|  |  |  |
| --- | --- | --- |
| 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 «Языки программирования»  Вариант 1 | | |
|  | | Работу выполнил  студент группы ИТХ-3,4-2023 1 курса  Батраков Я. А.  «9» июля 2024 г. |
| Работу проверила  старший преподаватель Анисимова С. И.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

**СОДЕРЖАНИЕ**

[Постановка задачи 3](#_30j0zll)

[Алгоритм решения 3](#_1fob9te)

[Тестирование](#_3znysh7) 4

[Код программы](#_2et92p0) 6

# Постановка задачи

В файле даны n целых чисел, и здесь же указан путь их размещения в бинарном дереве в виде двоичного кода (коды не повторяются). Построить двоичное бинарное 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 дерева автоматически заносится значение 0. Учитывать ситуацию, когда дерево не может быть построено.

# Алгоритм решения

Алгоритм решения задачи разделяется на пункты: чтение данных из файла, проверка прочтённых данных, построение дерева, проверка корректности постронения и вывод дерева в наглядном для чтения формате.

Для прочтения данных из файла был отдельно реализован класс FileInstance для работы с данными файла. Класс инициализируется путём к файлу для чтения. После файл считывается и все его строки помещаются в двумерный список с предварительным удалением символов новой строки (“\n”). При помощи метода get\_data() возвращается этот двумерный список. Далее в теле цикла каждая строка двумерного списка разделяется по пробелу и переносится в другой двумерный список (data). Следующим этапом инициализируется дерево, в его головной корень заносится значение 0. Запускается цикл по data. Перед занесением значения в дерево, необходимо проверить корректность данных. Проверяется, что данные - это строка с числом и строка с путём, содержащая только “0” и “1”, если иначе, то выводится ошибка с текстом “Некорректные данные” вместе с некорректной строкой, путём к исходному файлу и номером строки, где обнаружена ошибка. После чего некорректные данные пропускаются. Если данные оказались корректными, то они передаются ф-ции для их добавления в дерево. Ф-ция добавления данных в дерево работает по следующему алгоритму: берётся головной узел, создаётся цикл по переданному маршруту, если “0”, то переход к левому потомку, а если “1”, то к правому. Если какой-то из потомков отсутствует, то создаётся новый со значением “None”. Когда поиск был окончен, то искомому узлу задаётся передаваемое число. Дерево построено, но внутри него могут быть узлы со значением “None” и это неправильно. После сборки дерева вводится отдельная проверка на эти “None”. Если они были найдены, то вывод пользователю сообщения, что дерево некорректное, подсказываем ему, что необходимо восполнить недостающие маршруты и выводим само это дерево по форме в одну строку. Если всё в порядке, то выводим пользователю сообщение, что дерево успешно построено и выводим дерево по форме в одну строку. Форма вывода дерева имеет следующий вид: <дерево>::=((<левое поддерево>)<вершина>(<правое поддерево>)) | ((<левое поддерево>)<вершина>) | (<вершина>(<правое поддерево>)) <вершина>::=<цифра><цифра> | <цифра> <левое поддерево>::=<дерево> <правое поддерево>::=<дерево>. Например, “(((1)2((3)4))5(6(7)))”.

# Тестирование

Тестовый случай 1:

1. Входные данные в файле:

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

1. Скрины работы:

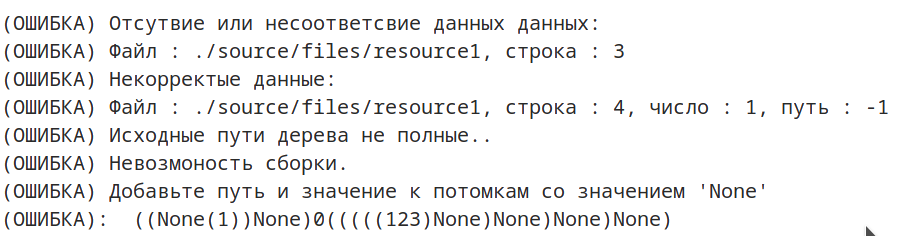


Тестовый случай 2:

1. Входные данные в файле

|  |  |
| --- | --- |
| Число | Путь |
| 1 | 001 |
| 123 | 10000 |
| “” |  |
| 1 | -1 |

