

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
по дисциплине «Алгоритмы и Структуры данных»
ТЕМА: «ГРАФЫ»

Студент гр. 1308,	_____	Мельник Д. А.
Студент гр. 1308,	_____	Лепов А. В.
Научный руководитель,	_____	Манирагена В.

Санкт-Петербург
2022 г.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ	3
1.1. Цель работы.....	3
1.2. Формулировки общего задания	3
1.3. Формулировка индивидуального задания.....	3
2. МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ В ТЕРМИНАХ ТЕОРИИ МНОЖЕСТВ	4
3. ВЫБОР И ОБОСНОВАНИЕ СПОСОБА ПРЕДСТАВЛЕНИЯ ДАННЫХ И ОПИСАНИЕ АЛГОРИТМА ПРОГРАММЫ.....	5
3.1. Способ представления данных	5
3.2. Описание алгоритма.	5
4. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ РАБОТЫ ПРОГРАММЫ.....	6
4.1. Задание пользователем исходных данных	6
4.2. Графическое представление	6
4.3. Результат работы программы	7
5. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ АЛГОРИТМА	8
6. ВЫВОДЫ	9
7. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	10
8. ПРИЛОЖЕНИЕ А.	11
8.1. Исходный текст программы для ЭВМ	11
8.2. Листинг программного кода h-файлов	12
8.2.1. Листинг «ElementaryCyclesSearch.h».....	12
8.2.2. Листинг «SCCResult.h»	15
8.2.3. Листинг «StrongConnectedComponents.h»	16
8.2.4. Листинг «Utils.h»	20
8.3. Файл с тестами.....	21
8.4. Программный файл для отображения графа.....	23

1. ВВЕДЕНИЕ

1.1. Цель работы

Исследование алгоритмов для работы с ориентированным графом, анализ структур данных, наиболее подходящих для реализации программы.

1.2. Формулировки общего задания

Выполнить курсовую работу по предложенному преподавателем индивидуальному заданию. Детальную постановку задачи взять из рекомендованных литературных источников. Реализовать алгоритм на языке C++ с использованием объектов и возможностей STL.

Выбрать оптимальную структуру данных для представления графа в памяти ЭВМ. Реализовать граф как объект, а обработку — как функцию-член для него. Результат обработки может быть или не быть частью объекта, способ его представления выбирается особо.

Для объекта должны быть объявлены все вспомогательные методы (методы по умолчанию) — конструкторы, деструктор и т. п. Использование ненужных методов блокируется на уровне компиляции или выполнения.

Стек и очередь (если нужны) берутся из STL.

Интерфейс программы должен быть удобен для испытаний алгоритма. Следует предусмотреть ввод заранее заготовленных и генерацию произвольных тестовых данных.

Дополнительное требование: оценить возможный объём исходных данных для решения поставленной задачи для следующих ограничений:

- возможность вывода данных на экран;
- доступный объём памяти;
- получение решения за разумное время.

1.3. Формулировка индивидуального задания

Таблица. № 1. Индивидуальный вариант задания

№ варианта	Алгоритм для исследования
4	Обнаружение всех элементарных циклов ориентированного графа.

2. МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ В ТЕРМИНАХ ТЕОРИИ МНОЖЕСТВ

Циклом в графе называется такая конечная цепь, которая начинается и заканчивается в одной вершине. Цикл обозначается последовательностью вершин, которые он содержит, например: (3-5-6-7-3).

Цикл называется *элементарным*, если все вершины, входящие в него, различны (за исключением начальной и конечной).

В данной работе нам необходимо искать в выбранном (посланным на ввод) пользователем графе именно элементарные циклы.

Пусть граф задан, как $G = (V, E)$, где V — множество вершин графа, а E — множество его ребер; x, y — переменные, встречающиеся в коде — координаты вершины на плоскости.

Для поиска всех элементарных цепей модифицируем рекурсивный алгоритм «поиск в глубину» (*DFS*) — это один из алгоритмов обхода графа.

