|  |
| --- |
| 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 № 1**

Выполнил: студент 2 курса

группы

шифр

*(фио студента)*

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

Москва 2020 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 балл из 5):

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 № 1.**

**Алгоритм:** **Дана матрица весов дуг. Определить и вывести все циклы в орграфе, заданной длины х (вводится с клавиатуры)**

**Способ представления графа: Матрица 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.

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.



Рисунок 2. – Пример матрицы смежности.

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.

**Листинг программы.**

import itertools

import networkx as nx

import matplotlib.pyplot as plt

import time

#Класс TextColors - класс с escape - последовательностями ANSI

#для цветового оформления вывода

class TextColors:

    HEADER = '\033[95m'

    OKBLUE = '\033[94m'

    OKCYAN = '\033[96m'

    OKGREEN = '\033[92m'

    WARNING = '\033[93m'

    FAIL = '\033[91m'

    ENDC = '\033[0m'

    BOLD = '\033[1m'

    UNDERLINE = '\033[4m'

#Класс Graph - Граф

class Graph:

    def \_\_init\_\_(self):

        self.length= 0

        self.vertex\_names = {}

        self.all\_cycles\_buff=[]

        self.visual\_buff = []

        self.matrix = [[0 for x in range(self.length)] for y in range(self.length)]

        pass

    #Визуализация графа

    def visualize(self):

        G = nx.DiGraph()

        G.add\_edges\_from(self.visual\_buff)

        nx.draw\_networkx(G)

        plt.show()

    #Добавление вершины с индеком в граф

    def add\_vertex\_with\_index(self,v\_name,v\_index):

        if (not v\_name in self.vertex\_names) and (not v\_index in self.vertex\_names.values()):

            self.vertex\_names[v\_name]=v\_index

            if v\_index>=self.length:

                self.length=v\_index+1

                new\_matrix = [[0 for x in range(self.length)] for y in range(self.length)]

                for i in range(len(self.matrix)):

                    for j in range(len(self.matrix)):

                        new\_matrix[i][j]=self.matrix[i][j]

                self.matrix = new\_matrix

        elif v\_name in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} ВЕРШИНА С НАЗВАНИЕМ '{v\_name}' УЖЕ ИСПОЛЬЗУЕТСЯ!")

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} ИНДЕКС '{v\_index}' УЖЕ ИСПОЛЬЗУЕТСЯ!")

    #Добавление вершины в граф (индекс выбирается автоматически)

    def add\_vertex(self,v\_name):

        if not v\_name in self.vertex\_names:

            self.vertex\_names[v\_name]=self.length

            self.length+=1

            new\_matrix = [[0 for x in range(self.length)] for y in range(self.length)]

            for i in range(len(self.matrix)):

                for j in range(len(self.matrix)):

                    new\_matrix[i][j]=self.matrix[i][j]

            self.matrix = new\_matrix

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} ВЕРШИНА С НАЗВАНИЕМ '{v\_name}' УЖЕ ИСПОЛЬЗУЕТСЯ!")

    #Удаление вершины из графа

    def del\_vertex(self,v\_name):

        if v\_name in self.vertex\_names:

            v\_index = self.vertex\_names[v\_name]

            self.vertex\_names.pop(v\_name)

            #Удаление всех дуг, связанных с удаленной вершиной

            for i in range(self.length):

                for j in range(self.length):

                   if i==v\_index or j==v\_index:

                       if self.matrix[i][j]==1:

                           self.matrix[i][j]=0

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v\_name}'!")

    ##обавление ребра в граф

    def add\_edge(self,v1\_name,v2\_name):

        if v1\_name and v2\_name in self.vertex\_names:

            v1\_index=self.vertex\_names[v1\_name]

            v2\_index=self.vertex\_names[v2\_name]

            self.matrix[v1\_index][v2\_index] = 1

            self.visual\_buff.append([v1\_name,v2\_name])

        elif v1\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v1\_name}'!")

        elif v2\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v2\_name}'!")

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИН С НАЗВАНИЯМИ '{v1\_name}' и '{v2\_name}'!")

    #Удаление ребра из графа

    def del\_edge(self,v1\_name,v2\_name):

        if v1\_name and v2\_name in self.vertex\_names:

            v1\_index=self.vertex\_names[v1\_name]

            v2\_index=self.vertex\_names[v2\_name]

            self.matrix[v1\_index][v2\_index] = 0

        elif v1\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v1\_name}'!")

        elif v2\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v2\_name}'!")