1. Скрины работы

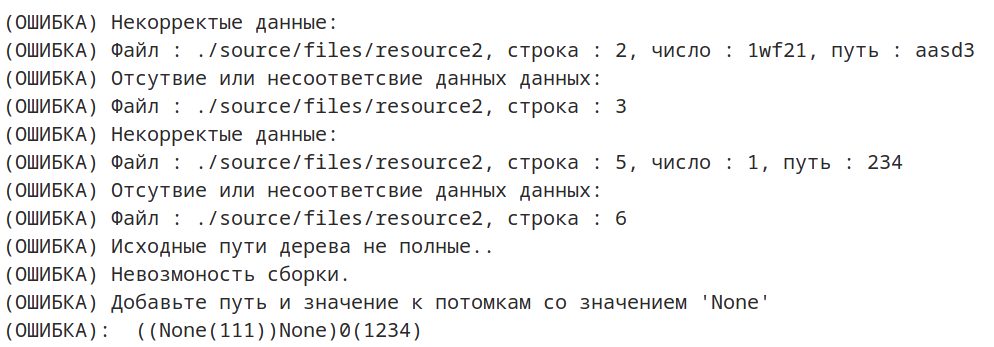


Тестовый случай 2:

1. Входные данные в файле

|  |  |
| --- | --- |
| Число | Путь |
| 1234 | 1 |
| 1wf21 | aasd3 |
| 2 | qwe re |
| 111 | 001 |
| 1 | 234 |

1. Скрины работы



# Код программы

Файл main.py:

**from** **source.IOFILESTREEM** **import** FileInstance **as** File

**from** **source.TREE** **import** BinaryTree

# validated data

**def** **vdata**(data):

**try**:

**return** int(data)

**except**:

**return** None

# validated path

**def** **is\_path**(path : str) -> bool:

**for** word **in** path:

**if** **not** word **in** '01': **return** False

**return** True

**def** **main**(path = "./source/files/resource2"):

**print**()

# read the data and div ' '

data = []

**for** line **in** File(path).data():

data.append(line.split())

# build tree

tree = BinaryTree()

**for** line **in** range(**0**, len(data)):

apnd = data[line]

**if** len(apnd) == **2**:

**if** **not** vdata(apnd[**0**]) **is** None **and** **not** vdata(apnd[**1**]) **is** None:

**if** is\_path(apnd[**1**]):

tree.append\_bypath(data=int(apnd[**0**]), path=apnd[**1**])

**else**:

**print**("(ОШИБКА) Некорректые данные:")

**print**(f"(ОШИБКА) Файл : {path}, строка : {line+1}, число : {apnd[0]}, путь : {apnd[1]}")

**else**:

**print**("**\n**(ОШИБКА) Некорректые данные:")

**print**(f"(ОШИБКА) Файл : {path}, строка : {line+1}, число : {apnd[0]}, путь : {apnd[1]}")

**else**:

**print**(f"(ОШИБКА) Отсутвие или несоответсвие данных данных:")

**print**(f"(ОШИБКА) Файл : {path}, строка : {line+1}")

**if** **not** tree.verified(tree.\_head):

**print**("(ОШИБКА) Исходные пути дерева не полные..")

**print**("(ОШИБКА) Невозмоность сборки.")

**print**("(ОШИБКА) Добавьте путь и значение к потомкам со значением 'None'")

**print**("(ОШИБКА): ", end=' ')

**else**:

**print**()

**print**("Дерево успешно построено.")

**print**(tree.format(tree.\_head))

**print**("**\n**(ОПИСАНИЕ ФОРМАТА ВЫВОДА ДЕРЕВА):")

**print**(" <дерево>::=")

**print**(" ((<левое поддерево>)<вершина>(<правоеподдерево>))")

**print**(" ((<левое поддерево>)<вершина>)")

**print**(" (<вершина>(<правое поддерево>))")

**print**(" <вершина>::=")

**print**(" <цифра><цифра>")

**print**(" <цифра>")

**print**(" <левое поддерево>::=")

**print**(" <дерево>")

**print**(" <правое поддерево>::=")

**print**(" <дерево")

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

Файл IOFILESTREEM.py:

**import** **os**

**class** **FileInstance**:

"""File instance interface"""

**def** **\_\_init\_\_**(

self,

path : str,

):

self.\_PATH = path

self.\_FILE\_DATA = [] # ['strings']

**with** open(self.\_PATH, 'r') **as** file:

**for** string **in** file.readlines():

self.\_FILE\_DATA.append(

# remove '\n'

self.\_remove\_new\_str\_symb(string)