3. ВЫБОР И ОБОСНОВАНИЕ СПОСОБА ПРЕДСТАВЛЕНИЯ ДАННЫХ И ОПИСАНИЕ АЛГОРИТМА ПРОГРАММЫ

3.1. Способ представления данных

В качестве способа представления данных для графа выбрано:

- для ввода пользователем выбрана строка, так как пользователю намного проще набирать текст, не задумываясь о переносе строки;
- для хранения графа перед обработкой выбран булев массив (матрица смежности);

3.2. Описание алгоритма.

Сначала строкой задается обходом вершин в глубину или любой цепью графа:

- через пробел записываются связанные вершины – ориентация от левой вершины к правой;
- через символ перевода строки «\n» записываются разные цепи.

Затем эта строка разделяется на цепи и далее каждая цепь разделяется на отдельные вершины, которые записываются в матрицу смежности.

Затем с помощью построенной матрицы ищутся элементарные циклы.

И, наконец, производится вывод на экран результата: найденные элементарные циклы.

4. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ РАБОТЫ ПРОГРАММЫ

4.1. Задание пользователем исходных данных

Пользователь задает граф следующим образом (графическое отображение в п. 4.2.):

«1 5 4\n2 6 5 4 8\n3 7 6 10 9 13 12\n6 10 9\n7 11 15 14 13\n8 4 5 6 2\n9 10 6\n11 15 14\n12 8 9 10 11 7 3\n13 9 10 6 7\n14 10 11»

4.2. Графическое представление

На рисунке 1 продемонстрирован результат работы программного модуля для графического представления графа, на языке программирования python (исходный код также представлен в приложении А).

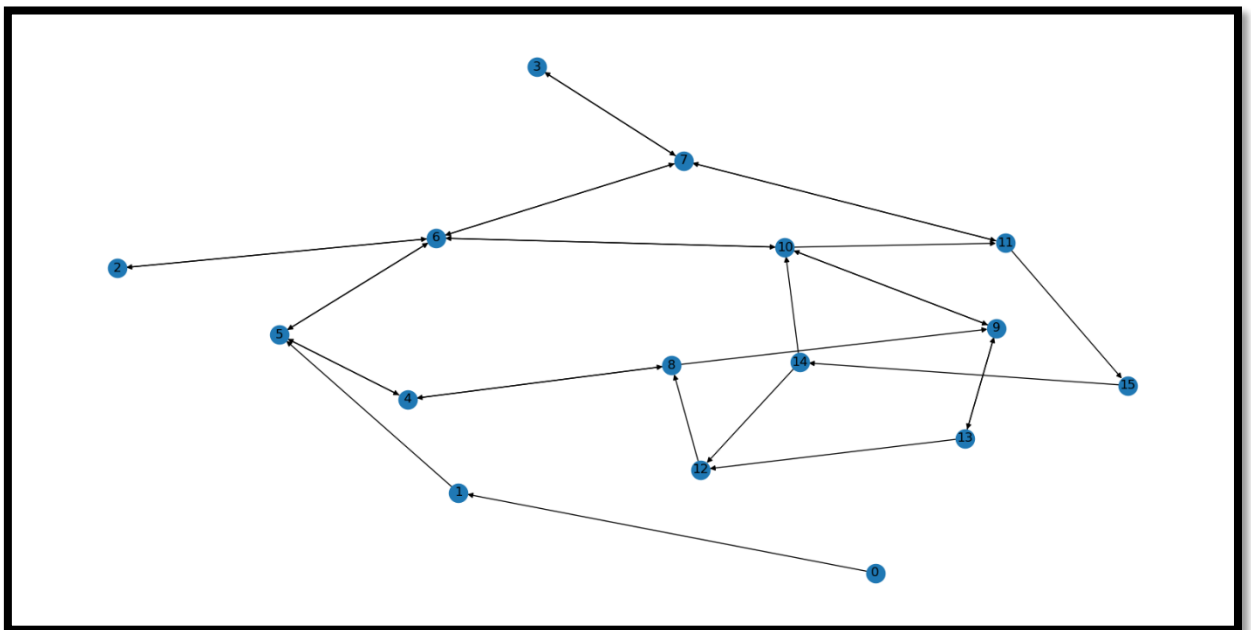
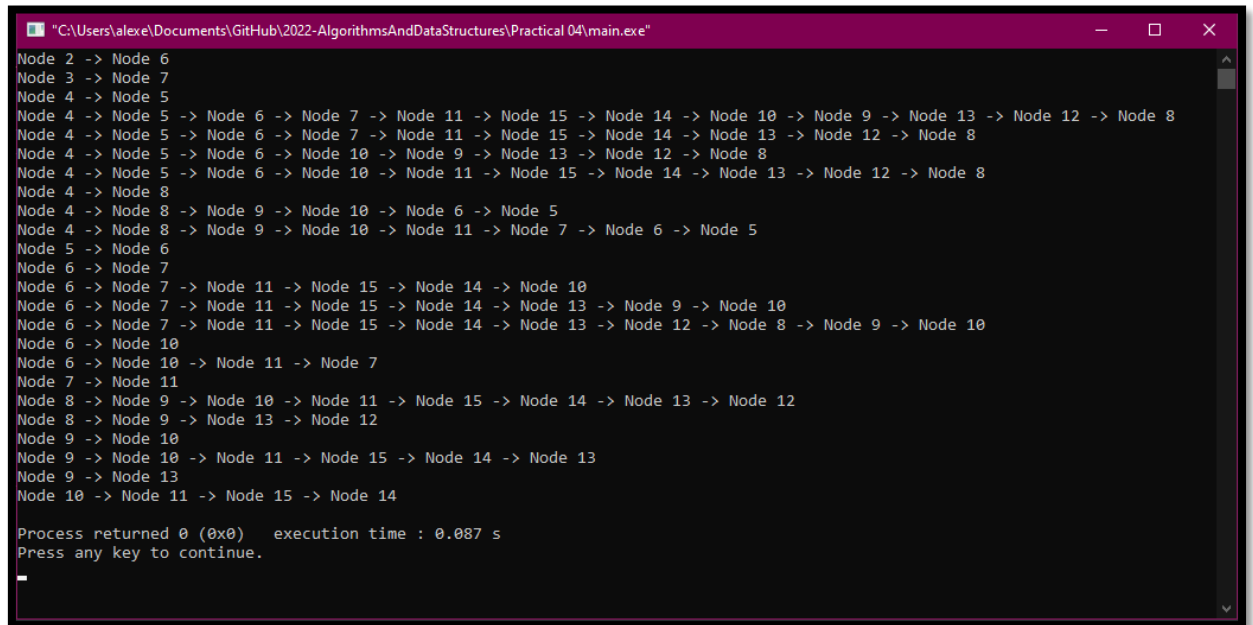


