

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет: Информатика и вычислительная техника

Кафедра: Кибербезопасность информационных систем

**ЛАБОРАТОРНАЯ РАБОТА № 3**

на тему «Компоненты связности графа»

Выполнил обучающийся гр. ВКБ33

Никоноров Иван Игоревич

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2022

Опираясь на две предыдущие лабораторные работы, реализовать следующую

функциональность:

**№1**

Проверить, что две вершины принадлежат одной компоненте связности (есть путь, их соединяющий).

def read\_graph(file\_name: str):

    graph\_list = []

    try:

        with open(file\_name, 'r') as file:

            for line in file:

                graph\_list.append(list(map(int, line.split())))

        return graph\_list

    except:

        return None

def dfs(graph\_list: list, vertex: int):

    visited = [False] \* len(graph\_list)

    def step(vertex):

        visited[vertex] = True

        for sub\_vertex in graph\_list[vertex]:

            if not visited[sub\_vertex]:

                step(sub\_vertex)

    step(vertex)

    return visited

def count\_component(graph\_list: list):

    vertex\_component = [-1] \* len(graph\_list)

    visited = [False] \* len(graph\_list)

    number = 0

    while False in visited:

       visited\_dfs = dfs(graph\_list, visited.index(False))

       for ind in range(len(visited\_dfs)):

           if visited\_dfs[ind]:

               visited[ind] = True

               vertex\_component[ind] = number

       number += 1

    return number, vertex\_component

def is\_connected(graph\_list: list, first\_vertex: int, second\_vertex: int):

    component = count\_component(graph\_list)[1]

    if component[first\_vertex] == component[second\_vertex]:

        return True

    else:

        return False

g = read\_graph('graph1.txt')

print(dfs(g, 0))

print(count\_component(g)[0])

print(count\_component(g)[1])

print(is\_connected(g, 1, 3))

print(is\_connected(g, 0, 6))

class Graph():

    def \_\_init\_\_(self, V):

        self.V = V

        self.graph = [[0 for column in range(V)] \

                                for row in range(V)]

    def isBipartite(self, src):

        colorArr = [-1] \* self.V

        colorArr[src] = 1

        queue = []

        queue.append(src)

        while queue:

            u = queue.pop()

            if self.graph[u][u] == 1:

                return False

            for v in range(self.V):

                if self.graph[u][v] == 1 and colorArr[v] == -1:

                    colorArr[v] = 1 - colorArr[u]

                    queue.append(v)

                elif self.graph[u][v] == 1 and colorArr[v] == colorArr[u]:

                    return False

        return True

g = Graph(4)

g.graph = [[0, 1, 0, 1],

            [1, 0, 1, 0],

            [0, 1, 0, 1],

            [1, 0, 1, 0]]

print ("Yes" if g.isBipartite(0) else "No")

**№2**

Выделить компоненту связности, к которой принадлежит данная вершина.

vector<int> topologySortInit(graphNotWeighted &G) {

    int n = G.size();

    vector<int> resultOrder;

    vector<char> color(n, 'w');

    for (int v = 0; v < n; v++)

        if (color[v] == 'w')

            topologySortProc(G, v, color, resultOrder);

    reverse(resultOrder.begin(), resultOrder.end());

    return resultOrder;

}

void topologySortProc(graphNotWeighted &G, int start, vector<char> &color, vector<int> &order) {

    stack<pair<int, int> > S;

    S.push(make\_pair(start, -1));

    color[start] = 'g';

    while (S.size()) {

        int v = S.top().first;

        int u = S.top().second;

        int w = -1;

        for (int i = 0; i < G[v].size(); i++)

            if (G[v][i] != u && color[G[v][i]] == 'w') {

                w = G[v][i];

                break;

            }

        if (w == -1) {

            S.pop();

            color[v] = 'b';

            order.push\_back(v);

        }

        else {

            S.top().second = w;

            if (color[w] == 'w') {

                S.push(make\_pair(w, -1));

                color[w] = 'g';

            }

        }

    }

    return;

}

**№3**

Разбить данный граф на составляющие его компоненты связности

graphNotWeighted transposingGraph(graphNotWeighted &G) {

    int n = G.size();

    graphNotWeighted GT(n);

    vector<int> count(n);

    for (int i = 0; i < n; i++)

        for (int j = 0; j < G[i].size(); j++)

            count[G[i][j]]++;

    for (int i = 0; i < n; i++)

        GT[i].reserve(count[i]);

    for (int i = 0; i < n; i++)

        for (int j = 0; j < G[i].size(); j++)

            GT[G[i][j]].push\_back(i);

    return GT;

}

graphNotWeighted getComponent(graphNotWeighted &G, int start, vector<char> &color) {

    stack<pair<int, int> > S;

    graphNotWeighted component(G.size());

    bool isEmpty = false;

    for(int p = 0; p < G[start].size(); p++)

        if(color[G[start][p]] == 'w')

            isEmpty = true;

    if(isEmpty == false) {

        component[start].push\_back(start);

        color[start] = 'b';

        return component;

    }

    S.push(make\_pair(start, -1));

    while (S.size()) {

        int v = S.top().first;

        int u = S.top().second;

        int w = -1;

        for (int i = 0; i < G[v].size(); i++)

            if (G[v][i] != u && color[G[v][i]] == 'w') {

                w = G[v][i];

                break;

            }

        if (w == -1) {

            S.pop();

            color[v] = 'b';

        }

        else {

            S.top().second = w;

            if (color[w] == 'w') {

                component[w].push\_back(v);

                S.push(make\_pair(w, -1));

                color[w] = 'g';

            }

        }

    }

    return component;

}

vector<graphNotWeighted> strongConnectedComponents(graphNotWeighted &G) {

    int n = G.size();

    vector<int> order = topologySortInit(G);

    graphNotWeighted GT = transposingGraph(G);

    vector<graphNotWeighted> strongConnectedComponents;

    vector<char> color(n, 'w');

    for (int i = 0; i < n; i++)

        if (color[order[i]] == 'w')

            strongConnectedComponents.push\_back(getComponent(GT, order[i], color));

    return strongConnectedComponents;

}