потрібно буде помножити відповідь на $a!\,b!\,c!$, оскільки всі дороги все ж таки різні). Розглянемо усі моменти часу, коли ми знаходилися в першому місті, та повинні були перейти до наступного міста. Усього таких моментів було $x+(c-z)$, серед яких рівно x разів ми перейшли до другого міста. Тому кількість способів зафіксувати порядок переходів з першого міста до наступного — $C^x_{x+c-z} = \frac{(x+c-z)!}{x!(c-z)!}$. Аналогічно кількість способів зафіксувати порядок переходів з другого та третього міст дорівнює C^y_{y+a-x} та C^z_{z+b-y} відповідно. Загальну кількість шляхів для заданих x, y, z можна підрахувати як $w = f(a, b, c, x, y, z) = C^x_{x+c-z} \cdot C^y_{y+a-x} \cdot C^z_{z+b-y}$. Кожен з w способів буде відповідати певній послідовності переходів, що починається та закінчується в першому місті. Але деякі з них не будуть відвідувати усі дороги, а саме певну кількість доріг між другим та третім містом. Це можливо у випадку, коли є хоча б один перехід від другого міста до третього, та від третього міста до другого, тобто $1 \leqslant y, b - y$. У такому випадку від w необхідно відняти $f(a, b - 2, c, x, y - 1, z)$.

Таким чином, остаточна відповідь на задачу дорівнює $a!\,b!\,c!\sum\limits_{x=0}^{a}(f(a, b, c, x, y, z) - f(a, b-2, c, x, y-1, z))$, де $y = \frac{2x+b-a}{2}$, $z = \frac{2x+c-a}{2}$, а $f(a, b, c, x, y, z)$ вважається рівним нулю у випадку, коли хоча б одне з чисел y, z не є цілим, або ж не виконується обмеження $0 \leqslant y \leqslant b$, $0 \leqslant z \leqslant c$.

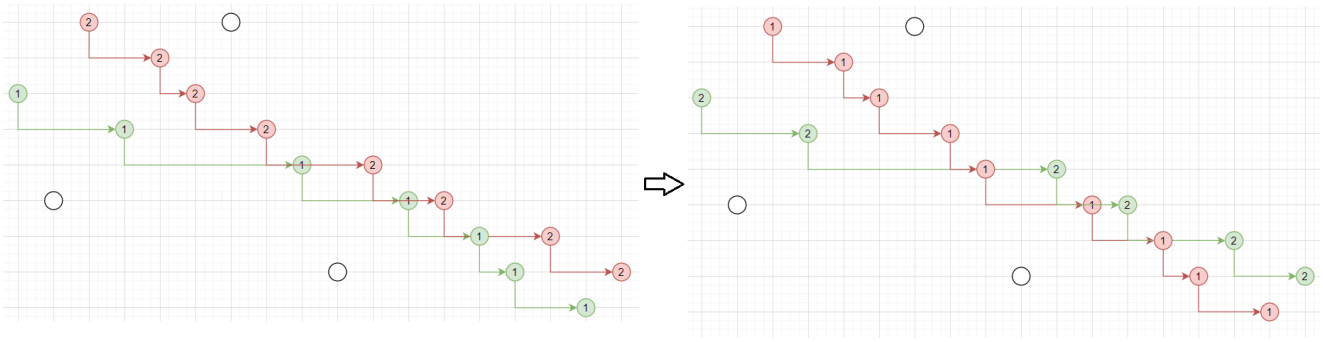
Для швидкого підрахунку функції f необхідно вміти швидко знаходити значення біноміальних коефіцієнтів C^k_n. Це можна зробити за допомогою попереднього підрахунку усіх значень факторіалів від 0 до $2n$, та відповідних зворотніх факторіалів. Такий підрахунок може бути виконаний за час $O(n + \log mod)$ або ж $O(n \log mod)$, після чого значення C^k_n можна знаходити за $O(1)$.

Загальна складність — $O(n + \log mod)$ чи $O(n \log mod)$ в залежності від реалізації підрахунку зворотніх факторіалів.