Рис. 1. Графическое представление графа

4.3. Результат работы программы

Результатом работы программы является вывод всех элементарных циклов заданного пользователем графа (рисунок 2).



```
"C:\Users\alex\Documents\GitHub\2022-AlgorithmsAndDataStructures\Practical 04\main.exe"
Node 2 -> Node 6
Node 3 -> Node 7
Node 4 -> Node 5
Node 4 -> Node 5 -> Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 10 -> Node 9 -> Node 13 -> Node 12 -> Node 8
Node 4 -> Node 5 -> Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12 -> Node 8
Node 4 -> Node 5 -> Node 6 -> Node 10 -> Node 9 -> Node 13 -> Node 12 -> Node 8
Node 4 -> Node 5 -> Node 6 -> Node 10 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12 -> Node 8
Node 4 -> Node 8
Node 4 -> Node 8 -> Node 9 -> Node 10 -> Node 6 -> Node 5
Node 4 -> Node 8 -> Node 9 -> Node 10 -> Node 11 -> Node 7 -> Node 6 -> Node 5
Node 5 -> Node 6
Node 6 -> Node 7
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 10
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 9 -> Node 10
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12 -> Node 8 -> Node 9 -> Node 10
Node 6 -> Node 10
Node 6 -> Node 10 -> Node 11 -> Node 7
Node 7 -> Node 11
Node 8 -> Node 9 -> Node 10 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12
Node 8 -> Node 9 -> Node 13 -> Node 12
Node 9 -> Node 10
Node 9 -> Node 10 -> Node 11 -> Node 15 -> Node 14 -> Node 13
Node 9 -> Node 13
Node 10 -> Node 11 -> Node 15 -> Node 14
Process returned 0 (0x0)   execution time : 0.087 s
Press any key to continue.
```

