**Не рекурсивная реализация.**

****

**1. Назначение программы**

Программа реализует класс BuildTree для **построения бинарного дерева** по заданным правилам. Основные функции:

* Генерация дерева с корнем root и высотой height.
* Автоматическое создание левых и правых потомков по формулам:
  + Левый потомок: родитель \* 3.
  + Правый потомок: родитель + 4.
* Валидация входных данных (типы и значения).

**2. Описание работы кода**

**2.1. Класс BuildTree**

**Атрибуты:**

* root (int): Значение корневого узла.
* height (int): Высота дерева (должна быть > 0).
* tree (dict): Словарь для хранения структуры дерева.

**Метод build:**

1. **Проверка входных данных:**
   * root должен быть целым числом (TypeError при нарушении).
   * height должна быть положительным целым числом (TypeError при нарушении).
2. **Инициализация дерева:**
   * Корень сохраняется в self.tree.
   * Очередь queue для обхода узлов в ширину.
3. **Генерация потомков:**
   * Для каждого узла создаются левый и правый потомки по формулам.
   * Узлы добавляются в очередь для дальнейшей обработки.
   * Цикл выполняется, пока не будет создано 2^height - 1 узлов.

**2.2. Обработка ошибок**

* Блок try-except перехватывает TypeError при создании объекта с некорректными параметрами.

**Рекурсивная реализация.**



**1. Назначение программы**

Программа реализует **рекурсивное построение бинарного дерева** с заданными правилами генерации потомков. Основные функции:

* Создание дерева с корнем root и максимальной высотой height.
* Автоматическая генерация левых и правых потомков по формулам:
  + Левый потомок: родитель \* 3.
  + Правый потомок: родитель + 4.
* Вывод структуры дерева в виде вложенного словаря.

**2. Описание работы кода**

**Рекурсивная функция build\_tree**

**Аргументы:**

* node: Текущий узел дерева.
* height: Максимальная глубина (высота) дерева.
* current\_height: Текущая глубина (по умолчанию 0).

**Логика работы:**

1. **Базовый случай:** Если current\_height >= height, возвращается пустой словарь (прекращение рекурсии).
2. **Рекурсивный шаг:**
   * Левый потомок: node \* 3.
   * Правый потомок: node + 4.
   * Для каждого потомка рекурсивно вызывается build\_tree с увеличением current\_height на 1.
3. **Возврат:** Словарь с текущим узлом и его поддеревьями.