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

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 с 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.

2. **Сохранение данных в файл**:

После выполнения операции вычисления, функция save_result() сохраняет изменения в новый файл.

3. **Построение бинарного дерева и его дальнейший обход**:

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

4. **Расчёт извинений:**

Функция _calculate_apologies() находит максимальную глубину узлов(в дереве) и рассчитывает формулу извинений Почтальона.

5. **Построение минимального поддерева:**

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

6. **Основной метод задачи:**

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

7. **Основания функция программы:**

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

8. **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)

9. 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:

```
================================================
 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):
```

Ответ программы на введенный пользователем файл:

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

================================================
 РЕЗУЛЬТАТ: тритону придется извиниться 2 раз
================================================

Сохранить результат в файл? (y/n):
```

Ответ программы на предложенный выбор (y/n):

```
Сохранить результат в файл? (y/n): y
Результат сохранен в файл output.txt

Спасибо за использование программы!
```

```
Сохранить результат в файл? (y/n): n

Спасибо за использование программы!
```

Работа программы с неверным пользовательским вводом

1. **Пользователь ввел неверное имя входного файла или неправильное расширение:**

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

Ошибка при чтении файла: [Errno 2] No such file or directory: 'аоакка'
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

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

Ошибка при чтении файла: [Errno 2] No such file or directory: 'input.txc
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

2. **Неверные данные в исходном файле:**

    1. Пустой файл:

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

Ошибка при чтении файла: Файл пуст
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

    2. Количество узлов отличается с первой строкой:

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

Ошибка при чтении файла: Недостаточно данных для узла 6
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

    3. Значение M не входит в указанный диапазон:

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

Ошибка при чтении файла: M должно быть в интервале (0, 1000)
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

    4. Несоответствие количества соседей:

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

Ошибка при чтении файла: Некорректный номер соседа для узла 1
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

5. Отрицательные числа в значениях гнёзд:

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

Ошибка при чтении файла: Некорректный номер соседа для узла 3
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

6. Несоответствие чисел в узлах:

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

Ошибка при чтении файла: Для узла 1 ожидается минимум 2 числа
Проверьте формат входных данных и попробуйте снова.

Спасибо за использование программы!
```

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()
```