|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования Python»  Вариант 19 | | |
|  | | Работу выполнил  студент группы ИТ-5,6-2024 1 курса  Дорофеев Алексей Анатольевич  «13» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc201073337)

[Алгоритм решения 3](#_Toc201073338)

[Структуры данных: 3](#_Toc201073339)

[Основные алгоритмы: 3](#_Toc201073340)

[Тестирование 5](#_Toc201073341)

[Код программы 7](#_Toc201073342)

# Постановка задачи

В файле даны п целых чисел, и здесь же указан путь их размещения в бинарном дереве виде двоичного кода (коды не повторяются). Построить двоичное дерево целых чисел, в котором путь по дереву определяется указанным двоичным кодом в этом листе (1 - переход к правому потомку, 0 - переход к левому потомку). В корень автоматически заносится значение 0.

# Алгоритм решения

## Структуры данных:

* Класс Node:

data - хранит значение узла

left - указатель на левого потомка

right - указатель на правого потомка

* Класс TreeManager:

Управляет построением дерева

Обеспечивает взаимодействие с пользователем

Реализует методы вывода дерева

* Класс InputHandler (вспомогательный)  
  Обрабатывает и валидирует входные данные.  
  Основные методы:  
  validate\_input(line) - проверяет формат "число код"  
  parse\_file(filename) - читает и парсит файл  
  sort\_entries() - сортирует узлы по длине кода

## Основные алгоритмы:

1. Построение дерева:

* Рекурсивное создание узлов по заданному пути
* Запрос значений для новых узлов
* Обработка конфликтов (попытка перезаписи существующих узлов)

1. Валидация ввода:

* Проверка формата данных (число + бинарный код)
* Проверка допустимых символов
* Валидация выбора режима работы

1. Вывод дерева:

* Список всех узлов с путями
* Графическое представление в консоли

1. Особенности реализации:

* Использование рекурсии для обхода дерева
* Два альтернативных представления дерева
* Интерактивный ввод с проверкой ошибок

# Тестирование

**Тест 1: Корректный ввод из файла**  
Входной файл:

10 0  
20 1  
15 01  
5 00  
Ожидаемый вывод:

root : 0

0 : 10

00 : 5

1 : 20

01 : 15  
**Тест 2: Ручной ввод**

Вводимые данные:

> 100 0

> 200 1

> 150 01

> q

Ожидаемый вывод:

root : 0

0 : 100

1 : 200

01 : 150

**Тест 3: Обработка ошибок**1. Некорректный выбор режима:

1 — из файла, 2 — вручную: 3

Ошибка: введите 1 или 2

2. Некорректный формат данных:

Ручной ввод. Формат: <число> <код 0/1>, 'q' — выход.

> abc 01

Неверный формат, повторите.

3. Файл не найден:

Введите имя файла: ijgk

Ошибка: файл 'ijgk' не найден

# Код программы

Ссылка(https://github.com/alex00770051/ikm2.git)  
  
  
*"""  
Программа для построения двоичного дерева по кодам путей.  
Коды путей представляют собой последовательности из 0 (левый потомок) и 1 (правый потомок).  
"""*class Node:  
 *"""Класс узла двоичного дерева"""* def \_\_init\_\_(self, data=None):  
 self.data = data # Значение узла  
 self.left = None # Левый потомок  
 self.right = None # Правый потомок  
  
  
class TreeManager:  
 *"""Класс для управления двоичным деревом и операциями с ним"""* def \_\_init\_\_(self):  
 *"""Инициализация дерева с корневым узлом (значение 0)"""* self.root = Node(0)  
  
 def ensure\_path(self, node, code, prompt\_intermediate, prompt\_final, final\_value=None):  
 *"""  
 Создает путь в дереве по заданному коду.  
  
 Args:  
 node: текущий узел  
 code: оставшаяся часть кода пути  
 prompt\_intermediate: запрашивать ли значения для промежуточных узлов  
 prompt\_final: запрашивать ли значения для конечных узлов  
 final\_value: значение для установки в конечном узле (если указано)  
  
 Returns:  
 Созданный/найденный конечный узел  
 """* if not code:  
 if final\_value is not None:  
 node.data = final\_value  
 return node  
  
 branch = 'left' if code[0] == '0' else 'right'  
 child = getattr(node, branch)  
  
 if child is None:  
 if (len(code) > 1 and prompt\_intermediate) or (len(code) == 1 and prompt\_final):  
 child = self.\_create\_node\_with\_prompt(code)  
 else:  
 child = Node()  
 setattr(node, branch, child)  
  
 return self.ensure\_path(child, code[1:], prompt\_intermediate, prompt\_final, final\_value)  
  
 def \_create\_node\_with\_prompt(self, code):  
 *"""Запрашивает у пользователя значение для нового узла"""* while True:  
 text = input(  
 f"Введите целое для {'промежуточного' if len(code) > 1 else 'конечного'} узла '{code[0]}' (от '{code}'): ").strip()  
 if text.isdigit():  
 return Node(int(text))  
 print("Ошибка: нужно целое неотрицательное число.")  
  
 def collect\_nodes(self, node, path="", out=None):  
 *"""  
 Собирает все узлы дерева с их путями и значениями.  
  
