**Документальний опис алгоритму:**

Тема задачі: "Trees Made to Order"

Опис задачі: Дано схему нумерації бінарних дерев, де кожне дерево отримує унікальний номер на основі кількості вершин та номерів його піддерев. Потрібно реалізувати алгоритм, який дозволяє вивести бінарне дерево за його номером у заданому форматі.

Структура проекту:

* constants.py: файл для зберігання константи NODE\_NUM\_MAX, яка визначає максимальну кількість вершин у бінарному дереві.
* binary\_tree\_printer.py: файл з класом BinaryTreePrinter, який відповідає за нумерацію бінарних дерев та виведення дерева за заданим номером.
* main.py: файл з головною функцією main(), яка зчитує вхідні дані, створює об'єкт BinaryTreePrinter, викликає методи для виведення дерев та обробляє вихідні дані.
* test\_binary\_tree\_printer.py: файл з тестами для класу BinaryTreePrinter, що перевіряє коректність його роботи.

Алгоритм:

1. Ініціалізація (\_initialize):
   * Обчислюємо кількість можливих дерев з певною кількістю вершин (tree\_num) на основі чисел Каталана.
   * Обчислюємо сумарну кількість дерев до певної кількості вершин (tree\_num\_sum).
   * Ініціалізуємо tree\_num[0] та tree\_num\_sum[0] значенням 1, оскільки існує лише одне пусте дерево.
   * Для кожної кількості вершин node\_num від 1 до NODE\_NUM\_MAX:
     + Обчислюємо кількість можливих дерев з node\_num вершинами, розглядаючи всі можливі варіанти розбиття вершин на ліве та праве піддерева.
     + Оновлюємо tree\_num\_sum[node\_num], додаючи до попереднього значення кількість дерев з node\_num вершинами.
2. Виведення дерева за номером (print\_tree та \_print\_tree\_helper):
   * Визначаємо кількість вершин node\_num у дереві на основі заданого номера order\_num, використовуючи tree\_num\_sum.
   * Рекурсивно обчислюємо номери лівого та правого піддерев:
     + order\_before - кількість дерев з node\_num вершинами, але з меншою кількістю вершин у лівому піддереві, ніж у поточного дерева.
     + left\_node\_num - кількість вершин у лівому піддереві поточного дерева.
     + left\_order\_num - номер лівого піддерева серед дерев з left\_node\_num вершинами.
     + right\_order\_num - номер правого піддерева серед дерев з node\_num - 1 - left\_node\_num вершинами.
   * Формуємо результуючий рядок, який представляє структуру дерева:
     + Якщо left\_node\_num не дорівнює 0 (ліве піддерево не пусте), рекурсивно викликаємо \_print\_tree\_helper для лівого піддерева та додаємо його в дужках перед вершиною.
     + Додаємо вершину X до результуючого рядка.
     + Якщо node\_num - 1 - left\_node\_num не дорівнює 0 (праве піддерево не пусте), рекурсивно викликаємо \_print\_tree\_helper для правого піддерева та додаємо його в дужках після вершини.

Обґрунтування алгоритму:

* Використання чисел Каталана для підрахунку кількості можливих дерев з певною кількістю вершин базується на тому, що n-те число Каталана дорівнює кількості структурно унікальних бінарних дерев з n вершинами.
* Рекурсивний підхід для обчислення номерів піддерев та формування результуючого рядка дозволяє ефективно генерувати дерева за заданим номером.
* Попереднє обчислення та зберігання значень tree\_num та tree\_num\_sum оптимізує процес обчислень та дозволяє швидко визначати номери піддерев.

Приклад розрахунку номерів піддерев:

Розглянемо першу «ітерацію» для дерева під номером 20 з 4 вершинами (node\_num = 4).

* order\_num = 11 (порядковий номер дерева серед дерев з 4 вершинами)
* left\_node\_num = 3 (кількість вершин у лівому піддереві)
* order\_before = 9 (існує 9 дерев з 4 вершинами, але з меншою кількістю вершин у лівому піддереві, ніж 3)
* order\_num - order\_before = 2 (на якому місці знаходиться наше дерево серед дерев з 4 вершинами та 3 вершинами у лівому піддереві)
* N\_R = tree\_num[node\_num - 1 - left\_node\_num] = 0 (кількість можливих правих піддерев з 0 вершинами)

Уявімо таблицю, де рядки відповідають різним можливим деревам для лівого піддерева, а стовпці - різним можливим деревам для правого піддерева.

* (order\_num - order\_before) // N\_R = 2 // 0 = 2 (номер рядка, який є індексом конкретного лівого піддерева)
* (order\_num - order\_before) % N\_R = 2 % 0 = 0 (номер стовпця, який є індексом правого піддерева)

Аналіз алгоритму:

* Часова складність:
  + Ініціалізація (\_initialize): O(NODE\_NUM\_MAX^2), де NODE\_NUM\_MAX - максимальна кількість вершин у бінарному дереві.
  + Виведення дерева за номером (print\_tree та \_print\_tree\_helper): O(node\_num), де node\_num - кількість вершин у дереві.
* Просторова складність:
  + O(NODE\_NUM\_MAX) для зберігання масивів tree\_num та tree\_num\_sum.