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИН С НАЗВАНИЯМИ '{v1\_name}' и '{v2\_name}'!")

    #Получение первого индекса, смежного с выранным

    def get\_first\_vertex(self,v\_index):

        for j in range(self.length):

            if self.matrix[v\_index][j]==1:

                return j

    #Получение первого индекса, смежного с вершиной v\_index, но после next\_v\_index

    def get\_next\_vert(self,v\_index, next\_v\_index):

         for j in range(self.length):

            if self.matrix[v\_index][j]==1 and j>next\_v\_index:

                return j

    #Изменить индекс вершины

    def change\_vertex\_index(self,v\_name, new\_v\_index):

        if (v\_name in self.vertex\_names) and (not new\_v\_index in self.vertex\_names.values()) :

            old\_v\_index = self.vertex\_names[v\_name]

            self.vertex\_names[v\_name]=new\_v\_index

            if new\_v\_index>=self.length:

                self.length=new\_v\_index+1

                new\_matrix = [[0 for x in range(self.length)] for y in range(self.length)]

                for i in range(len(self.matrix)):

                    for j in range(len(self.matrix)):

                        new\_matrix[i][j]=self.matrix[i][j]

                self.matrix = new\_matrix

            for i in range(len(self.matrix)):

                    for j in range(len(self.matrix)):

                        if self.matrix[old\_v\_index][j]==1:

                            self.matrix[old\_v\_index][j]=0

                            self.matrix[new\_v\_index][j]=1

                        if self.matrix[j][old\_v\_index]==1:

                            self.matrix[j][old\_v\_index]=0

                            self.matrix[j][new\_v\_index]=1

        elif not v\_name in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{new\_v\_index}'!")

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} ИНДЕКС '{new\_v\_index}' УЖЕ ИСПОЛЬЗУЕТСЯ!")

    #Вывести матрицу смежности

    def print\_matrix(self):

        for line in self.matrix:

            print(line)

    #Получение последних элементов маршрутов буфера (для работы алгоритма)

    def last\_in\_buff(self,buff):

        last\_el\_buff = []

        try:

            for i in range(len(buff)):

                z = buff[i][len(buff[i])-1][1] #[len(buff[i][len(buff[i])-1]

                last\_el\_buff.append(z)

        except: last\_el\_buff.append(buff[0][1])

        last\_el\_buff = list(set(last\_el\_buff))

        return last\_el\_buff

    #Алгоритм нахождения циклов (проходит по всем маршрутам и хранит их)

    def search\_cycles\_algorithm(self,start\_i,start\_j):

        buff=[[(start\_i, start\_j)]]

        # start\_time = time.time()

        for c in range(self.length):

            for i in range(self.length):

                last\_el\_buff = self.last\_in\_buff(buff)

                for j in range(self.length):

                    for k in range(len(last\_el\_buff)):

                        if self.matrix[i][j]==1 and (i==last\_el\_buff[k]):

                            for l in range(len(buff)):

                                if (buff[l][len(buff[l])-1][1]==last\_el\_buff[k]):

                                    temp=[]

                                    for elem in buff[l]:

                                            temp.append(elem)

                                    temp.append((i,j))

                                    buff.append(temp)

            buff.sort()

            buff = list(num for num,\_ in itertools.groupby(buff))

        final\_buff = []

        for i in range(len(buff)):

            if buff[i][0][0]==buff[i][len(buff[i])-1][1]:

                if len(set(buff[i])) == len(buff[i]):

                    final\_buff.append(buff[i])

        temp = []

        for i in range(len(final\_buff)):

            tmp\_dct={}

            for j in range(len(final\_buff[i])):

                for k in range(2):

                    if not final\_buff[i][j][k] in tmp\_dct:

                        tmp\_dct[final\_buff[i][j][k]]=1

                    else:

                        tmp\_dct[final\_buff[i][j][k]]+=1

            for key in tmp\_dct:

                if tmp\_dct[key]>2:

                    temp.append(i)

                    break

        for i in range(len(temp)):

            final\_buff.pop(temp[i])

            for j in range(len(temp)):

                temp[j]-=1

        # print(f"\n{TextColors.OKGREEN}///DEBUG///{TextColors.ENDC} FOR EDGE {start\_i,start\_j}:")

        # for i in final\_buff:

        #     print(i)

        # print(f"{TextColors.OKGREEN}///DEBUG///{TextColors.ENDC} ELAPSED TIME: {time.time()-start\_time} sec")

        return final\_buff

    #Сохранение циклов

    def collect\_cycles(self,start\_i,start\_j):

        self.all\_cycles\_buff += self.search\_cycles\_algorithm(start\_i, start\_j)

    #Вывод циклов в виде индексов (для дебаггинга)

    def print\_cycles\_indexes(self):

            print(f"\n{TextColors.OKGREEN}///DEBUG///{TextColors.ENDC} ЦИКЛЫ:")

            # for cycle in self.all\_cycles\_buff:

            #     print(cycle)

            for cycle\_list in self.all\_cycles\_buff:

                cycle = str()

                for elem in cycle\_list:

                    cycle+=str(elem[0])+"->"

                cycle+=str(cycle\_list[len(cycle\_list)-1][1])

                print(cycle)

    #Вывод циклов

    def print\_cycles(self,cycle\_length=None):

        original\_lists=[]

        for cycle\_list in self.all\_cycles\_buff:

            cycle = str()

            if cycle\_length==None:

                self.cleared\_cycles(cycle\_list,cycle,original\_lists)

            else:

                if cycle\_length==len(cycle\_list):

                    self.cleared\_cycles(cycle\_list,cycle,original\_lists)

        self.all\_cycles\_buff=[]

    #Очистка повторяющихся циклов их вывод

    def cleared\_cycles(self,cycle\_list,cycle,original\_lists):

        temp\_cycle\_list=[]

        for elem in cycle\_list:

            cycle+=list(self.vertex\_names.keys())[list(self.vertex\_names.values()).index(elem[0])]+"->"

            temp\_cycle\_list.append(elem[0])

        cycle+=list(self.vertex\_names.keys())[list(self.vertex\_names.values()).index(cycle\_list[len(cycle\_list)-1][1])]

        in\_original\_lists = False

        for elem in original\_lists:

            if set(temp\_cycle\_list)!=set(elem):

                continue

            else:

                in\_original\_lists=True

                break

        if in\_original\_lists==False:

            original\_lists.append(temp\_cycle\_list)

            print(cycle)

    #Поиск циклов

    def find\_all\_cycles(self,cycle\_length=None):

        for i in range(self.length):

            for j in range(self.length):

                if self.matrix[i][j]==1:

                    self.collect\_cycles(i,j)

        if cycle\_length==None:

            print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ВСЕ ЦИКЛЫ:")

        else:

            print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ЦИКЛЫ ДЛИНОЙ, РАВНОЙ {TextColors.OKBLUE}{cycle\_length}{TextColors.ENDC}:")

        self.print\_cycles(cycle\_length)

    #Поиск циклов из ребра

    def find\_cycles\_from\_edge(self,v1\_name,v2\_name,cycle\_length=None):

        if v1\_name and v2\_name in self.vertex\_names:

            v1\_index=self.vertex\_names[v1\_name]

            v2\_index=self.vertex\_names[v2\_name]

            self.collect\_cycles(v1\_index,v2\_index)

            if cycle\_length==None:

                print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ЦИКЛЫ ИЗ РЕБРА ['{v1\_name}'->'{v2\_name}']:")

            else:

                print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ЦИКЛЫ ИЗ РЕБРА ['{v1\_name}'->'{v2\_name}'] ДЛИНОЙ, РАВНОЙ {TextColors.OKBLUE}{cycle\_length}{TextColors.ENDC}:")

            self.print\_cycles(cycle\_length)

        elif v1\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v1\_name}'!")

        elif v2\_name not in self.vertex\_names:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v2\_name}'!")

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИН С НАЗВАНИЯМИ '{v1\_name}' и '{v2\_name}'!")