Задача H. Hunter In Jury

Розбір підготував: Олег Валлас

Спочатку покажемо, що порядок, в якому ми випускаємо стріли, не впливає на результат. Для цього розглянемо політ 2х стріл. Якщо їх траєкторії не перетинаються, тоді очевидно, що їх можна випускати в будь-якому порядку. Якщо ж вони перетинаються, то зміна порядку просто поміняє їх місцями в точці перетину, в той час як множина збитих кульок не зміниться. Дивіться малюнок для кращого розуміння.



Використовуючи цю властивість можна запускати всі стріли паралельно. Будемо обробляти кульки зліва направо і підтримувати масив cnt на мільйон елементів, де $cnt[h]$ означатиме кількість стріл, які летять на висоті h. Спочатку заповнимо масив нулями. Коли ми зустрічаємо чергову кульку на висоті h, перевіримо чи є у нас якась стріла на висоті h. Якщо немає, то необхідно додати нову і збільшити відповідь (`cnt[h]++; answer++;`). Після цього кульку можна збити будь-якою стрілою на висоті h (пам'ятаємо, що порядок стріл не важливий), а саме — зробити «`cnt[h]--; cnt[h - 1]++;`». В кінці просто виведемо змінну *answer*.

Складність $O(N+H)$ часу, та $O(H)$ пам'яті, де N — кількість кульок, а H — максимальна висота кульки, що в рамках задачі дорівнює 10^6.

Задача I. Input Jumbled Key

Розбір підготував: Олег Валлас

Назвемо паролі, які містять рівно одну парну цифру, *парними*, а ті, що складаються повністю з непарних цифр — *непарними*. Назвемо непарний пароль *сусіднім* до парного, якщо непарний може відкрити замок замість парного (за умовою завдання), тобто відрізняється рівно в одній цифрі на одиницю.

За умовою необхідно щоб після введення списку паролів замок обов'язково відкрився. Оскільки ми не знаємо, який саме пароль правильний, необхідно підібрати список паролів мінімального розміру (а серед них лексикографічно мінімальний) так, щоб для кожного пароля з вводу в нашому списку був присутній або він сам, або сусідній до нього непарний.

Спочатку покажемо, як знайти розмір відповіді (поки забудемо що вона повинна бути лексикографічно мінімальною).

Твердження 1: коректна множина паролів мінімального розміру завжди може бути складена тільки з непарних паролів, сусідніх із вхідними.
Дійсно:

- парний пароль може відкрити тільки себе, тому його завжди можна замінити на сусідній непарний і кількість "покритих"паролів не зменшиться.
- паролі, які не є сусідніми з вхідними, очевидно брати не вигідно взагалі, тому що вони нічого не можуть відкрити.

Тоді побудуємо граф, в якому непарні паролі, сусідні з вхідними, будуть вершинами, а вхідні паролі — ребрами (у кожного парного пароля якраз рівно 2 сусідніх).

Твердження 2: отриманий граф є дводольним.

Коли ми переходимо по будь-якому ребру цього графа, в паролі рівно одна цифра збільшується або зменшується на 2. Не складно помітити, що для того щоб прийти з вершини в себе нам завжди доведеться зробити парну кількість таких операцій. Це означає, що всі цикли в графі парної довжини, значить він дводольний за визначенням.

Отже, ми хочемо вибрати мінімальну кількість непарних (виходячи з твердження 1) паролів, щоб покрити всі парні. Це ні що інше, як мінімальне вершинне покриття в отриманому графі.

Оскільки граф дводольний, розмір його мінімального вершинного покриття дорівнює розміру його максимального парування. Його можна знайти, наприклад, алгоритмом Куна.

Тепер ми знаємо розмір відповіді, покажемо як знайти сам лексикографічно мінімальний набір паролів цього розміру.

Зауважимо, що в лексикографічно мінімальному наборі можуть бути як непарні, так і парні паролі (наприклад, 2й тест з умови). У твердженні 1 говорилося лише про мінімальний розмір відповіді, але така відповідь не обов'язково буде лексикографічно мінімальною.

Саму відповідь будемо відновлювати жадібно.

