|  |  |  |
| --- | --- | --- |
| 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-8 1 курса  Блажко И.А.  «\_\_\_» \_\_\_\_\_\_\_\_2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 7](#_Toc153130029)

[Код программы 11](#_Toc153130030)

# Постановка задачи

В файле даны n 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. Учитывать ситуацию, когда дерево не может быть построено.

# Алгоритм решения

Эта программа строит и визуализирует бинарное дерево на основе данных, прочитанных из Excel-файла. Данные в файле указывают, какое число должно быть вставлено в дерево и по какому пути (последовательности '0' и '1') к нему добраться от корня.

1**. Класс Node (Узел дерева):**

• Назначение: Представляет один узел в бинарном дереве.

• Структура данных:

• data: Содержит значение, хранящееся в узле (целое число).

• left: Указатель на левого потомка. Если левого потомка нет, содержит None.

• right: Указатель на правого потомка. Если правого потомка нет, содержит None.

• count: Счетчик, показывающий, сколько раз данное значение встречается в узле (для обработки дубликатов).

• Обоснование: Бинарное дерево - это структура данных, состоящая из узлов, соединенных связями. Каждый узел имеет данные и ссылки на другие узлы (потомки). None используется для обозначения отсутствия потомка. count добавляет возможность хранить одинаковые значения в дереве, просто увеличивая счетчик, а не создавая новый узел.

2. **Класс Tree (Дерево):**

• Назначение: Представляет бинарное дерево целиком и содержит методы для работы с ним.

• Структура данных:

• root: Указатель на корневой узел дерева.

**• Методы:**

• **\_\_init\_\_(self):** Конструктор класса. Создает новый объект Tree и инициализирует root корневым узлом со значением 0. Корень всегда существует.

• **insert(self, node, data, path, index=0):** Вставляет новый узел с данными data в дерево, следуя по пути path. node - текущий узел, в котором идет поиск места для вставки. index отслеживает текущую позицию в строке path.

\* Алгоритм вставки:

1. Базовый случай (рекурсия): Если index достиг конца пути (path), значит, найдено место для вставки.

\* Если в текущем узле (node) уже есть значение data, увеличиваем счетчик count.

\* Если в текущем узле значение отличается от 0, значит, по этому пути уже есть другое число, выбрасываем исключение.

\* Иначе, записываем data в node.data.

2. Рекурсивный шаг:

\* Берем текущий символ direction из пути path (либо '0', либо '1').

\* Если direction равен '0', переходим к левому потомку. Если левого потомка не существует, создаем новый узел Node и присваиваем его. Затем рекурсивно вызываем insert для левого потомка.

\* Если direction равен '1', делаем то же самое для правого потомка.

\* Если direction не равен ни '0', ни '1', выбрасываем исключение, т.к. путь некорректен.

• **check\_missing\_nodes(self, data):** Проверяет, не пропущены ли промежуточные узлы в путях, указанных в данных.

\* Алгоритм проверки:

1. Извлекает все пути из входных данных data.

2. Сортирует список путей, чтобы более короткие пути шли раньше более длинных (это важно для логики проверки). Реализована сортировка без использования встроенной функции.

3. Перебирает все пары путей, где второй путь длиннее первого.

4. Если второй путь начинается с первого, то второй путь является потомком первого. Нужно проверить, существует ли узел, представляющий непосредственного потомка первого пути.

5. Вычисляет путь к этому непосредственному потомку (missing\_path).

6. Если missing\_path отсутствует в списке путей, значит, узел пропущен, и выбрасывается исключение.

**3. Функция read\_excel\_data(file\_path):**

• Назначение: Читает данные из Excel-файла, проверяет их корректность и возвращает список кортежей (число, путь).

• Алгоритм:

1. Использует pandas для чтения Excel-файла в DataFrame.

2. Перебирает строки DataFrame.

3. Для каждой строки:

\* Извлекает число и путь из столбцов.

\* Проверяет, что число можно преобразовать в целое число.

\* Проверяет, что число не содержит ведущих нулей.

\* Проверяет, что путь состоит только из символов '0' и '1'.

\* Добавляет кортеж (число, путь) в список data.

4. Возвращает список data.

• Типы данных:

• file\_path: строка (путь к файлу).

• df: pandas.DataFrame (таблица данных).

• data: список кортежей (целое число, строка).

• Обоснование: Функция инкапсулирует логику чтения и валидации данных, что делает основной код более чистым. Использование pandas упрощает чтение Excel-файла. DataFrame позволяет удобно работать с табличными данными.