    #Поиск циклов из вершины

    def find\_cycles\_from\_vertex(self,v\_name,cycle\_length=None):

        if v\_name in self.vertex\_names:

            v\_index = self.vertex\_names[v\_name]

            for j in range(self.length):

                    if self.matrix[v\_index][j]==1:

                        self.collect\_cycles(v\_index,j)

            if cycle\_length==None:

                print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ЦИКЛЫ ИЗ ВЕРШИНЫ '{v\_name}:")

            else:

                print(f"\n{TextColors.OKGREEN}///OUTPUT///{TextColors.ENDC} ЦИКЛЫ ИЗ ВЕРШИНЫ '{v\_name}' ДЛИНОЙ, РАВНОЙ {TextColors.OKBLUE}{cycle\_length}{TextColors.ENDC}:")

            self.print\_cycles(cycle\_length)

        else:

            print(f"{TextColors.FAIL}///ERROR///{TextColors.ENDC} НЕТ ВЕРШИНЫ С НАЗВАНИЕМ '{v\_name}'!")

graph = Graph()

graph.add\_vertex("A")

graph.add\_vertex("B")

graph.add\_vertex("C")

graph.add\_vertex("D")

graph.add\_vertex("E")

graph.add\_vertex("F")

graph.add\_vertex("G")

graph.add\_edge("A","B")

graph.add\_edge("B","C")

graph.add\_edge("C","D")

graph.add\_edge("D","A")

graph.add\_edge("B","D")

graph.add\_edge("A","G")

graph.add\_edge("G","D")

graph.add\_edge("C","E")

graph.add\_edge("E","B")

graph.add\_edge("E","F")

graph.add\_edge("F","B")

graph.add\_edge("F","A")

graph.find\_cycles\_from\_vertex("C",cycle\_length = 3)

graph.find\_cycles\_from\_vertex("C")

graph.find\_cycles\_from\_edge("G","D",cycle\_length = 2)

graph.find\_cycles\_from\_edge("G","D")

graph.find\_all\_cycles(cycle\_length = 3)

graph.find\_all\_cycles()

graph.visualize()

**Скриншот работы программы:**

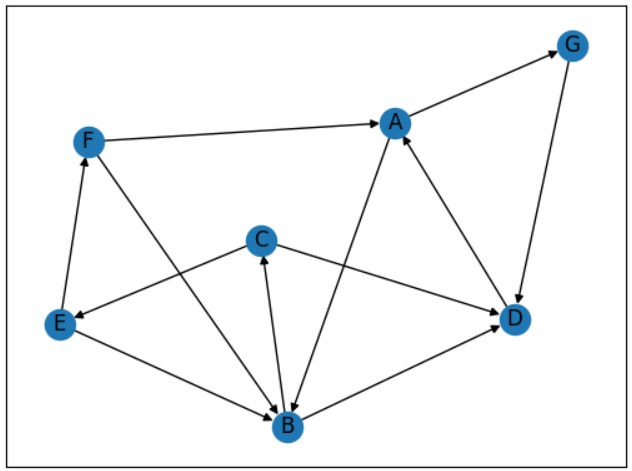
****

Рисунок 3. – Визуализация графа.

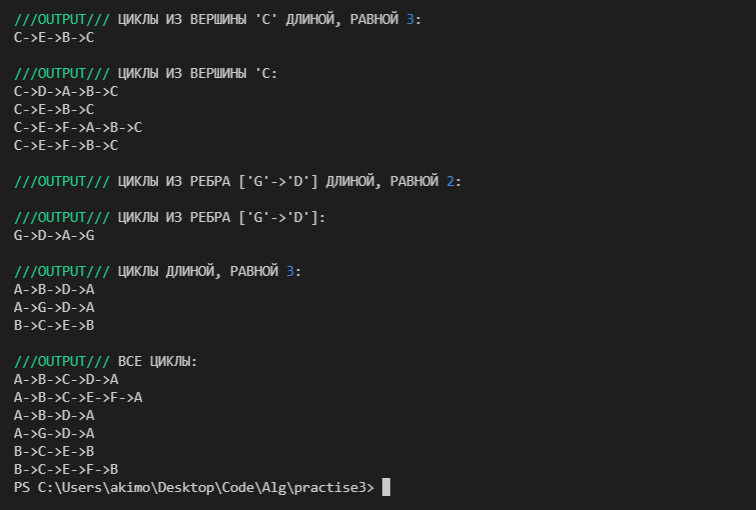
****

Рисунок 4. – Пример вывода информации.

**Выводы.**

В результате выполнения данной работы был реализован алгоритм, способный находить циклы из какой-либо вершины, ребра, все циклы, а также циклы определенной длины в ориентированном графе.

**Литература:**

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

2. https://en.wikipedia.org/wiki/Cycle\_(graph\_theory)