)

**def** **\_remove\_new\_str\_symb**(self, string):

out = ''

**for** wd **in** string:

**if** wd == '**\n**':

**return** out

out = out + wd

**return** out # string hasn`t '\n'

# getters

**def** **get\_path**(self):

**return** self.\_PATH

**def** **write**(

self,

data : list,

mode = 'a',

beginstr = '',

endstr = '**\n**'

):

**if** mode == 'w': # rewrite file

self.\_FILE\_DATA = []

**try**:

**with** open(self.\_PATH, mode) **as** file:

**for** string **in** data:

file.write(

f'{beginstr}{string}{endstr}')

self.\_FILE\_DATA.append(

f'{string}')

**except** FileNotFoundError:

**print**(f"(ERROR) no such of file: '{self.\_PATH}'")

**def** **read**(self):

**print**(f"(OUTPUT) from file: {self.\_PATH}")

**for** string **in** self.\_FILE\_DATA:

**print**(string)

**def** **data**(self):

**return** self.\_FILE\_DATA

Файл TREE.py:

**class** **Node**:

**def** **\_\_init\_\_**(self, data=None, left=None, right=None) -> None:

# content

self.\_data = data

# descedants

self.\_left = left

self.\_right = right

**def** **\_\_str\_\_**(self):

out = f"head {self.\_data}"

**return** out

**def** **is\_end**(self) -> bool:

**if** **not** self.\_left **is** None **and**\

**not** self.\_right **is** None:

**return** False

**return** True

# getters

**def** **get\_data**(self):

**return** self.\_data

**def** **get\_left**(self):

**return** self.\_left

**def** **get\_right**(self):

**return** self.\_right

# setters

**def** **set\_data**(self, data):

self.\_data = data

**def** **set\_left**(self, node):

**if** isinstance(node, Node):

self.\_left = node

**else**:

**raise** **TypeError**(f"(ERROR) append to left not Node instance")

**def** **set\_right**(self, node):

**if** isinstance(node, Node):

self.\_right = node

**else**:

**raise** **TypeError**(f"(ERROR) append to right not Node instance")

**class** **BinaryTree**:

**def** **\_\_init\_\_**(self, node\_data=**0**):

self.\_head = Node(node\_data)

**def** **find\_for\_insert**(self, data, node:Node) -> Node:

**if** data <= node.get\_data():

**if** node.get\_left() **is** None:

**return** node

**else**:

**return** self.find\_for\_insert(data, node.get\_left())

**else**:

**if** node.get\_right() **is** None:

**return** node

**else**:

**return** self.find\_for\_insert(data, node.get\_right())

# append with path 0 -> left, 1 -> right chiled

**def** **append\_bypath**(self, data, path : str):

node = self.\_head

**for** to **in** path:

**if** to == '1':

**if** node.get\_right() **is** None:

node.set\_right(Node(None))

node = node.get\_right()

**elif** to == '0':

**if** node.get\_left() **is** None:

node.set\_left(Node(None))

node = node.get\_left()

node.set\_data(data)

# search None data into tree (not in end)

**def** **verified**(self, node:Node) -> bool:

**if** node **is** None:

**pass**

**elif** node.get\_left() **is** None **and** node.get\_right() **is** None:

**if** node.get\_data() **is** None:

**return** False

**return** True

**elif** **not** node.get\_left() **is** None **and** node.get\_right() **is** None:

**if** node.get\_data() **is** None:

**return** False

**return** self.verified(node=node.get\_left())

**elif** node.get\_left() **is** None **and** **not** node.get\_right() **is** None:

**if** node.get\_data() **is** None:

**return** False

**return** self.verified(node=node.get\_right())

**else**:

**if** node.get\_data() **is** None:

**return** False

**return** self.verified(node=node.get\_left()) **and** \

self.verified(node=node.get\_right())

**def** **format**(self, node : Node):

**if** node **is** None:

**return** ''

**if** node.get\_left() **is** None **and** node.get\_right() **is** None:

**return** f'{node.get\_data()}'

**if** **not** node.get\_left **is** None **and** node.get\_right() **is** None:

**return** f"({self.format(node.get\_left())}){node.get\_data()}"

**if** **not** node.get\_right() **is** None **and** node.get\_left() **is** None:

**return** f"{node.get\_data()}({self.format(node.get\_right())})"

**return** f"({self.format(node.get\_left())}){node.get\_data()}({self.format(node.get\_right())})"