|  |
| --- |
| 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 № 80**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_

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

*(фио студента)*

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

Москва 2022 г.

**Вариант № 80.**

**Алгоритм:** определить k-связанность заданного неориентированного графа и  
вывести полученное число k на экран. (Граф называется k-связным,  
если между любой парой вершин v и w существует не менее k  
разных путей, таких, что, за исключением вершин v и w, ни одна из  
вершин, входящих в один путь, не входит ни в какой другой из этих  
путей)

**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

Задача поиска числа к-связности графа сводится к задаче поиска максимального потока в графе. Так-как наш граф является неориентированным и ненагруженным, то установим пропускную способность для каждого ребра равную 1/1 и найдем поток между каждой парой вершин. После этого находим минимальный из максимальных потоков. Это и будет наше число k. Для поиска максимального потока воспользуемся алгоритмом Форда-Фалкерсона.

**Пример работы алгоритма Форда-Фалкерсона.**

* Допустим наш алгоритм нашел следующий путь из вершинывнашей *остаточной сети*, которая на момент начала, совпадает с исходным графом.

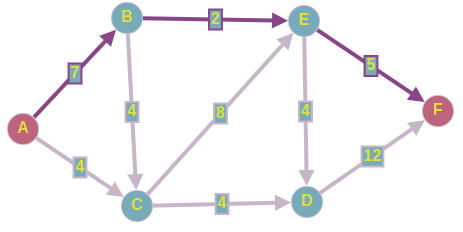


Рисунок 3. Начальная остаточная сеть

- Сколько потока можем провести по этому пути?  
- Больше 2 ед. мы никак не сможем пустить, пропускаем наш поток по этим рёбрам из *истока* в *сток.*

Получаем следующее:

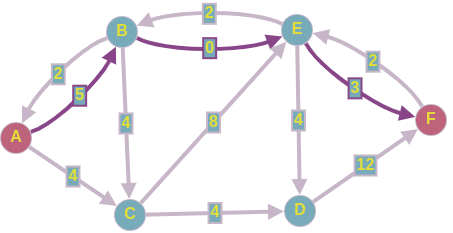


Рисунок 4. Остаточная сеть после 1-ой итерации

Рёбра с нулевым весом можно удалять. Таким образом, на первой итерации мы смогли увеличить максимальный поток на 2 ед.

Теперь дело за малым, остается всего лишь итерироваться до тех пор, пока существует путь из A в F.

* Допустим, на *2-ой итерации* мы нашли такой путь в нашей *остаточной сети*:

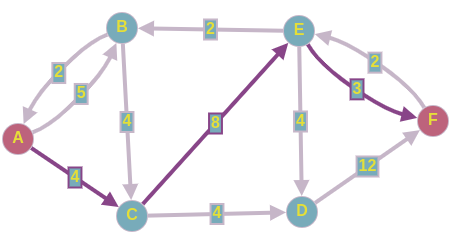


Рисунок 5. Остаточная сеть после 1-ой итерации

Пропускаем 3 ед**.** потока по этому пути, и перестраиваем *остаточную сеть.*   
Получаем следующее:

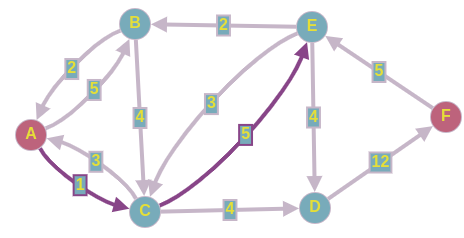


Рисунок 6. Остаточная сеть после 2-ой итерации

На 3-ей итерации нашли такой путь в нашей модифицированной *остаточной сети:*

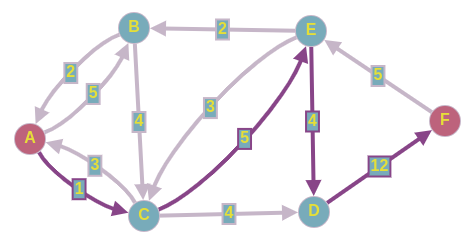


Рисунок 7. Остаточная сеть после 2-ой итерации.

Пускаем 1 ед.потока по этому пути и перестраиваем нашу *остаточную сеть.*  
Получим следующую картину:

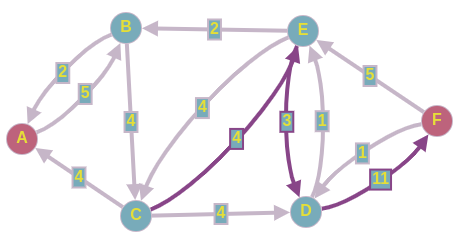


Рисунок 8. Остаточная сеть после 3-ей итерации

На 4-ой итерации находим следующий путь в *остаточной сети:*

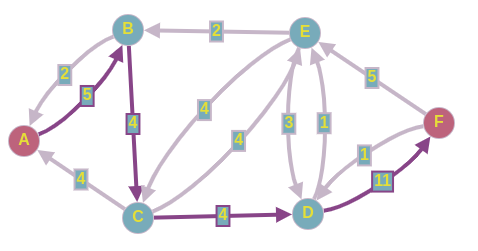


Рисунок 9. Остаточная сеть после 3-ей итерации

Пускаем 4 ед.потока по этому пути и перестраиваем нашу *остаточную сеть.*  
Получим следующую картину:

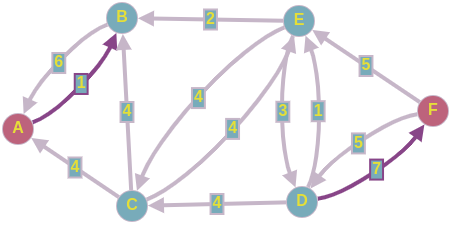


Рисунок 10. Итоговая остаточная сеть

На этом этапе наш алгоритм прекратит выполнение из-за того, что пути из *истока* в *сток* не существует. И ответом к поставленной задаче будет служить сумма потоков всех найденных *увеличивающихся путей.****Ответ:*** 10 ед.