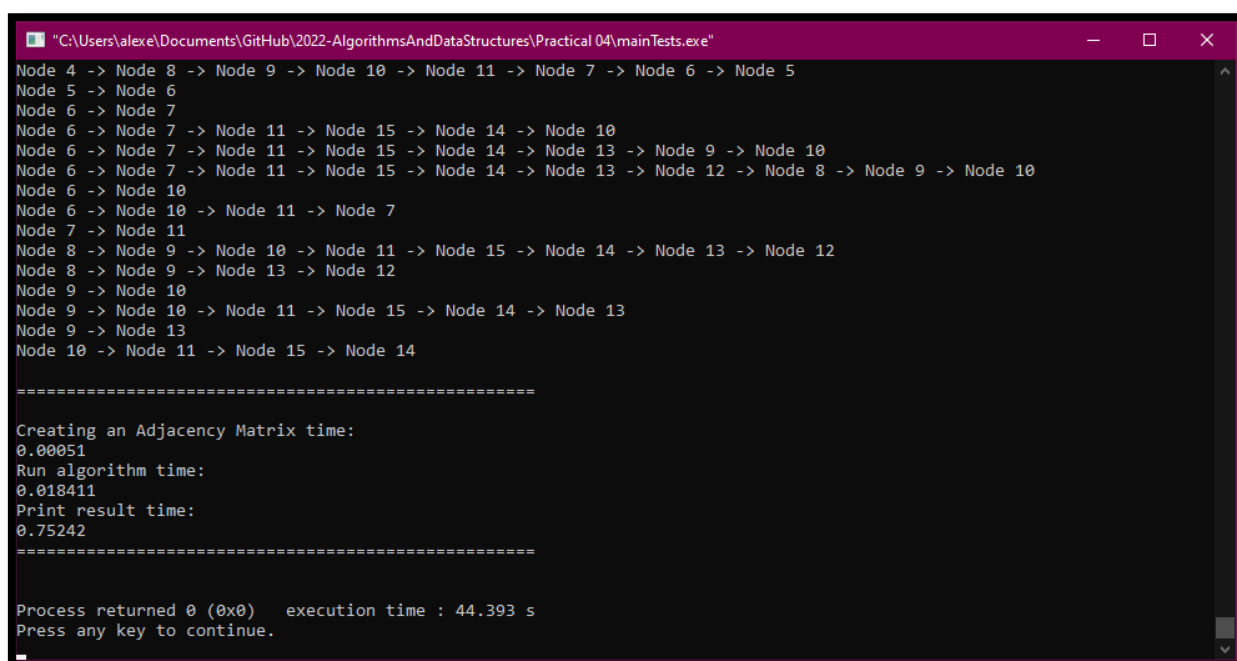
Рис. 2. Результат работы программы

5. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ АЛГОРИТМА

Таблица. № 2. Индивидуальный вариант задания

Создание матрицы смежности	Время поиска элементарных циклов	Время вывода результата на экран
0.00051 с.	0.018411 с.	0.75242 с.

Ниже, на рисунке 1 продемонстрирован результат тестирования алгоритма.



```
"C:\Users\alexe\Documents\GitHub\2022-AlgorithmsAndDataStructures\Practical 04\mainTests.exe"
Node 4 -> Node 8 -> Node 9 -> Node 10 -> Node 11 -> Node 7 -> Node 6 -> Node 5
Node 5 -> Node 6
Node 6 -> Node 7
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 10
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 9 -> Node 10
Node 6 -> Node 7 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12 -> Node 8 -> Node 9 -> Node 10
Node 6 -> Node 10
Node 6 -> Node 10 -> Node 11 -> Node 7
Node 7 -> Node 11
Node 8 -> Node 9 -> Node 10 -> Node 11 -> Node 15 -> Node 14 -> Node 13 -> Node 12
Node 8 -> Node 9 -> Node 13 -> Node 12
Node 9 -> Node 10
Node 9 -> Node 10 -> Node 11 -> Node 15 -> Node 14 -> Node 13
Node 9 -> Node 13
Node 10 -> Node 11 -> Node 15 -> Node 14

=====
Creating an Adjacency Matrix time:
0.00051
Run algorithm time:
0.018411
Print result time:
0.75242
=====

Process returned 0 (0x0)   execution time : 44.393 s
Press any key to continue.
```

Рис. 3. Вывод результата временного тестирования.

