|  |  |  |
| --- | --- | --- |
| 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 17 | | |
|  | | Работу выполнил  студент группы ИТ-9-2024 1 курса  Шурманов Д.А.  «8» июня 2025 г. |
| Работу проверила  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Анисимова С.И.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc200302093)

[Алгоритм решения 4](#_Toc200302094)

[1. Чтение данных из файла .txt: 4](#_Toc200302095)

[2. Сохранение данных в файл 4](#_Toc200302096)

[3. Построение бинарного дерева и его дальнейший обход 4](#_Toc200302097)

[4. Расчёт извинений: 4](#_Toc200302098)

[5. Построение минимального поддерева: 4](#_Toc200302099)

[6. Основной метод задачи: 4](#_Toc200302100)

[7. Основания функция программы: 4](#_Toc200302101)

[8. Интерфейс пользователя: 4](#_Toc200302102)

[9. Формирование отчётов: 5](#_Toc200302103)

[Отчёт №1 5](#_Toc200302104)

[Тестирование 5](#_Toc200302105)

[Вывод меню: 5](#_Toc200302106)

[Ответ программы на введенный пользователем файл: 5](#_Toc200302107)

[Ответ программы на предложенный выбор (y/n): 5](#_Toc200302108)

[Работа программы с неверным пользовательским вводом 6](#_Toc200302109)

[1. Пользователь ввел неверное имя входного файла или неправильное расширение: 6](#_Toc200302110)

[2. Неверные данные в исходном файле: 6](#_Toc200302111)

[Код программы 7](#_Toc200302112)

# Постановка задачи

Почтальон» Мудрый тритон работает почтальоном в Лукоморье. В перечень его обязанностей входит доставка корреспонденции птицам, разместившим свои гнезда на ветвях дуба (постепенно вытеснив оттуда русалку). Птицы вьют свои гнезда там, где ветка дерева разделяется на несколько более тонких веток. Лукоморье является районом с развитой инфраструктурой, поэтому все разветвления на дубе заняты гнездами птиц. За время работы тритон не только запомнил, где кто живет, но и уяснил, что если его путь проходит через чье-либо гнездо, и его обитатели не получают при этом корреспонденции, ему приходится извиняться за доставленные неудобства. Естественно, интеллигентный тритон стремится как можно меньше беспокоить обитателей дуба, поэтому он заранее планирует свой маршрут так, чтобы приносить минимальное число извинений.

Напишите программу, которая по описанию мест обитания птиц и списку корреспонденции определяет, сколько раз придется извиняться мудрому тритону. Каждое гнездо дуба обозначено уникальным номером, что соответствует номеру квартиры, который мы иногда указываем, отправляя письма родным и близким по обычной почте. Гнездо, расположенное на первой развилке ствола имеет номер один. Тритон может только ползти по веткам или стволу дуба, прыжки и перелеты полностью исключены. Никакие ветки не соприкасаются и не пересекаются.

Формат входного файла: в первой строке входного файла находится число M – количество гнезд на дубе (0<M<1000). В следующих M строках находится информация о размещении гнезд и количестве писем для каждого из возможных адресатов. Первые два числа ni и li – количество соседей гнезда с номером i и количество направленных по этому адресу писем соответственно. Далее в строке приводится ni чисел – номера гнезд, являющихся соседними с гнездом c номером i.

Формат выходного файла: в единственной строке выходного файла должно содержаться единственное число – минимальное количество извинений, которые придется приносить мудрому тритону, для того чтобы доставить все письма.

# Алгоритм решения

1. Чтение данных из файла .txt:

Программа читает данные из файла input.txt с помощью функции read\_input(), обрабатывает всю информацию.

1. Сохранение данных в файл:

После выполнения операции вычисления, функция save\_result() сохраняет изменения в новый файл.

1. Построение бинарного дерева и его дальнейший обход:

Функция build\_tree() создает бинарное дерево, инициализирует структуру данных, также задает алгоритм обхода в ширину (BFS).

1. Расчёт извинений:

Функция \_calculate\_apologies() находит максимальную глубину узлов(в дереве) и рассчитывает формулу извинений Почтальона.

1. Построение минимального поддерева:

Приватный метод \_\_build\_subtree() строит минимальное поддерево гнезд, включающее: целевые гнезда, добавление всех предков и гарантированное наличие корня.