Переберемо все $3N$ потенційних кандидатів в порядку зростання. Видалимо з графа відповідну поточному паролю вершину або ребро. Порахуємо в отриманому графі розмір максимального парування. Якщо він залишився таким самим, то поточний пароль взяти не вийде. Якщо ж розмір зменшився на 1, то це означає, що ми зможемо покрити решту паролів набором розміру на один менше. Тоді візьмемо поточний пароль у відповідь.

Якщо після кожного видалення шукати максимальне парування заново, часова складність буде $O(3N \times 2N \times (3N + 2N)) = O(N^3)$, тому що в графі $O(3N)$ вершин і $O(2N)$ ребер. Це занадто повільно, тому будемо перераховувати цей розмір більш оптимально. Спочатку знайдемо будь-яке парування. Тепер якщо ми розглядаємо пароль, який відповідає:

- ребру не з парування: просто пропустимо цей пароль. Очевидно, що після видалення такого ребра розмір парування не зміниться;
- вершині, що не є інцидентною жодному ребру з парування: просто пропустимо цей пароль з аналогічних міркувань;
- ребру з парування: видалимо ребро і запустимо пошук шляху розширення з обох кінців ребра;
- вершині, що є інцидентною ребру з парування: видалимо вершину і запустимо пошук шляху розширення з протилежного кінця цього ребра.

Оскільки пошук шляху розширення працює за $O(V + E)$, підсумкова складність рішення буде $O((3N + 2N) \times (3N + 2N)) = O(N^2)$.

Задача J. Jungle Kingdom Laws

Розбір підготував: **Денис Смірнов**

Позначимо за $S(i, j)$ площу багатокутника $P_i, P_{i+1}, \ldots, P_j$ (індекси тут і далі беруться за модулем n). Позначимо увесь багатокутник за $P = P_1 \ldots P_n$, тоді $S(P) = S(i, j) + S(j, i)$. В задачі нас просять мінімізувати $|S(i, j) - S(j, i)|$. Помітимо, що $S(i, j) = S(i, j-1) + S(P_i P_{j-1} P_j) > S(i, j-1)$, а значить $S(i, j) - S(j, i) = 2S(i, j) - S(P)$ — зростаюча функція за j. Це означає, що вона має не більше ніж дві підозрілі на мінімум модулю точки: останню точку P_{j_1}, для якої $S(i, j_1) \leqslant S(j_1, i)$, і першу точку P_{j_2}, для якої $S(i, j_2) \geqslant S(j_2, i)$ ($j_1 \leqslant j_2 \leqslant j_1 + 1$). Помітимо що остання умова означає, що $S(i, j_1) \leqslant \frac{1}{2}S(P)$, а $S(i, j_2) \geqslant \frac{1}{2}S(P)$.

Таким чином, для кожного i нам достатньо розглянути не більш ніж дві точки — P_{j_1} і P_{j_2}. Помітимо, що $\frac{1}{2}S(P) \geqslant S(i, j_1) = S(i+1, j_1) + S(P_i P_{i+1} P_{j_1}) > S(i+1, j_1)$, а отже при збільшенні i, j_1 могла тільки збільшиться, аналогічно j_2. Будемо проводити операцію млина: збільшувати усі індекси i, j_1, j_2 та оновлювати відповідь щоразу коли ми отримаємо валідну трійку: так як кожного разу кожен індекс ми змінюємо на одиницю, то будемо підтримувати і $S(i, j_1) - S(j_1, i)$, і $S(i, j_2) - S(j_2, i)$.

Складність розв'язку — $O(n)$: усі позиції i, j_1, j_2 в сумі зроблять приблизно один оберт (бо після нього це повинні бути ті самі j_1, j_2 що були для першої розглянутої вершини та не більше одного оберту на пошук перших значень.

Задача K. Kinky Letters Movement

Розбір підготував: **Максим Щерба**

Якщо з рядка s_1 за чотири операції можна одержати рядок s_2, то існує послідовність операцій $s_1 \to a \to b \to c \to s_2$. З рядка s_1 можна одержати рядок b за дві операції. Оскільки перевертання підрядка — оборотна операція, то з рядка s_2 також можна одержати рядок b за дві операції. Задача звелася до того, що потрібно перевірити, чи існує такий рядок b, що його можна одержати за дві операції з s_1 і s_2.