 Args:  
 node: текущий узел  
 path: текущий путь от корня  
 out: список для сбора результатов  
  
 Returns:  
 Список кортежей (путь, значение)  
 """* if out is None:  
 out = []  
 out.append((path, node.data))  
 if node.left:  
 self.collect\_nodes(node.left, path + "0", out)  
 if node.right:  
 self.collect\_nodes(node.right, path + "1", out)  
 return out  
  
 def print\_all(self):  
 *"""Выводит полную информацию о дереве"""* print("\nСписок узлов (код путь : значение):")  
 for path, val in self.collect\_nodes(self.root):  
 print(f"{path or 'root':>5} : {val}")  
  
 print("\nГоризонтальное представление дерева:")  
 self.\_print\_tree(self.root)  
  
 def \_print\_tree(self, node, level=0):  
 *"""  
 Рекурсивно печатает дерево в горизонтальном формате.  
  
 Args:  
 node: текущий узел  
 level: текущий уровень вложенности  
 """* if node.right:  
 self.\_print\_tree(node.right, level + 1)  
 print(" " \* level + f"-> {node.data}")  
 if node.left:  
 self.\_print\_tree(node.left, level + 1)  
  
  
class InputHandler:  
 *"""Класс для обработки входных данных (из файла или ручного ввода)"""* def \_\_init\_\_(self):  
 self.entries = [] # Список пар (число, код)  
  
 def process\_file(self, filename):  
 *"""  
 Читает и обрабатывает данные из файла.  
  
 Args:  
 filename: имя файла для чтения  
  
 Returns:  
 Отсортированный список входных данных или None при ошибке  
 """* try:  
 with open(filename, 'r', encoding='utf-8') as f:  
 for lineno, line in enumerate(f, 1):  
 self.\_process\_line(line, lineno)  
 return self.\_sort\_entries()  
 except FileNotFoundError:  
 print(f"Ошибка: файл '{filename}' не найден.")  
 return None  
  
 def process\_manual\_input(self):  
 *"""Обрабатывает ручной ввод данных пользователем"""* print("Ручной ввод. Формат: <число> <код 0/1>, 'q' — выход.")  
 while True:  
 s = input("> ").strip()  
 if s.lower() == 'q':  
 break  
 self.\_process\_line(s)  
 return self.\_sort\_entries()  
  
 def \_process\_line(self, line, lineno=None):  
 *"""  
 Парсит одну строку входных данных.  
  
 Args:  
 line: строка для обработки  
 lineno: номер строки (для сообщений об ошибках)  
 """* text = line.strip()  
 if not text:  
 return  
 parts = text.split()  
 if len(parts) != 2 or not parts[0].isdigit() or any(ch not in '01' for ch in parts[1]):  
 if lineno:  
 print(f"[Строка {lineno}] Пропущена: ожидается '<число> <код>'.")  
 else:  
 print("Неверный формат, повторите.")  
 return  
 self.entries.append((int(parts[0]), parts[1]))  
  
 def \_sort\_entries(self):  
 *"""Сортирует входные данные по длине кода (от коротких к длинным)"""* for i in range(1, len(self.entries)):  
 key = self.entries[i]  
 j = i - 1  
 while j >= 0 and len(self.entries[j][1]) > len(key[1]):  
 self.entries[j + 1] = self.entries[j]  
 j -= 1  
 self.entries[j + 1] = key  
 return self.entries  
  
  
def main():  
 *"""Основная функция программы"""* print("=== Построение двоичного дерева по кодам путей ===")  
  
 while True:  
 # Выбор режима ввода  
 while True:  
 mode = input("1 — из файла, 2 — вручную: ").strip()  
 if mode in ('1', '2'):  
 break  
 print("Ошибка: введите 1 или 2")  
  
 manager = TreeManager()  
 input\_handler = InputHandler()  
  
 # Обработка выбранного режима  
 if mode == '1':  
 filename = input("Введите имя файла: ").strip()  
 entries = input\_handler.process\_file(filename)  
 if entries is None: # Если файл не найден, начинаем сначала  
 continue  
 else:  
 entries = input\_handler.process\_manual\_input()  
  
 # Построение дерева  
 for number, code in entries:  
 node = manager.ensure\_path(manager.root, code,  
 prompt\_intermediate=True,  
 prompt\_final=False,  
 final\_value=number)  
 if node.data != number:  
 print(f"[Код '{code}'] узел уже = {node.data}, пропуск {number}.")  
  
 # Вывод результатов  
 manager.print\_all()  
 break # Выход из основного цикла после успешного выполнения  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()