|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Цифровые технологии обработки данных»

**ОТЧЕТ   
о выполнении лабораторной работы №2**

**«Алгоритмы на графах»**

**по дисциплине   
«Алгоритмы и структуры данных»**

**Вариант № 99**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(фио студента)*

Проверил: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2022 г.

**Вариант № 99.**

**Алгоритм:** определить минимальное число красок, которыми можно раскрасить   
граф и вывести пример такой раскраски.

**Способ представления графа:** Список смежности.

**Теория о Графах.**

Граф — это математический объект, который состоит из точек и линий, которые их соединяют. Точки называют вершинами графа, а линии — ребрами. Граф, ребра которого имеют направления, называется ориентированным, если же ребра графа не имеют направления, то такой граф называется неориентированным.



Рисунок 1. – Пример ориентированного графа.

Матрица смежности — это вид представления графа в виде матрицы, когда пересечение столбцов и строк задаёт дуги. Используя матрицу смежности, можно задать вес дуг и ориентацию. Каждая строка и столбец матрицы соответствуют вершинам, номер строки соответствует вершине, из которой выходит дуга, а номер столбца - в какую входит дуга. Пример матрицы смежности графа, изображенного на рисунке 1, представлен на рисунке 2.



Рисунок 2. – Пример матрицы смежности.

**Список смежности** — одна из форм представления графа. Номер строки в списке означает номер вершины, из которой выходят ребра, а номера в строке означают номера вершин смежных с текущей вершиной.

Например, список смежности для графа из рисунка 1 может выглядеть так:

1. B G
2. C D
3. D E
4. A
5. B F
6. A B
7. A

Для поиска хроматического числа и раскраски графа отсортируем список вершин по их степени с сохранением номеров.

Далее внешним циклом будем проходить по непокрашенным вершинам и, если такая вершина существует, то создавать новый цвет.

Внутренним циклом по непокрашенным вершинам будем проверять не имеет ли эта вершина связей с вершинами, покрашенными этим цветом, если связей нет, то красим вершину и добавляем ее в список покрашенных этим цветом.

После завершения цикла, добавляем в список покрашенных вершин список покрашенных вершин этим цветом.

На выходе в переменной текущего цвета мы будем иметь хроматическое число нашего графа, а в списке покрашенных вершин будет пример его раскраски.

**Листинг программы.**

import json

from typing import Optional, Tuple, List, Dict

class Vertex:

    \_index: int

    label: str

    def \_\_init\_\_(self, index: int, label: Optional[str] = ""):

        self.\_index = index

        self.label = label

    @property

    def index(self) -> int:

        return self.\_index

    def \_\_repr\_\_(self):

        return f"Vertex[i: {self.\_index}, label: {self.label}]"

class Edge:

    \_vertices: Tuple[int, int]

    len: int

    def \_\_init\_\_(self, v1: int = None, v2: int = None, len: int = 1):

        self.\_vertices = (v1, v2)

        self.len = len

    @property

    def vertices(self) -> Tuple[int, int]:

        return self.\_vertices

    def \_\_repr\_\_(self):

        return f"Edge[{self.\_vertices[0]} -> {self.\_vertices[1]}, {self.len}]"

class Graph:

    \_vertices: List[Vertex]

    \_edges: List[Edge]

    def \_\_init\_\_(self) -> None:

        self.\_vertices = []

        self.\_edges = []

    def \_init\_vertices(self, dim: int):

        self.\_vertices = []

        for i in range(dim):

            self.\_vertices.append(Vertex(i))

    def \_load\_adj\_matrix(self, data: List[List[int]], dim: int) -> None:

        self.\_init\_vertices(dim)

        for i, vertex in enumerate(data):

            for j, edge\_len in enumerate(vertex):

                if edge\_len > 0:

                    self.add\_e(i, j, edge\_len)

    def \_load\_adj\_list(self, data: List[List[int]], dim: int) -> None:

        self.\_init\_vertices(dim)

        for i, adj in enumerate(data):

            for j in adj:

                self.add\_e(i, j, 1)

    def \_load\_edge\_list(self, data: List[List[int]], dim: int) -> None:

        self.\_init\_vertices(dim)

        for edge in data:

            if len(edge) == 3:

                v1, v2, edge\_len = edge

            else:

                v1, v2, edge\_len = \*edge, 1

            self.add\_e(v1, v2, edge\_len)

    def \_load\_inc\_matrix(self, data: List[List[int]], dim: int) -> None:

        self.\_init\_vertices(dim)

        transposed\_data = list(zip(\*data))  # транспонирование

        for edge in transposed\_data:

            v1 = None

            v2 = None

            v1\_v2\_len = 0

            v2\_v1\_len = 0

            for j in range(dim):

                if edge[j] != 0 and v1 is None:

                    v1 = j

                    v1\_v2\_len = edge[j]

                elif edge[j] != 0 and v2 is None:

                    v2 = j

                    v2\_v1\_len = edge[j]

                    break

            if v1\_v2\_len > 0:

                self.add\_e(v1, v2, v1\_v2\_len)

            if v2\_v1\_len > 0:

                self.add\_e(v2, v1, v2\_v1\_len)

    def \_load\_labels(self, labels: Dict[str, str]) -> None:

        for index, label in labels.items():

            index = int(index)

            self.\_vertices[index].label = label

    def load(self, filename: str) -> None:

        with open(filename, "r", encoding="utf8") as f:

            graph\_json: dict = json.load(f)

        \_format = graph\_json.get("format", None)

        if \_format is None:

            raise Exception("Не указан формат для задания графа! Смотри документацию!")

        \_dim: Optional[str] = graph\_json.get("dim", None)

        if \_dim is None or not isinstance(\_dim, int):

            raise Exception("Не укаказан/верный размер графа! Смотри документацию!")

        \_data: Optional[List[List[int]]] = graph\_json.get("data", None)

        if \_data is None:

            raise Exception(

                "Не заданны данные для построения графа! Смотри документацию!"

            )

        \_labels: Optional[Dict[str, str]] = graph\_json.get("labels", None)

        if \_format == "adj\_matrix":

            self.\_load\_adj\_matrix(\_data, \_dim)

        elif \_format == "adj\_list":

            self.\_load\_adj\_list(\_data, \_dim)

        elif \_format == "edge\_list":

            self.\_load\_edge\_list(\_data, \_dim)

        elif \_format == "inc\_matrix":

            self.\_load\_inc\_matrix(\_data, \_dim)

        else:

            raise Exception("Неизвестный формат задания графа! Смотри документацию!")

        if \_labels:

            self.\_load\_labels(\_labels)

    def first(self, v: int) -> None | int:

        for edge in self.\_edges:

            if edge.vertices[0] == v:

                return edge.vertices[1]

        return None

    def next(self, v: int, i: int) -> None | int:

        state = 0

        next\_index = None

        for edge in self.\_edges:

            if edge.vertices[0] != v:

                if state == 0:

                    continue

                if state == 1:

                    break

            else:

                state = 1

                if edge.vertices[1] > i:

                    next\_index = edge.vertices[1]

                    break

        return next\_index

    def vertex(self, v: int, i: int) -> None | int:

        state = 0

        vertices\_set: List[int] = []

        for edge in self.\_edges:

            if edge.vertices[0] != v:

                if state == 0:

                    continue

                if state == 1:

                    break

            else:

                state = 1

                vertices\_set.append(edge.vertices[1])

        if i < len(vertices\_set):

            return vertices\_set[i]

        else:

            return None

    def add\_v(self, index: int, label: str = "") -> None:

        for vertex in self.\_vertices:

            if vertex.index == index:

                raise Exception(f"Вершина с таким индексом уже существует! ({index})")

        self.\_vertices.append(Vertex(index, label))

    def add\_e(self, v1: int, v2: int, edge\_len: int = 1) -> None:

        found\_v1 = False

        found\_v2 = False

        for vertex in self.\_vertices:

            if vertex.index == v1:

                found\_v1 = True

            elif vertex.index == v2:

                found\_v2 = True

        if not (found\_v1 and found\_v2) or v1 == v2:

            raise Exception(f"Невозможная пара индексов вершин! ({v1}, {v2})")

        else:

            self.\_edges.append(Edge(v1, v2, edge\_len))

    def del\_v(self, index: int) -> None:

        for vertex in self.\_vertices:

            if vertex.index == index:

                break

        else:

            raise Exception(f"Вершина с таким индексом не существует! ({index})")

        edges\_to\_remove = []

        for edge in self.\_edges:

            if edge.vertices[0] == index or edge.vertices[1] == index:

                edges\_to\_remove.append(edge)

        for edge in edges\_to\_remove:

            self.\_edges.remove(edge)

    def del\_e(self, v1: int, v2: int) -> None:

        for edge in self.\_edges:

            if edge.vertices[0] == v1 and edge.vertices[1] == v2:

                break

        else:

            raise Exception(f"Ребра с таким набором вершин не существует! ({v1}, {v2})")

    def to\_adj\_matrix(self) -> List[List[int]]:

        adj\_matrix = [

            [0 for \_ in range(len(self.\_vertices))] for \_ in range(len(self.\_vertices))