Утворимо всі можливі рядки двома перевертаннями підрядка з s_1, обчислимо поліноміальний геш для кожного з них та додамо їх у геш-таблицю (unordered_set у C++). Далі утворимо таким же чином рядки з s_2 та перевіримо, чи є в геш-таблиці такий рядок.

Можливих рядків, які можна одержати за дві операції з s_1 є $O(|s_1|^4)$, для кожного з них обчислюємо геш за $O(|s_1|)$. Складність такого розв'язку: $O(|s_1|^5)$.

Задача L. Live Mercurian Navigation

Розбір підготував: Максим Щерба

Нехай P — множина всіх простих шляхів із міста 1 до міста n, p — деякий такий шлях, k_p — сума c_i по всіх ребрах шляху p, b_p — сума d_i по всіх ребрах шляху p. Тоді $f(t) = \min\limits_{p \in P}\{k_p t + b_p\}$ — опукла вгору кусково-лінійна функція.

Згенеруємо всі шляхи p з множини P. Кожен із них задає пряму $f_p(t) = k_p t + b_p$. Побудуємо опуклу оболонку цих прямих, таким чином знайдемо функцію f у явному вигляді. З огляду на адитивність інтеграла інтеграл на проміжку $[0, T]$ дорівнює сумі інтегралів на проміжках, де функція f — лінійна. Інтеграл від лінійної функції можна обчислити, використовуючи табличні інтеграли: $\int_l^r (kt + b)dt = k\frac{r^2 - l^2}{2} + b(r - l)$.

Складність розв'язку: $O(n!)$.

Задача M. Make Necklace Optimal

Розбір підготував: Денис Смірнов

Переберемо діамант y, який ми залишаємо у намисті: за умовою їх не більше 10. Запишемо решту каменів (починаючи з каменя, який йде за діамантом), які не є діамантами у масив. Помітимо, що поміж цих каменів ми маємо Z, а потім X. Тобто в нас спочатку є незростаюча послідовність, а потім неспадна, а нам треба максимізувати вагу X і Z разом.

Спочатку з'ясуємо як максимізувати вагу Z, якщо відомо, в якому елементі вона закінчується. Будемо підтримувати множину $\{(x_1, v_1), (x_2, v_2), \ldots, (x_k, v_k)\}$ для перших i каменів, де x — розмір останнього взятого каменя, а v — найбільша сумма яку ми змогли досягнути, $x_1 < x_2 < \cdots < x_k$, $v_1 > v_2 > \ldots v_k$. Розглядаючи a_i ми можемо лише додати його до множин $x_1 > \cdots > x_l \geqslant a_i$, помітимо, що серед них максимальна сума це v_l, тобто треба додати пару $(x = a_i, v = v_l + a_i)$, але цим ми могли порушити впорядкованність v, проте якщо $a_i = x > x_j, a_i + x_l = v \geqslant v_j$, то нам немає сенсу брати послідовність, що закінчується на x_j, ми можемо її лише покращити замінивши на (x, v), тому після видалення неоптимальних пар, властивість впорядкованності збережеться. Виконувати всі ці операції можно ефективно за допомогою `std::map<int, int64_t>`.

Таким чином, ми навчилися для кожної позиції з'ясовувати найважчу незростаючу послідовність, що закінчується в данному камені. Розвернувши масив, ми зможемо так само з'ясувати найважчу неспадну послідовність, яка починається у кожному камені. Далі, додавши найлегший камінь з $X \cup Z$, до X і Z, ми можемо вважати що Z закінчується в тому самому елементі що починається X, тобто нам треба знайти максимум з сум двох масивів, що ми знайшли вище (не забувши про те, що спільний елемент треба врахувати один раз).

Помітимо, що шукаючи оптимальну множину X ми виконали n ітерацій, на кожній з яких додали не більше одного елемента, скількись видалили та розглянули, лишивши у мапі, не більше одного, отже (видалень не більше ніж додавань) ми провели $O(n)$ операцій з мапою. Отже, складність розв'язку: $O(10 \cdot n \log n) = O(n \log n)$.