|  |  |  |
| --- | --- | --- |
| 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  студентка группы ИТ-5-2024 1 курса  Носкова А. В  «12» июня 2025 г. |
| Работу проверил  Рубцова М. Б  «12» июня 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200756681)

[Алгоритм решения 4](#_Toc200756682)

[Тестирование 5](#_Toc200756683)

[Код программы 6](#_Toc200756684)

# 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

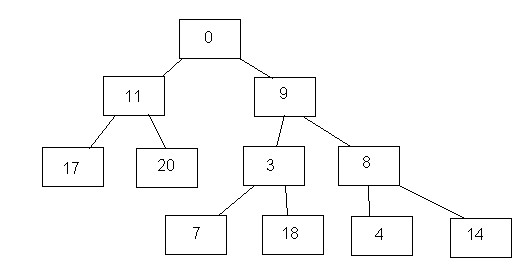
в этом листе (1 – переход к правому потомку левому потомку). В корень автоматически

заносится значение 0.

Например, для исходных данных:

|  |  |
| --- | --- |
| 15 | 111 |
| 18 | 101 |
| 3 | 10 |
| 8 | 11 |
| 9 | 1 |
| 11 | 0 |
| 7 | 100 |
| 4 | 110 |
| 17 | 00 |
| 20 | 01 |

должно быть построено такое дерево:



Учитывать ситуацию, когда дерево не может быть построено

# Алгоритм решения

1. **Постановка задачи**

Нужно построить бинарное дерево на основе введённых пользователем данных или данных, считанных из файла. Каждый узел дерева содержит ключ (число) и путь (последовательность из '0' и '1'), где '0' означает переход к левому потомку, а '1' — к правому.

* Корень дерева всегда имеет ключ 0.
* Конечные узлы (листья) содержат положительные целые числа.
* Пути должны быть уникальными и корректными (только символы '0' и 1.

1. **Ключевая идея**

Использование рекурсивной структуры данных (бинарное дерево) и двух стеков для валидации:

* TreeNode — узел дерева с ключом и ссылками на потомков.
* BinaryTree — класс для управления деревом и проверки его корректности.

1. **Подробный алгоритм**

**1. Ввод данных**

* **Ввод вручную**:
* Пользователь вводит данные в формате: <значение> <путь>.
* Путь состоит из символов 0 (левый потомок) и 1 (правый потомок).
* При вводе end программа переходит к проверке дерева.
* **Чтение из файла**:
* Данные считываются построчно.
* Каждая строка обрабатывается так же, как при вводе вручную.
* При обнаружении ошибок (неверный формат, конфликты значений) выводятся предупреждения.

**2. Построение дерева**

* **Вставка узла (insertValue)**:
* Путь разбивается на символы, и дерево строится рекурсивно.
* Если узел уже существует, проверяется совпадение значений.
* Если путь содержит промежуточные узлы, они создаются автоматически (с key = None).
* **Проверка пропущенных узлов (findNodes)**:
* Собираются все префиксы путей (например, для пути 101 префиксы: 1, 10).
* Если узел с таким префиксом отсутствует или не имеет значения, он добавляется в missing\_nodes.
* **Заполнение пропущенных узлов**:
* Пользователю предлагается ввести значения для всех узлов из missing\_nodes.
* Узлы сортируются по длине пути (от корня к листьям).

**3. Валидация дерева (validateTree):**

* Проверяется, что корень имеет значение 0.
* Проверяется, что все узлы (конечные и промежуточные) имеют значения.
* Выводятся обнаруженные ошибки (если есть).

**4. Вывод результата**

* Печатается структура дерева.
* Выводится отчет об ошибках или успешном построении.

# Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемый результат** | **Проверяемый случай** |
| 1 | 5 0 | Корректное дерево | Простое добавление листа |
| 2 | 7 0  8 01  9 011 | Корректное дерево | Вложенные пути |
| 3 | 3 11  4 11 | Конфликт значений | Дублирование пути |
| 4 | 9 2 | Недопустимый символ | Неверный символ пути |
| 5 | 10 | Неверный формат | Отсутствует путь |
| 6 | “ “ | Неверный формат | Пустая строка |
| 7 | 4 j | Недопустимый символ | Неверный символ пути |
| 8 | k 01 | Неверный формат | Неверный символ значения |
| 9 | 6 0  7 001 | Некорректное дерево | Пропущен узел 00 |