        ]

        for edge in self.\_edges:

            adj\_matrix[edge.\_vertices[0]][edge.\_vertices[1]] = edge.len

        return adj\_matrix

    def as\_dict(self) -> Dict:

        return {"Vertices": self.\_vertices, "Edges": self.\_edges}

    @property

    def vertices(self) -> List[Vertex]:

        return self.\_vertices

    @property

    def edges(self) -> List[Edge]:

        return self.\_edges

COLORS = {1: "RED", 2: "GREEN", 3: "BLUE", 4: "YELLOW", 5: "BLACK", 6: "PURPLE"}

def colorize(matrix: List[List[int]]) -> Tuple[int, List[Tuple[int, int]]]:

    v\_power = [(i, sum(row)) for i, row in enumerate(matrix)]

    v\_power.sort(key=lambda tup: -tup[1])

    v\_color = []

    colored = set()

    color = 0

    for v1, \_ in v\_power:

        if v1 in colored:

            continue

        color += 1

        colored\_on\_step = set()

        for v2, \_ in v\_power:

            if v2 in colored:

                continue

            need\_skip = False

            for v3 in colored\_on\_step:

                if matrix[v2][v3]:

                    need\_skip = True

                    break

            if need\_skip:

                continue

            colored\_on\_step.add(v2)

            v\_color.append((v2, color))

        colored.update(colored\_on\_step)

    v\_color.sort(key=lambda tup: tup[0])

    return color, v\_color

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

    graph = Graph()

    graph.load("graph\_task99.json")

    colors, node\_color = colorize(graph.to\_adj\_matrix())

    print(f"Хроматическое число графа: {colors}")

    print("Пример раскраски:")

    for node, color in node\_color:

        print(f"{node}: {COLORS[color]}")

**Скриншот работы программы:**

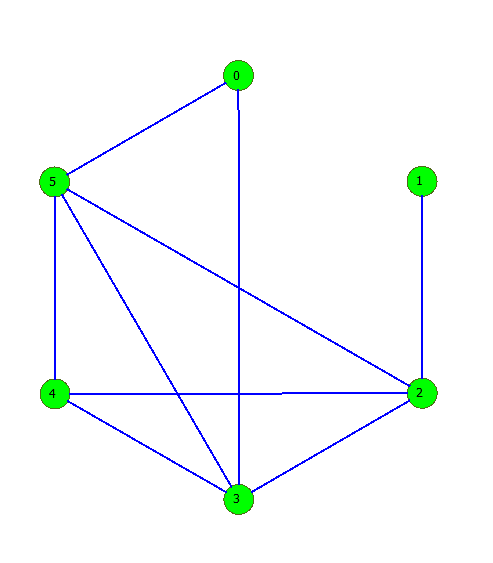


Рисунок 3. – Пример неориентированного ненагруженного графа.

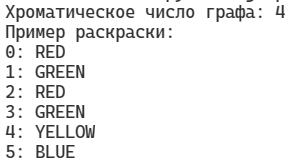
****

Рисунок 4. – Пример работы программы.

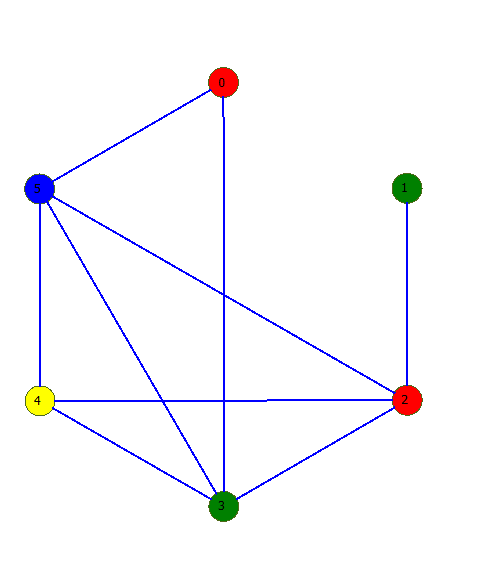


Рисунок 5. Пример графа после раскраски

**Выводы.**

В результате выполнения данной работы были изучены свойства графа, способы его задания и работа с ним. Был реализован класс Graph предоставляющий базовые операции для работы с графами. Также был реализован алгоритм для поиска хроматического числа графа и его раскраски.

**Литература:**

1. Алгоритмы: построение и анализ. Кормен Т.Х., Лейзерсон Ч.И.
2. Донец, Г. А. & Шор, Н. З. (1982), Алгебраический подход к проблеме раскраски плоских графов, Наукова думка.
3. Beigel, R. & Eppstein, D. (2005), 3-coloring in time O(1.3289n), Journal of Algorithms.