|  |
| --- |
| 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 №3**

**«Алгоритмы на графах»**

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

**Вариант № 12**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_

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

*(фио студента)*

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

Москва 2022 г.

**Задание на лабораторную работу № 3.**

Реализовать в виде программы абстрактный тип данных «Граф» согласно варианту (Номер варианта – две последние цифры шифра студента, номера зачетной книжки) с учетом заданного представления графа. Операторы (операции) АТД «Граф» функционально должны выполнять

следующие операции (названия операций – примерные):

1. FIRST(v) - возвращает индекс первой вершины, смежной с вершиной v. Если вершина v не имеет смежных вершин, то возвращается "нулевая" вершина A.

2. NEXT(v, i)- возвращает индекс вершины, смежной с вершиной v, следующий за индексом i. Если i — это индекс последней вершины, смежной с вершиной v, то возвращается A.

3. VERTEX(v, i) - возвращает вершину с индексом i из множества вершин, смежных с v.

4. ADD\_V(<имя>,<метка, mark>) - добавить УЗЕЛ

5. ADD\_Е(v, w, c) - добавить ДУГУ (здесь c — вес, цена дуги (v,w))

6. DEL\_V(<имя>) - удалить УЗЕЛ

7. DEL\_Е(v, w) – удалить ДУГУ

8. EDIT\_V(<имя>, <новое значение метки или маркировки>) - изменить метку (маркировку) УЗЛА

EDIT\_Е(v, w, <новый вес дуги>) - изменить вес ДУГИ

**Вариант № 12**

**Алгоритм:** Орграф G' = (V, Е') называется минимальным эквивалентным   
орграфом для орграфа G = (V, Е), если Е' — наименьшее   
подмножество множества Е (E’ ⊆ E) такое что транзитивные   
замыкания обоих орграфов G и G' совпадают (причем если граф G   
ацикличен, то для него существует только один минимальный   
эквивалентный орграф). Написать программу нахождения   
минимального эквивалентного орграфа.

**Способ представления графа:** список дуг.

**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.

Цикл графа – некоторое число вершин, соединенных замкнутой цепью. Так, например, на графе, изображенном на рисунке 1, циклами будут являться следующие соединенные вершины: A->B->C->D->A, A->B->C->E->F->A, A->B->D->A, A->G->D->A, B->C->E->B и B->C->E->F->B.

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.

Для поиска всех циклов в ориентированном графе воспользуемся поиском в глубину. Применим его к каждой вершине на графе и уберем из списка пути, конец которых не совпадает с началом.

**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):

        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):

        self.\_edges = []

        self.\_init\_vertices(dim)

        for i, vertex in enumerate(data):

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

                if edge\_len > 0:

                    self.\_edges.append(Edge(i, j, edge\_len))

    def \_load\_adj\_list(self, data: List[List[int]], dim: int):

        self.\_edges = []

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

    def \_load\_inc\_matrix(self, data: List[List[int]], dim: int):

        self.\_edges = []

        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.\_edges.append(Edge(v1, v2, v1\_v2\_len))

            if v2\_v1\_len > 0:

                self.\_edges.append(Edge(v2, v1, v2\_v1\_len))

    def \_load\_labels(self, labels: Dict[str, str]):

        for index, label in labels.items():

            index = int(index)

            self.\_vertices[index].label = label

    def load(self, filename: str):

        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 == "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):

        for edge in self.\_edges:

            if edge.vertices[0] == v:

                return edge.vertices[1]

        return None

    def next(self, v: int, i: 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):

        state = 0

        vertices\_set = []

        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 = ""):

        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):

        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):

        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):

        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):

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

    @property

    def vertices(self):

        return self.\_vertices

    @property

    def edges(self):

        return self.\_edges

def find\_minimal\_equivalent(adj\_matrix: List[List[int]]) -> List[List[int]]:

    """

    Функция поиска минимального эквивалентного графа

    """

    dim = len(adj\_matrix)

    for k in range(dim):

        for i in range(dim):

            for j in range(dim):

                if (

                    adj\_matrix[i][j] != 0

                    and adj\_matrix[i][k] != 0

                    and adj\_matrix[k][j] != 0

                ):

                    adj\_matrix[i][j] = 0

    return adj\_matrix

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

    graph = Graph()

    graph.load("graph\_task12.json")

    adj\_matrix = graph.to\_adj\_matrix()

    print(

        "Матрица смежности:\n"

        + "\n".join([" ".join([str(j) for j in i]) for i in adj\_matrix])

    )

    adj\_matrix = find\_minimal\_equivalent(graph.to\_adj\_matrix())

    print(

        "Матрица смежности минимального эквивалентного графа:\n"

        + "\n".join([" ".join([str(j) for j in i]) for i in adj\_matrix])

    )

**Скриншот работы программы:**

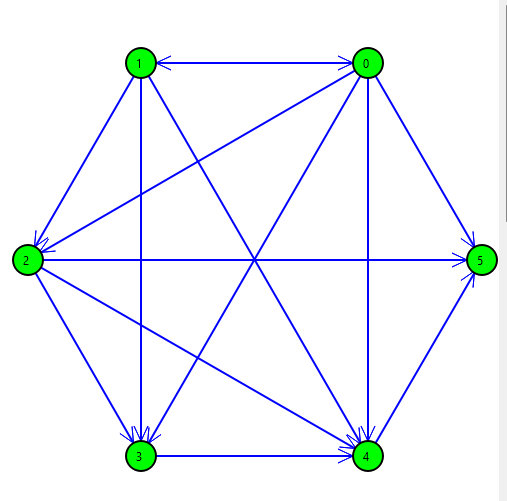


Рисунок 3. – Пример ориентированного ненагруженного графа.

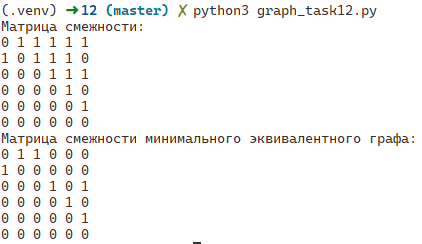
****

Рисунок 4. – Пример работы программы.

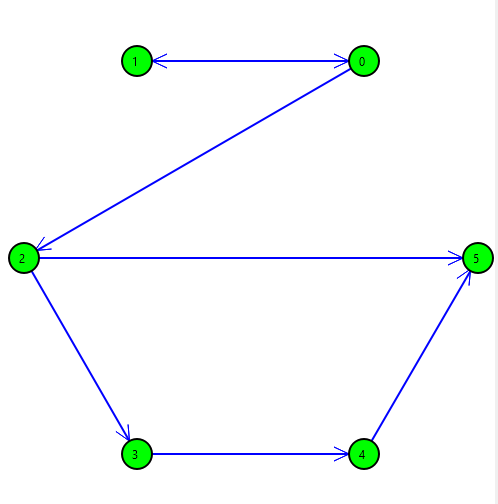


Рисунок 5. - Пример результирующего минимального эквивалентного графа

**Выводы.**

В результате выполнения данной работы был реализован класс Graph способный выполнять базовые операции на графе. Так же был реализован алгоритм поиска минимального эквивалентного графа и получен результат работы программы.

**Литература:**

1. Алгоритмы: построение и анализ. Кормен Т.Х., Лейзерсон Ч.И.

2. <https://en.wikipedia.org/wiki/Cycle_(graph_theory)>

3. <https://en.wikipedia.org/wiki/Depth-first_search>