6. ВЫВОДЫ

В результате выполнения курсовой работы и испытаний алгоритмов для задания и обработки ориентированного графа, а также временного тестирования студенты изучили необходимую информацию для реализации программы и теоретические основы структур данных.

7. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. П. Г. Колинко // «Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1»: пользовательские структуры данных» // Издательство: СПб.: СПбГЭТУ «ЛЭТИ», 2022г. – [64 с.]
2. Dale, Nell B. «C++ plus data stractures» // Издательство: Prentice-Hall, Inc., 2003г. – [816 с.]
3. Robert L. Kruse, Alexander J. Ryba // «Data Structures and Program Design in C++» // Издательство: Prentice-Hall, Inc., 2000г. – [717 с.]

8. ПРИЛОЖЕНИЕ А.

Исходный код программы, тестов, заголовочных файлов, документация расположены на онлайн-ресурсе github: <https://github.com/AlexeyLepov/LETI-2022-AlgorithmsAndDataStructures-team04>

8.1. Исходный текст программы для ЭВМ

```
#include "Utils.h"
#include "SCCResult.h"
#include "StrongConnectedComponents.h"
#include "ElementaryCyclesSearch.h"

int main(void)
{
    const int N = 16;
    vector<string> nodes(N);
    vector < vector<bool> > adjMatrix(N, vector<bool>(N));

    // initialization
    for (int i = 0; i < N; i++)
    {
        nodes[i] = "Node " + std::to_string(i);
    }

    string SR = "1 2\n3 2\n3 4 5 6 7\n4 5 6 7 3\n5 6 3\n6 3\n6 7 3";
    // "1 2 3\n2 3 1\n3 1 2\n3 2";
    // "1 5 4\n2 6 5 4 8\n3 7 6 10 9 13 12\n4 0 1\n6 10 9\n7 11 15 14 13\n8 4 5 6
    2\n9 10 6\n11 15 14\n12 8 9 10 11 7 3\n13 9 10 6 7\n14 10 11";

    vector<string> lines = split(SR, '\n');

    // Creating an Adjacency Matrix
    for (int i = 0; i < lines.size(); i++)
    {
        vector<string> components = split(lines[i], ' ');

        for (int i = 0; i < components.size() - 1; i++)
        {
            int x = atoi(components[i].c_str());
            int y = atoi(components[i + 1].c_str());
            adjMatrix[x][y] = true;
        }
    }

    // Run algorithm
    ElementaryCyclesSearch ecs = ElementaryCyclesSearch(adjMatrix, nodes);
    vector<vector<string> > cycles = ecs.getElementaryCycles();

    // Print result
    for (int i = 0; i < cycles.size(); i++)
    {
        vector<string> cycle = cycles[i];
        for (int j = 0; j < cycle.size(); j++)
        {
            string node = cycle[j];
            if (j < cycle.size() - 1)
            {
```

```

        cout << node << " -> ";
    }
    else
    {
        cout << node;
    }
}
cout << endl;
}
return 0;
}

```

8.2. Листинг программного кода h-файлов

8.2.1. Листинг «ElementaryCyclesSearch.h»

```

#ifndef ELEMENTARYCYCLESSEARCH_H
#define ELEMENTARYCYCLESSEARCH_H

#include "StrongConnectedComponents.h"

class ElementaryCyclesSearch
{
public:

    ElementaryCyclesSearch()
    {
    }

    ElementaryCyclesSearch(const ElementaryCyclesSearch& orig)
    {
    }

    virtual ~ElementaryCyclesSearch()
    {
    }

    /**
     * Конструктор
     *
     * @param матрица смежности графа
     * @param graphNodes массив узлов графа; используется для постройки элементарных
    циклов содержащих объекты исходного графа
     */
    ElementaryCyclesSearch(vector<vector<bool> > matrix, vector<string> gNodes)
    {
        graphNodes = gNodes;
        adjList = getAdjacencyList(matrix);
    }

    /**
     * Возвращает объект с векторами узлов всех элементарных циклов графа
     *
     * @return Vector::Vector::Object с Векторами (Vectors) узлов всех элементарных
    циклов
     */
    vector<vector<string> > getElementaryCycles()
    {
        blocked.clear();
        B.clear();
    }
}

