|  |
| --- |
| 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 № 30**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_\_\_\_

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

*(фио студента)*

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

Москва 2021 г.

**Вариант № 30.**

**Алгоритм:** определить внешний радиус невзвешенного ориентированного   
графа методом обхода в ширину. (Внешним радиусом графа   
будем называть наибольшее среди кратчайших расстояние от   
центра до какого-либо узла.) Вывести значение, а также   
соответствующие ему цепи.

**Способ представления графа:** Матрица инцидентности

**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.

**Матрица инцидентности** — одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро(дуга) и вершина). Столбцы матрицы соответствуют ребрам, строки — вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

В случае ориентированного графа каждой дуге <x,y> ставится в соответствующем столбце: «1» в строке вершины x и «-1» в строке вершины y; если связи между вершиной и ребром нет, то в соответствующую ячейку ставится «0».

Например, для графа изображенного на рисунке 1 матрица инцидентности будет выглядеть так:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| A | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 | 0 |
| B | -1 | 1 | 1 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 |
| C | 0 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | -1 | -1 | 0 | 1 | 0 | 0 | 0 | 0 | -1 |
| E | 0 | 0 | 0 | 0 | -1 | 0 | 1 | 1 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 1 | 0 |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Для поиска центра графа методом поиска в ширину составим при помощи него матрицу расстояний. Матрицей расстояний назовем матрицу, где в i-ой строке каждый j элемент – это минимальное расстояние от вершины i до вершины j (если существует путь проходящий через вершины графа).

Далее для каждой строки матрицы, где все элементы ненулевые, найдем максимальное значение расстояния. А потом из списка максимальных расстояний найдем минимальное, номер строки, которой принадлежит данное расстояние и является центром графа, а расстояние – радиус

Далее обходом в глубину от центра графа найдем цепи с длинной равной радиусу.

**Листинг программы.**

def read\_inc\_matrix(filename):

    with open(filename, 'r') as f:

        dim = int(f.readline().strip())

        matrix = [[0] \* dim for \_ in range(dim)]

        inc\_matrix = []

        for line in f.readlines():

            inc\_matrix.append(list(map(int, line.split())))

        for i in range(len(inc\_matrix[0])):

            first = None

            first\_e = 0

            second = None

            second\_e = 0

            for j in range(dim):

                if inc\_matrix[j][i] != 0:

                    if first is None:

                        first = j

                        if inc\_matrix[j][i] > 0:

                            first\_e = inc\_matrix[j][i]

                    else:

                        second = j

                        if inc\_matrix[j][i] > 0:

                            second\_e = inc\_matrix[j][i]

                    if first and second:

                        break

            if first\_e :

                matrix[first][second] = first\_e

            if second\_e:

                matrix[second][first] = second\_e

    return matrix

def BFS(matrix):

    distance\_matrix = [[0] \* len(matrix) for i in range(len(matrix))]

    def \_bfs(v: int):

        queue = [v]

        visited = set()

        while queue:

            cur\_node = queue.pop(0)

            for i, e in enumerate(matrix[cur\_node]):

                if i in visited:

                    continue

                if e > 0:

                    distance = e + distance\_matrix[v][cur\_node]

                    distance\_matrix[v][i] = distance

                    queue.append(i)

                    visited.add(i)

    for i in range(len(matrix)):

        \_bfs(i)

    return distance\_matrix

def find\_center(matrix):

    max\_distances = []

    for i, row in enumerate(matrix):

        if min(row) == 0:

            continue

        else:

            max\_distances.append((i, max(row)))

    max\_distances.sort(key=lambda tup: tup[1])

    return max\_distances[0]

def find\_chains(matrix: list[list[int]], v: int, distance: int):

    stack = []

    chains = []

    def \_dfs(v: int, \_distance: int):

        stack.append(v)

        for i, e in enumerate(matrix[v]):

            if i in stack:

                continue

            if e:

                if \_distance - e == 0:

                    chains.append([\*stack.copy(), i])

                elif \_distance - e >= 0:

                    \_dfs(i, \_distance - e)

        stack.remove(v)

    \_dfs(v, distance)

    return chains

if \_\_name\_\_ == '\_\_main\_\_':

    from pprint import pprint

    matrix = read\_inc\_matrix('graph\_task30.txt')

    distance\_matrix = BFS(matrix)

    print("Матрица расстояний:")

    pprint(distance\_matrix)

    center\_v, radius = find\_center(distance\_matrix)

    print(f"Центр графа: {center\_v}, радиус: {radius}")

    chains = find\_chains(matrix, center\_v, radius)

    print("Цепи соответствующие радиусу:")

    print("\n".join("->".join(map(str, chain)) for chain in chains))

**Скриншот работы программы:**

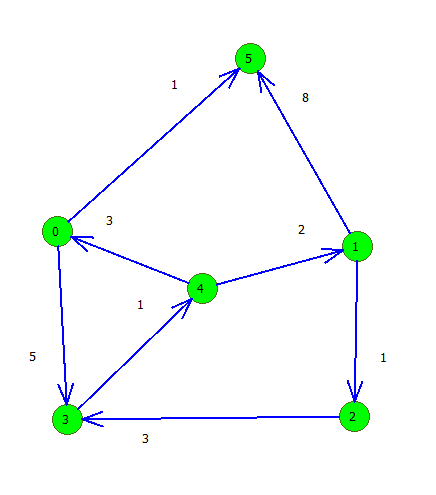


Рисунок 3. – Пример ориентированного нагруженного графа.

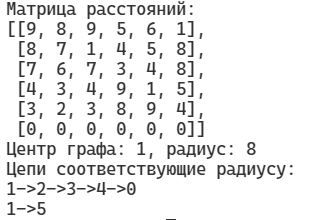
****

Рисунок 4. – Пример работы программы для графа из рисунка 3.

**Выводы.**

В результате выполнения данной работы была изучена структура граф, ее свойства и операции над ней. Так же был реализован алгоритм поиска в ширину для поиска центра графа, радиуса и цепей, соответствующих ему.

**Литература:**

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

2. https://e-maxx.ru/algo/bfs

3. https://ru.wikipedia.org/wiki/Центр\_графа