# Код программы

class TreeNode:  
 def \_\_init\_\_(self, key=None, left=None, right=None):  
 self.key = key  
 self.left = left  
 self.right = right  
 self.visited = False  
  
  
class BinaryTree:  
 def \_\_init\_\_(self):  
 self.root = TreeNode(0)  
 self.all\_paths = set()  
 self.errors = []  
 self.missing\_nodes = set()  
  
 def bubbleSortByLength(self, items):  
 *"""Сортировка пузырьком по длине строки"""* n = len(items)  
 for i in range(n):  
 for j in range(0, n - i - 1):  
 if len(items[j]) > len(items[j + 1]):  
 items[j], items[j + 1] = items[j + 1], items[j]  
 return items  
  
 def findNodes(self):  
 *"""Поиск пропущенных узлов"""* self.missing\_nodes = set()  
 paths\_list = list(self.all\_paths)  
 sorted\_paths = self.bubbleSortByLength(paths\_list)  
  
 for path in sorted\_paths:  
 for i in range(1, len(path)):  
 prefix = path[:i]  
 node = self.getNode(prefix)  
 if node is None or node.key is None:  
 self.missing\_nodes.add(prefix)  
  
 def insertValue(self, key, path):  
 *"""Вставка значения с обработкой ошибок"""* try:  
 current = self.root  
 self.all\_paths.add(path)  
  
 # Добавляем все префиксы пути  
 for i in range(1, len(path) + 1):  
 self.all\_paths.add(path[:i])  
  
 index = 0  
 while index < len(path):  
 bit = path[index]  
 if bit == '0':  
 if index == len(path) - 1:  
 if current.left is None:  
 current.left = TreeNode(key)  
 else:  
 if current.left.key is not None and current.left.key != key:  
 raise ValueError(f"Конфликт в узле {path}: {key} ≠ {current.left.key}")  
 current.left.key = key  
 current.left.visited = True  
 else:  
 if current.left is None:  
 current.left = TreeNode()  
 current = current.left  
 elif bit == '1':  
 if index == len(path) - 1:  
 if current.right is None:  
 current.right = TreeNode(key)  
 else:  
 if current.right.key is not None and current.right.key != key:  
 raise ValueError(f"Конфликт в узле {path}: {key} ≠ {current.right.key}")  
 current.right.key = key  
 current.right.visited = True  
 else:  
 if current.right is None:  
 current.right = TreeNode()  
 current = current.right  
 index += 1  
 return True  
 except ValueError as e:  
 self.errors.append(str(e))  
 return False  
  
 def validateTree(self):  
 *"""Проверка целостности дерева"""* errors = []  
  
 if self.root.key != 0:  
 errors.append("Корень ≠ 0")  
  
 paths\_list = list(self.all\_paths)  
 sorted\_paths = self.bubbleSortByLength(paths\_list)  
  
 for path in sorted\_paths:  
 node = self.getNode(path)  
 if node is None or node.key is None:  
 errors.append(f"Узел '{path}' без значения")  
  
 self.findNodes()  
 missing\_list = list(self.missing\_nodes)  
 sorted\_missing = self.bubbleSortByLength(missing\_list)  
  
 for path in sorted\_missing:  
 errors.append(f"Пропущен промежуточный узел '{path}'")  
  
 return errors  
  
 def getNode(self, path):  
 *"""Получение узла по пути"""* current = self.root  
 for bit in path:  
 if bit == '0':  
 if current.left is None:  
 return None  
 current = current.left  
 elif bit == '1':  
 if current.right is None:  
 return None  
 current = current.right  
 return current  
  
 def printTree(self):  
 *"""Печать структуры дерева"""* def printHelper(node, prefix="", is\_left=True):  
 if node:  
 if node.right:  
 printHelper(node.right, prefix + ("│ " if is\_left else " "), False)  
 print(prefix + ("└── " if is\_left else "┌── ") + str(node.key if node.key is not None else "None"))  
 if node.left:  
 printHelper(node.left, prefix + (" " if is\_left else "│ "), True)  
  
 print("\nГоризонтальное представление дерева:")  
 printHelper(self.root)  
  
  
class TreeManager:  
 def \_\_init\_\_(self):  
 self.tree = BinaryTree()  
 self.line\_num = 1  
  
 def validatePath(self, path):  
 *"""Проверка корректности пути"""* if not path:  
 return False  
 for char in path:  
 if char not in ('0', '1'):  
 return False  
 return True  
  
 def processInputLine(self, line):  
 *"""Обработка строки ввода"""* line = line.strip()  
 if not line:  
 return False  
  
 parts = line.split()  
 if len(parts) != 2:  
 print(f"Ошибка формата в строке {self.line\_num}")  
 self.line\_num += 1  
 return False  
  
 value, path = parts  
 if not self.validatePath(path):  
 print(f"Неверный путь в строке {self.line\_num}")  
 self.line\_num += 1  
 return False  
  
