# Розбір задачі «Леді та таксі»

Існує оптимальний розв'язок задачі, у якому буде не більше одного таксі $i$, такого, що ми проїдемо у цьому таксі відстань $l_i$, де $1 \leqslant l_i < Z_i$. У всіх інших таксі $j$ ми проїдемо відстань рівну або 0, або $Z_j$. Також можна довести, що серед таксі у яких ми проїдемо хоча б один кілометр, таксі у якому ми проїдемо не максимальну можливу відстань $Z_i$, буде мати максимальне значення $c_i$, бо інакше на ньому було б оптимально проїхати як мінімум 1 кілометр, який ми могли проїхати за більшу ціну в іншому таксі.

Спочатку відсортуємо усі таксі у порядку зростання $c_i$. Можемо розв'язувати задачу динамічним програмуванням, де стан $dp_{i,j}$ позначає, за яку мінімальну вартість ми можемо проїхати $i$ кілометрів, використовуючи таксі що мають номера не більше за $j$. Можливі переходи будуть наступні:

- $dp_{i,j+1} = min(dp_{i,j}, dp_{i,j+1})$
- $dpi + z_j, j = min(dp_{i+z_j,j}, dp_{i,j-1} + d_j + c_j * z_j)$

У другому переході, якщо $w \leqslant i + z_j$, можна спробувати оновити відповідь, тоді таксі із номером $j$ може бути таксі у якому ми проїдемо не максимальну відстань.

# Розбір задачі «Острів»

Побудуємо граф, у якому вершинами будуть усі дороги в нашому початковому графі. Між кожною парою доріг що належать одному одиничному квадрату проведемо у графі два односторонніх ребра. Виділимо в цьому графі вершини, що відповідають дорогам з одного боку від яких вода, а з іншого одиничний квадрат. Можна довести, що будь-який шлях що проходить між парою різних виділених вершин, буде ділити острів на кілька частин. Додамо ціну ребрам, яка буде рівна ціні дороги, якій відповідає кінцева вершина ребра. Ціна щоб почати шлях у деякій вершині буде рівна ціні відповідної дороги.

Залишилось знайти найдешевший шлях між будь-якою парою виділених вершин. Для цього запустимо алгоритм Дейкстри, який буде зберігати два найоптимальніших шляхи до кожної вершини, а також номер вершини з якої починався кожен шлях. Це знадобиться щоб уникнути випадків циклу, де шлях починається та закінчується в одній і тій самій вершині.

# Розбір задачі «Пазл»

Для кожного значення $i$ від 0 до $k-1$, запам'ятаємо усі позиції від 1 до $n$, що дають залишок $i$ при діленні на $k$. Кожна позиція відповідає деякому рядку довжини $k$, що на ній починається. Кожна група містить усі позиції, що відповідають рядкам при деякому розбитті.

У кожній групі треба визначити оцінку, тобто знайти лексикографічно найменший рядок, і з усіх груп обрати максимальну оцінку. Відсортуємо кожну групу у лексикографічному порядку, тоді оцінкою буде перший рядок.

Для того щоб швидко порівнювати пару рядків застосуємо бінарний пошук та хешування. Бінарним пошуком будемо шукати максимальний префікс де обидва рядки співпадають. Тоді за наступним символом можна визначити який рядок менше.

# Розбір задачі «Дерева»

Спочатку, визначимо, що висота дерева вирощеного під час $i$-го запиту буде не $h_i$, а $h_i - i$. Тоді дерева можна легко порівнювати незалежно від часу.

Будемо зберігати два індексні масиви. В першому масиві ми для позиції $i$ будемо зберігати висоту дерева що росте на цій позиції, а також, значення $d1_i$ — довжина максимальної зростаючої підпослідовності, що починається з дерева на позиції $i$. На цьому масиві побудуємо дерево відрізків, де для кожного відрізку ми зберігаємо мінімальну висоту дерева що на ньому росте ($+\inf$ для позицій де нема дерев), та максимальне значення $d1$.

В другому масиві для позиції $i$ ми будемо зберігати позицію де росте дерево із висотою $i$, а також значення $d2_i$, яке відповідає довжині максимальної зростаючої підпослідовності в першому масиві, яка починається з дерева, що має висоту $i$.

Перейдемо до запитів.

Запит першого типу: оскільки нове дерево буде мати довжину не більше 10, буде існувати не більше 9 дерев що мають меншу висоту, і очевидно, що тільки для них може змінитись значення $d1$ та $d2$, і якщо воно зміниться, то підпослідовність якій будуть відповідати ці значення, буде містити нове дерево. Спочатку знайдемо усі дерева що мають меншу висоту ніж поточне. Тепер знайдемо значення $d1$ для нового дерева. Для цього треба взяти максимум по $d1$ в дереві відрізків першого масиву, проте до критеріїв зупинки під час спуску по дереву відрізків додамо умову що усі дерева на цьому відрізку мають більшу висоту. Тоді ми не більше 9 разів спустимось у вершини які нам не підходять — мають меншу висоту, але знаходяться правіше. Залишилось оновити значення вершин дерев що мають меншу висоту та знаходяться зліва. Оскільки їх значення оновиться лише при умові що їх оптимальна підпослідовність буде проходити через нове дерево, можна перерахувати для них динаміку за квадрат від їх кількості.

Запит другого типу: зліва від видаленого дерева буде не більше 9 старих дерев. Спочатку знайдемо ці дерева, та помітимо їх позиції у другому масиві. Тоді, якщо їх кількість рівна $k$, другий масив розіб'ється на $k + 1$ інтервалів що не містять жодне з цих дерев. У кожному з інтервалів знайдемо максимальне значення $d2$. Тепер, оскільки ми видаляємо деяке дерево, треба спробувати оновити висоту кожного дерева зліва, бо їх оптимальна підпослідовність могла проходити через це дерево. Щоб знайти нове значення $d2$ для певного дерева, нам треба взяти максимальне значення $d2$ серед дерев що мають більшу висоту та знаходяться правіше. Для цього, знайдемо максимум серед максимумів для кожного інтервалу що знаходяться правіше за це дерево у другому масиві. Таким чином ми зможемо знайти значення $d2$ для дерев які ми оновили, але тільки припускаючи що оптимальна з послідовностей для кожного дерева не перетинає жодне інше дерево із тих які треба оновити. Щоб врахувати такі випадки після цього перерахуємо динаміку за квадрат від кількості дерев, як у першому запиті.

Зауважте, що коли ми оновлюємо значення певного дерева ми оновлюємо його значення $d1$ та $d2$ в обох масивах.