1. Основной метод задачи:

Функция solve() собирает в себе несколько предыдущих функций, тем самым она собирает целевые гнезда, строит минимальное поддерево и вычисляет количество извинений.

1. Основания функция программы:

Функция main() получает информацию от пользователя; создает, отображает и сохраняет результат программы.

1. Интерфейс пользователя:

Функции display\_welcome(), get\_input\_filename(), display\_result() реализуют пользовательский интерфейс, который показывает пользователю поставленную задачу, запрашивает дополнительные данные и выводит результат:

Меню программы "Почтальон":

==================================================

ПРОГРАММА РАСЧЕТА МАРШРУТА ПОЧТАЛЬОНА

==================================================

Описание задачи:

Тритон должен доставить письма птицам в гнездах на дубе.

При проходе через гнездо без доставки письма, он должен извиниться.

Программа рассчитывает минимальное количество таких извинений.

Введите имя файла с данными (например, input.txt)

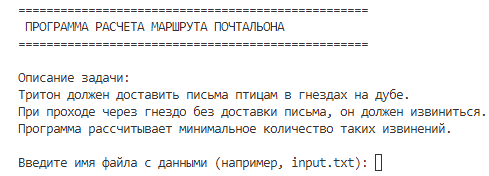
1. Формирование отчётов:

Отчёт №1: Поиск минимального количества извинений Почтальона

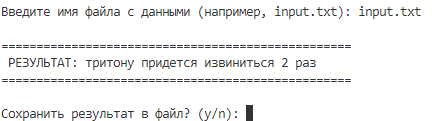
1. Запрашиваем у пользователя входные данные
2. Реализуем результат с выбором сохранения в выходной файл.

# Тестирование

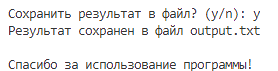
## Вывод меню:



## Ответ программы на введенный пользователем файл:

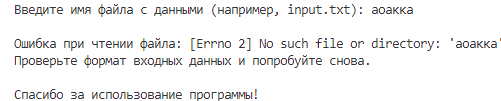
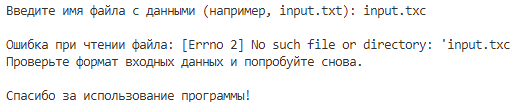


## Ответ программы на предложенный выбор (y/n):

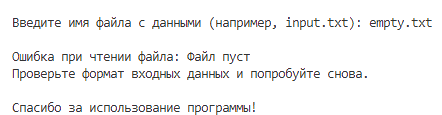
# Работа программы с неверным пользовательским вводом

# Пользователь ввел неверное имя входного файла или неправильное расширение:

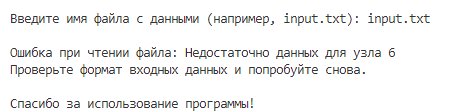
 

# Неверные данные в исходном файле:

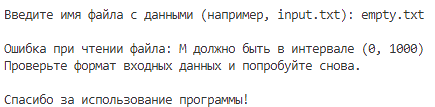
1. Пустой файл:



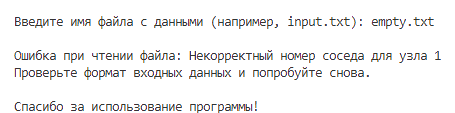
1. Количество узлов отличается с первой строкой:



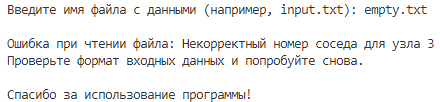
1. Значение M не входит в указанный диапазон:



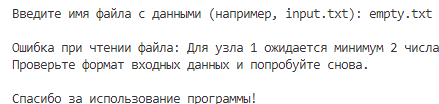
1. Несоответствие количества соседей:



1. Отрицательные числа в значениях гнёзд:



1. Несоответствие чисел в узлах:



# Код программы

class PostmanProblemSolver:

    def \_\_init\_\_(self):

        self.M = 0

        self.letters = []

        self.graph = []

        self.parent = []

        self.depth = []

    def read\_input(self, filename):

        try:

            with open(filename, 'r') as f:

                lines = [line.strip() for line in f if line.strip()]

            if not lines:

                raise ValueError("Файл пуст")

            self.M = int(lines[0])

            if not (0 < self.M < 1000):

                raise ValueError("M должно быть в интервале (0, 1000)")

            self.letters = [0] \* (self.M + 1)  # индексация с 1

            self.graph = [[] for \_ in range(self.M + 1)]

            for i in range(1, self.M + 1):

                if i >= len(lines):

                    raise ValueError(f"Недостаточно данных для узла {i}")

                data = lines[i].split()

                if len(data) < 2:

                    raise ValueError(f"Для узла {i} ожидается минимум 2 числа")

                try:

                    ni = int(data[0])  #количество соседей

                    li = int(data[1])  #количество писем

                except ValueError:

                    raise ValueError(f"Некорректные числовые данные для узла {i}")

                self.letters[i] = li

                if len(data) < 2 + ni:

                    raise ValueError(f"Для узла {i} указано недостаточно соседей")

                neighbors = []

                for j in range(2, 2 + ni):

                    try:

                        neighbor = int(data[j])

                        if not (1 <= neighbor <= self.M):

                            raise ValueError(f"Неверный номер узла {neighbor}")

                        neighbors.append(neighbor)

                    except ValueError:

                        raise ValueError(f"Некорректный номер соседа для узла {i}")

                self.graph[i] = neighbors

        except Exception as e:

            print(f"\nОшибка при чтении файла: {e}")

            print("Проверьте формат входных данных и попробуйте снова.")

            exit(1)

    def build\_tree(self):

        self.parent = [0] \* (self.M + 1)

        self.depth = [-1] \* (self.M + 1)

        visited = [False] \* (self.M + 1)

        queue = [1]

        visited[1] = True

        self.depth[1] = 0

        self.parent[1] = 0

        while queue: #обход в ширину

            node = queue.pop(0)

            for neighbor in self.graph[node]:

                if not visited[neighbor]:

                    visited[neighbor] = True

                    self.depth[neighbor] = self.depth[node] + 1

                    self.parent[neighbor] = node

                    queue.append(neighbor)

    def \_\_collect\_target\_nodes(self):

        target\_nodes = set()

        for i in range(1, self.M + 1):

            if self.letters[i] > 0:

                target\_nodes.add(i)

        return target\_nodes

    def \_\_build\_subtree(self, target\_nodes):

        subtree = set()

        for node in target\_nodes:

            current = node

            while current != 0:

                subtree.add(current)

                current = self.parent[current]

        return subtree if subtree else {1}  # Если писем нет - только корень

    def \_calculate\_apologies(self, subtree, target\_nodes):

        size = len(subtree)

        K = len(target\_nodes)

        max\_depth = 0

        if target\_nodes:

            for node in target\_nodes:

                if self.depth[node] > max\_depth:

                    max\_depth = self.depth[node]

        total\_visits = 2 \* (size - 1) - max\_depth + 1 #Формула для расчёта извинений

        return total\_visits - K

    def solve(self):

        target\_nodes = self.\_\_collect\_target\_nodes()

        subtree = self.\_\_build\_subtree(target\_nodes)

        return self.\_calculate\_apologies(subtree, target\_nodes)

def display\_welcome():

    print("\n" + "="\*50)

    print(" ПРОГРАММА РАСЧЕТА МАРШРУТА ПОЧТАЛЬОНА")

    print("="\*50)

    print("\nОписание задачи:")

    print("Тритон должен доставить письма птицам в гнездах на дубе.")

    print("При проходе через гнездо без доставки письма, он должен извиниться.")

    print("Программа рассчитывает минимальное количество таких извинений.\n")

def get\_input\_filename():

    while True:

        filename = input("Введите имя файла с данными (например, input.txt): ")

        if filename.strip():

            return filename

        print("Ошибка: имя файла не может быть пустым!")

def display\_result(result):

    print("\n" + "="\*50)

    print(f" РЕЗУЛЬТАТ: тритону придется извиниться {result} раз")

    print("="\*50 + "\n")

def save\_result(result, filename="output.txt"):

    try:

        with open(filename, 'w') as f:

            f.write(str(result))

        print(f"Результат сохранен в файл {filename}")

    except Exception as e:

        print(f"Ошибка при сохранении результата: {e}")

def main():

    display\_welcome()

    filename = get\_input\_filename()

    try:

        solver = PostmanProblemSolver()

        solver.read\_input(filename)

        solver.build\_tree()

        result = solver.solve()

        display\_result(result)

        save = input("Сохранить результат в файл? (y/n): ").lower()

        if save == 'y':

            save\_result(result)

    except Exception as e:

        print(f"\nПроизошла ошибка: {e}")

        print("Программа завершена с ошибкой.")

    finally:

        print("\nСпасибо за использование программы!")

main()