```

```

        if(blocked.size()==0) blocked.resize(adjList.size());
        if(B.size()==0)B.resize(adjList.size());
        StrongConnectedComponents *sccs = new StrongConnectedComponents(adjList);
        int s = 0;

        while (true)
        {
            SCCResult *sccResult = sccs->getAdjacencyList(s);
            if (sccResult != NULL && sccResult->getAdjList().size() > 0)
            {
                vector<vector<int> > scc = sccResult->getAdjList();
                s = sccResult->getLowestNodeId();
                for (int j = 0; j < scc.size(); j++)
                {
                    if (scc[j].size() > 0)
                    {
                        blocked[j] = false;
                        B[j].clear();
                    }
                }

                findCycles(s, s, scc);
                s++;
            }
            else
            {
                break;
            }
        }
        return cycles;
    }

private:
    /**
     * Вектор циклов
     */
    vector<vector<string> > cycles;

    /**
     * Вектор смежности графа
     */
    vector<vector<int> > adjList;

    /**
     * Узлы графа
     */
    vector<string> graphNodes;

    /**
     * Заблокированные узлы, используемые алгоритмом Джонсона
     */
    vector<bool> blocked;

    /**
     * В-векторы, используемые алгоритмом Джонсона
     */
    vector<vector<int> > B;

    /**
     * Стек для узлов, используемый алгоритмом Джонсона
     */
    vector<int> stack;

```

```

/**
 * Вычисляет циклы, содержащие данный узел в компоненте сильной связности. Метод
 рекурсивно вызывает сам себя
 *
 * @param v
 * @param s
 * @param список смежности adjList с подграфом сильно связанного компонента s
 * @return true, если цикл найден; false - иначе
 */
bool findCycles(int v, int s, vector<vector<int>> adjList)
{
    bool f = false;
    stack.push_back(v);
    blocked[v] = true;

    if(v == 13 || v == 9)
        v = v+1-1;

    for (int i = 0; i < adjList[v].size(); i++)
    {
        int w = adjList[v][i];
        // Найдти цикл
        if (w == s)
        {
            vector<string> cycle;
            for (int j = 0; j < stack.size(); j++)
            {
                int index = (stack[j]);
                cycle.push_back(graphNodes[index]);
            }
            cycles.push_back(cycle);
            f = true;
        }
        else if (!blocked[w])
        {
            if (findCycles(w, s, adjList))
            {
                f = true;
            }
        }
    }

    if (f)
    {
        unblock(v);
    }
    else
    {
        for (int i = 0; i < adjList[v].size(); i++)
        {
            int w = (adjList[v][i]);
            if (std::find(B[w].begin(), B[w].end(), v) == B[w].end())
            {
                B[w].push_back(v);
            }
        }
    }
    stack.erase(std::remove(stack.begin(), stack.end(), v), stack.end());
    return f;
}

```

```

/**
 * Рекурсивное разблокирование всех заблокированных узлов, начиная с данного
 узла.
 *
 * @param node - узел для разблокировки
 */
void unblock(int node)
{
    blocked[node] = false;
    //vector<int> Bnode = B[node];

    while (B[node].size() > 0)
    {
        int w = B[node][0];
        B[node].erase(B[node].begin() + 0);
        if (blocked[w])
        {
            unblock(w);
        }
    }
}

};

#endif /* ELEMENTARYCYCLESSEARCH_H */

```

8.2.2. Листинг «SCCResult.h»

```

#ifndef STRONGCONNECTEDCOMPONENTSRESULT_H
#define STRONGCONNECTEDCOMPONENTSRESULT_H

#include <unordered_set>

using namespace std;

class SCCResult
{
public:

    SCCResult()
    {
    }

    SCCResult(const SCCResult& orig)
    {
    }

    virtual ~SCCResult()
    {
    }

    SCCResult(vector<vector<int> > adjL, int lId)
    {
        adjList = adjL;
        lowestNodeId = lId;
        if (adjList.size() > 0)
        {
            for (int i = lowestNodeId; i < adjList.size(); i++)
            {
                if (adjList[i].size() > 0)