 try:  
 value = int(value)  
 except ValueError:  
 print(f"Нечисловое значение в строке {self.line\_num}")  
 self.line\_num += 1  
 return False  
  
 if not self.tree.insertValue(value, path):  
 print(f"Ошибка в строке {self.line\_num}: {self.tree.errors[-1]}")  
  
 self.line\_num += 1  
 return True  
  
 def fillNodes(self):  
 *"""Заполнение пропущенных узлов"""* self.tree.findNodes()  
 if self.tree.missing\_nodes:  
 print("\nНеобходимо заполнить промежуточные узлы:")  
 missing\_list = list(self.tree.missing\_nodes)  
 sorted\_missing = self.tree.bubbleSortByLength(missing\_list)  
  
 for path in sorted\_missing:  
 while True:  
 try:  
 value = input(f"Введите значение для узла '{path}': ").strip()  
 if not value:  
 print("Значение не может быть пустым!")  
 continue  
 value = int(value)  
 self.tree.insertValue(value, path)  
 break  
 except ValueError:  
 print("Ошибка: введите целое число!")  
  
 def interactiveInput(self):  
 *"""Интерактивный ввод данных"""* print("\nВвод данных (формат: <значение> <путь>):")  
 print("Пример: 15 101")  
 print("Для завершения введите 'end'")  
  
 while True:  
 line = input(f"[{self.line\_num}]> ").strip()  
  
 if line.lower() == 'end':  
 if not self.tree.all\_paths:  
 print("Нет данных. Введите хотя бы одно значение.")  
 continue  
 self.fillNodes()  
 break  
  
 self.processInputLine(line)  
  
 def processFileLine(self, line):  
 *"""Обработка строки файла"""* line = line.strip()  
 if not line:  
 return False  
  
 parts = line.split()  
 if len(parts) != 2:  
 print(f"Пропуск строки {self.line\_num}: неверный формат")  
 self.line\_num += 1  
 return False  
  
 value, path = parts  
 if not self.validatePath(path):  
 print(f"Пропуск строки {self.line\_num}: неверный путь")  
 self.line\_num += 1  
 return False  
  
 try:  
 value = int(value)  
 except ValueError:  
 print(f"Пропуск строки {self.line\_num}: нечисловое значение")  
 self.line\_num += 1  
 return False  
  
 if not self.tree.insertValue(value, path):  
 print(f"Ошибка в строке {self.line\_num}: {self.tree.errors[-1]}")  
  
 self.line\_num += 1  
 return True  
  
 def fileInput(self, filename):  
 *"""Чтение данных из файла"""* try:  
 with open(filename, 'r') as f:  
 for line in f:  
 self.processFileLine(line)  
  
 if not self.tree.all\_paths:  
 print("Файл не содержит корректных данных!")  
 return False  
  
 # Проверяем и заполняем пропущенные узлы сразу после чтения файла  
 self.tree.findNodes()  
 if self.tree.missing\_nodes:  
 print("\nОбнаружены пропущенные узлы при чтении файла:")  
 missing\_list = list(self.tree.missing\_nodes)  
 sorted\_missing = self.tree.bubbleSortByLength(missing\_list)  
  
 for path in sorted\_missing:  
 while True:  
 try:  
 value = input(f"Введите значение для узла '{path}': ").strip()  
 if not value:  
 print("Значение не может быть пустым!")  
 continue  
 value = int(value)  
 self.tree.insertValue(value, path)  
 break  
 except ValueError:  
 print("Ошибка: введите целое число!")  
  
 return True  
  
 except FileNotFoundError:  
 print("Файл не найден")  
 return False  
 except Exception as e:  
 print(f"Ошибка чтения: {e}")  
 return False  
  
 def showResults(self):  
 *"""Вывод результатов"""* errors = self.tree.validateTree()  
 print("\n" + "=" \* 40)  
 if errors:  
 print("Обнаружены проблемы:")  
 for error in errors:  
 print(f"• {error}")  
 else:  
 print("Дерево построено успешно!")  
 self.tree.printTree()  
  
def main():  
 print("Построение бинарного дерева")  
 print("--------------------------")  
  
 manager = TreeManager()  
  
 while True:  
 print("\nМеню:")  
 print("1. Ручной ввод")  
 print("2. Загрузка из файла")  
 print("3. Выход")  
  
 choice = input("Выберите действие (1-3): ").strip()  
  
 if choice == '1':  
 manager.interactiveInput()  
 manager.showResults()  
 elif choice == '2':  
 filename = input("Введите имя файла: ").strip()  
 if manager.fileInput(filename):  
 manager.showResults()  
 elif choice == '3':  
 print("Завершение работы")  
 break  
 else:  
 print("Некорректный выбор, попробуйте снова")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()