**4. Функция visualize\_tree(root):**

• Назначение: Визуализирует бинарное дерево с помощью библиотеки networkx и matplotlib.

• Алгоритм:

1. Создает граф networkx.

2. Использует обход дерева в ширину (BFS) для создания координат узлов для визуализации.

3. Для каждого узла создает метку, содержащую значение узла и счетчик дубликатов (если есть).

4. Добавляет узлы и ребра в граф.

5. Использует matplotlib для отрисовки графа с заданными координатами и метками.

• Обоснование: Визуализация позволяет наглядно увидеть структуру дерева и проверить правильность его построения. networkx предоставляет удобные инструменты для работы с графами, а matplotlib - для их отрисовки.

**5. Основной блок кода**

• Алгоритм:

1. Читает данные из Excel-файла с помощью read\_excel\_data.

2. Создает экземпляр класса Tree.

3. Проверяет наличие пропущенных узлов с помощью tree.check\_missing\_nodes.

4. Если проверка прошла успешно, вставляет узлы в дерево с помощью tree.insert.

5. Визуализирует дерево с помощью visualize\_tree.

• Обработка исключений:

• FileNotFoundError: Перехватывает исключение, если файл tree\_data.xlsx не найден.

• ValueError: Перехватывает исключения, возникающие при чтении данных, проверке корректности данных, при вставке узлов (например, при конфликте путей) или при обнаружении пропущенных узлов.

**Обоснование общих решений:**

• Бинарное дерево: Выбрано, потому что пути в данных бинарные (состоят из '0' и '1'). Это позволяет эффективно организовывать данные в иерархическую структуру.

• Рекурсия в insert: Рекурсия естественно подходит для обхода древовидных структур.

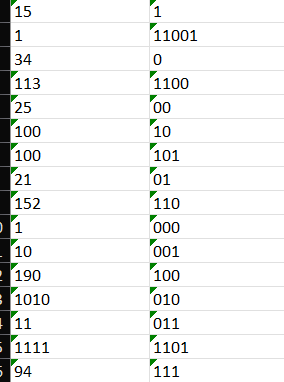
• Обработка исключений: Важна для обеспечения надежности программы. Позволяет корректно обрабатывать ошибки, возникающие при чтении данных, проверке их корректности или при работе с деревом.

• Использование pandas, networkx и matplotlib: Эти библиотеки использовались для чтения данных из excel-файла (pandas) и наглядного примера построения двоичного дерева (matplotlib и networkx)

• Строковый тип данных для путей: Пути состоят из последовательности символов '0' и '1', поэтому строковый тип является наиболее подходящим для их представления.

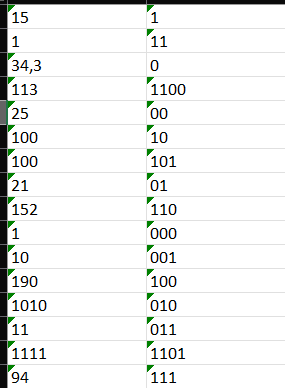
# Тестирование

Отсутствие промежуточного узла:

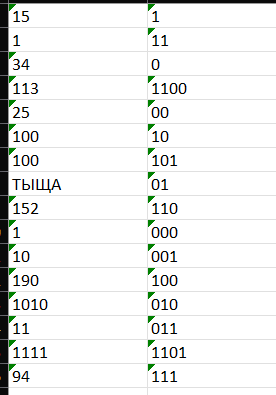




В таблице содержится не целое число:

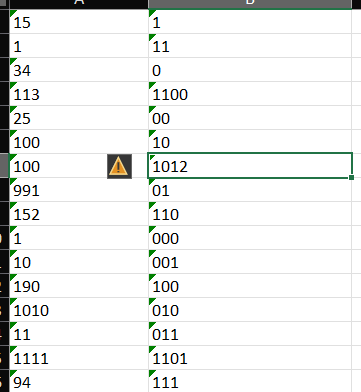






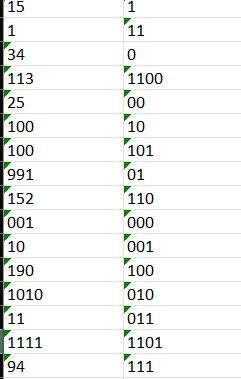


Неправильно задан путь узла:





В числе содержатся ведущие нули:

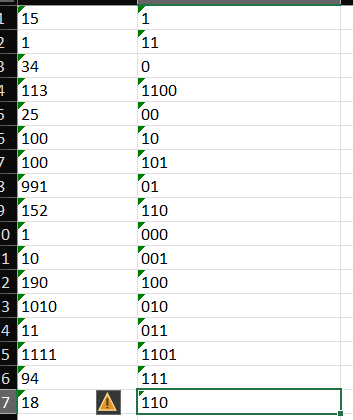




Отсутствие файла:



Узел уже занят другим числом:





# Код программы

import pandas as pd  
import matplotlib.pyplot as plt  
import networkx as nx  
  
  
class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.left = None  
 self.right = None  
 self.count = 1  
  
  
class Tree:  
 def \_\_init\_\_(self):  
 self.root = Node(0)  
  
 def insert(self, node, data, path, index=0):  
 if index == len(path):  
 if node.data == data:  
 node.count += 1  
 elif node.data != 0:  
 raise ValueError(f"Ошибка: число {data} не помещается по пути {path}")  
 else:  
 node.data = data  
 return  
  
 direction = path[index]  
  
 if direction == '0':  
 if node.left is None:  
 node.left = Node(0)  
 self.insert(node.left, data, path, index + 1)  
 elif direction == '1':  
 if node.right is None:  
 node.right = Node(0)  
 self.insert(node.right, data, path, index + 1)  
 else:  
 raise ValueError(f"Ошибка в строке {index + 1}: в пути '{path}' есть что-то кроме 0 и 1")  
  
 def check\_missing\_nodes(self, data):  
 paths = [path for \_, path in data]  
 # Сортировка без sort()  
 for i in range(len(paths)):  
 for j in range(i + 1, len(paths)):  
 if paths[i] > paths[j]:  
 paths[i], paths[j] = paths[j], paths[i]  
  
 for i in range(len(paths)):  
 for j in range(i + 1, len(paths)):  
 path1 = paths[i]  
 path2 = paths[j]  
 if len(path2) >= len(path1) and path2[:len(path1)] == path1 and len(path2) > len(path1):  
 missing\_path = path2[:len(path1) + 1]  
 if missing\_path not in paths:  
 raise ValueError(f"Ошибка: не хватает узла '{missing\_path}'")  
 return True  
  
  
def read\_excel\_data(file\_path):  
 try:  
 df = pd.read\_excel(file\_path, header=None, dtype={0: str, 1: str})  
 except Exception as e:  
 raise ValueError(f"Ошибка: не получилось прочитать файл: {e}")  
  
 data = []  
 for index, row in df.iterrows():  
 num\_str = str(row[0]).strip()  
 path = str(row[1]).strip()  
  
 try:  
 number = int(num\_str)  
 except (ValueError, TypeError):  
 raise ValueError(f"Ошибка в строке {index + 1}: '{num\_str}' - это не целое число")  
  
 if num\_str.startswith('0') and len(num\_str) > 1:  
  
 raise ValueError(f"Ошибка в строке {index + 1}: в числе '{num\_str}' есть ведущие нули")  
  
 if not all(c in ('0', '1') for c in path):  
 raise ValueError(f"Ошибка в строке {index + 1}: в пути '{path}' есть что-то кроме 0 и 1")  
  
 data.append((number, path))  
  
 return data  
  
  
def visualize\_tree(root):  
 G = nx.Graph()  
 pos = {}  
 labels = {}  
 queue = [(root, (0, 0))]  
 level\_height = 2  
  
 while queue:  
 node, (x, y) = queue.pop(0)  
 label = f"{node.data}" + (f" (x{node.count})" if node.count > 1 else "")  
 labels[node] = label  
 pos[node] = (x, -y)  
  
 if node.left:  
 G.add\_edge(node, node.left)  
 left\_x = x - 1 / (2 \* (y / level\_height + 1))  
 queue.append((node.left, (left\_x, y + level\_height)))  
  
 if node.right:  
 G.add\_edge(node, node.right)  
 right\_x = x + 1 / (2 \* (y / level\_height + 1))  
 queue.append((node.right, (right\_x, y + level\_height)))  
  
 plt.figure(figsize=(12, 8))  
 nx.draw(G, pos, labels=labels, with\_labels=True, node\_size=2000,  
 node\_color='skyblue', font\_size=10, font\_weight='bold')  
 plt.title("Бинарное дерево", size=15)  
 plt.show()  
  
  
try:  
 data = read\_excel\_data("tree\_data.xlsx")  
  
 tree = Tree()  
  
 if tree.check\_missing\_nodes(data):  
 for number, path in data:  
 tree.insert(tree.root, number, path)  
  
 print("Дерево построено!")  
 visualize\_tree(tree.root)  
  
except FileNotFoundError:  
 print("Ошибка: файл 'tree\_data.xlsx' не найден")  
except ValueError as e:  
 print(f"Ошибка: {e}")