**Листинг программы.**

import json

from typing import Dict, List, Optional, Tuple

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

def find\_graph\_k(matrix: List[List[int]]) -> int:

    graph = []

    dim = len(matrix)

    def bfs(s: int, t: int, parent: List[int]) -> bool:

        visited = [False] \* dim

        queue = []

        queue.append(s)

        visited[s] = True

        while queue:

            u = queue.pop(0)

            for ind, val in enumerate(graph[u]):

                if visited[ind] == False and val > 0:

                    queue.append(ind)

                    visited[ind] = True

                    parent[ind] = u

                    if ind == t:

                        return True

        return False

    def ford\_fulkerson(source: int, sink: int) -> int:

        parent = [-1] \* dim

        max\_flow = 0

        while bfs(source, sink, parent):

            path\_flow = float("Inf")

            s = sink

            while s != source:

                path\_flow = min(path\_flow, graph[parent[s]][s])

                s = parent[s]

            max\_flow += path\_flow

            v = sink

            while v != source:

                u = parent[v]

                graph[u][v] -= path\_flow

                graph[v][u] += path\_flow

                v = parent[v]

        return max\_flow

    k = None

    for i in range(len(matrix)):

        for j in range(len(matrix)):

            if i == j:

                continue

            graph = [row[:] for row in matrix]

            local\_k = ford\_fulkerson(i, j)

            if k is None or local\_k < k:

                k = local\_k

    return k

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

    graph = Graph()

    graph.load("graph\_task80.json")

    k = find\_graph\_k(graph.to\_adj\_matrix())

    print(f"K-связность графа равна: {k}")

**Скриншот работы программы:**



Рисунок 12. Пример работы программы

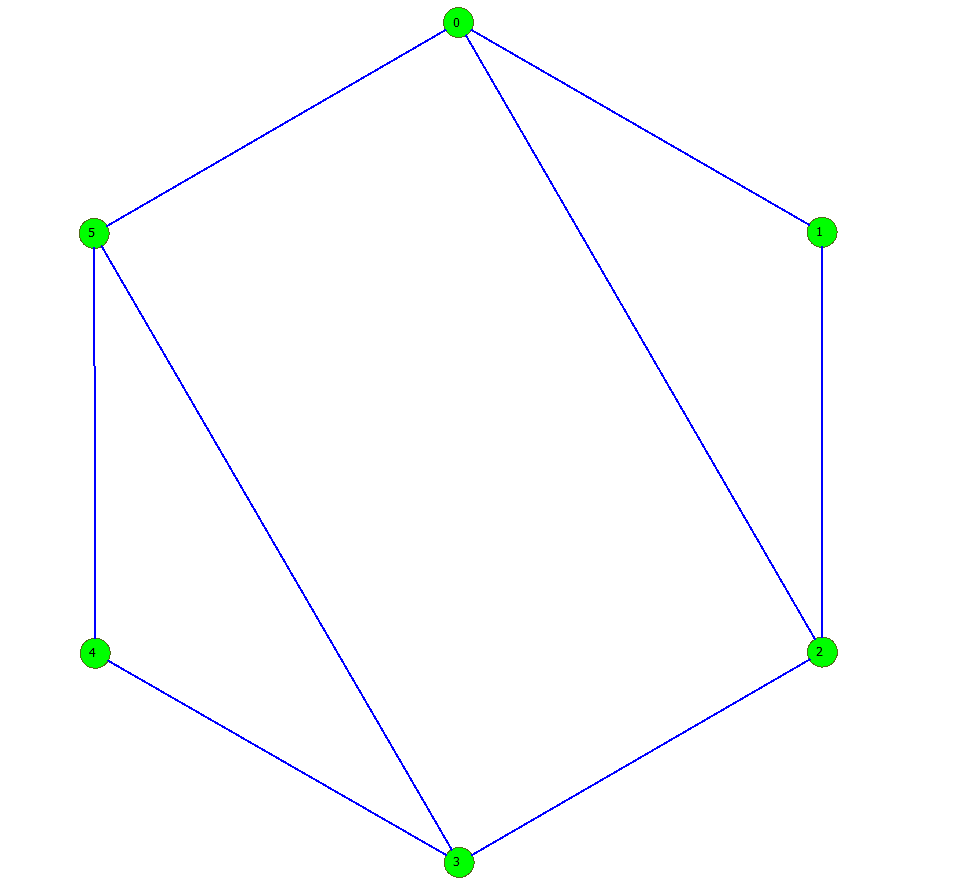


Рисунок 11. Пример ненагруженного неориентированного графа

**Выводы.**

В результате выполнения данной работы был реализован класс Graph предоставляющий интерфейс для удобного взаимодействия с графом в языке python. Были реализованы базовые операции над ним. Также был реализован алгоритм Форда-Фалкерсона для поиска максимального потока в ориентированном нагруженном графе. На основе этого алгоритма был реализован поиск числа k-связности графа.

**Литература:**

1. Алгоритмы: построение и анализ. Кормен Т.Х., Лейзерсон Ч.И.
2. <https://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm>
3. <https://en.wikipedia.org/wiki/K-edge-connected_graph>