```

```

        {
            nodeIDsOfSCC.insert(i);
        }
    }
}

vector<vector<int> > getAdjList()
{
    return adjList;
}

int getLowestNodeId()
{
    return lowestNodeId;
}

private:
    std::unordered_set<int> nodeIDsOfSCC;
    vector<vector<int> > adjList;
    int lowestNodeId = -1;
};

#endif // SCCRESULT_H

```

8.2.3. Листинг «StrongConnectedComponents.h»

```

#ifndef STRONGCONNECTEDCOMPONENTS_H
#define STRONGCONNECTEDCOMPONENTS_H

#include <math.h>

class StrongConnectedComponents
{
public:

    StrongConnectedComponents()
    {
    }

    StrongConnectedComponents(const StrongConnectedComponents& orig)
    {
    }

    virtual ~StrongConnectedComponents()
    {
    }

    /**
     * Конструктор
     *
     * @param adjList-список смежности графа
     */
    StrongConnectedComponents(vector<vector<int> > adjList)
    {
        adjListOriginal = adjList;
    }
}

```



```

/**
 * Этот метод возвращает структуру смежности компонента сильной связности
 * с наименьшей вершиной в подграфе исходного графа, порожденного узлами
 * {s, s + 1, ..., n}, где s - заданный узел
 *
 * (Функция не будет возвращать один узел)
 *
 * @param node node s
 * @return HFILE_STRONGCONNECTEDCOMPONENTSRESULT_H with adjacency-structure of
the strong connected
 * component; null, if no such component exists
 */
SCCResult* getAdjacencyList(int node)
{
    visited.clear();
    lowlink.clear();
    number.clear();
    stack.clear();
    currentSCCs.clear();
    if(visited.size()==0)visited.resize(adjListOriginal.size());
    if(lowlink.size()==0)lowlink.resize(adjListOriginal.size());
    if(number.size()==0)number.resize(adjListOriginal.size());

    makeAdjListSubgraph(node);

    for (int i = node; i < adjListOriginal.size(); i++)
    {
        if (!visited[i])
        {
            getStrongConnectedComponents(i);
            vector<int> nodes = getLowestIdComponent();
            if (nodes.size() > 0 && std::find(nodes.begin(), nodes.end(), node)
== nodes.end() && std::find(nodes.begin(), nodes.end(), node + 1) == nodes.end())
            {
                return getAdjacencyList(node + 1);
            }
            else
            {
                vector<vector<int> > adjacencysList = getAdjList(nodes);
                if (adjacencysList.size() > 0)
                {
                    for (int j = 0; j < adjListOriginal.size(); j++)
                    {
                        if (adjacencysList[j].size() > 0)
                        {
                            return new SCCResult(adjacencysList, j);
                        }
                    }
                }
            }
        }
    }
    return NULL;
}

private:

/**
 * Строит список смежности для подграфа, содержащего только узлы >= заданного
индекса
 *
 * @param node Узел с наименьшим индексом в подграфе

```

```

    */
void makeAdjListSubgraph(int node)
{
    adjList.clear();
    if(adjList.size()==0)adjList.resize(adjListOriginal.size());

    for (int i = node; i < adjList.size(); i++)
    {
        vector<int> successors;
        for (int j = 0; j < adjListOriginal[i].size(); j++)
        {
            if (adjListOriginal[i][j] >= node)
            {
                successors.push_back(adjListOriginal[i][j]);
            }
        }
        if (successors.size() > 0)
        {
            adjList[i].clear();
            if(adjList[i].size()==0) adjList[i].resize(successors.size());
            for (int j = 0; j < successors.size(); j++)
            {
                int succ = successors[j];
                adjList[i][j] = succ;
            }
        }
    }
}

/**
 * Вычисляет компонент сильной связности из набора scc, который содержит узел с
наименьшим индексом
 *
 * @return Vector::Integer - из scc, содержащего наименьший номер узла
 */
vector<int> getLowestIdComponent()
{
    int min = adjList.size();
    vector<int> currScc;

    for (int i = 0; i < currentSCCs.size(); i++)
    {
        vector<int> scc = currentSCCs[i];
        for (int j = 0; j < scc.size(); j++)
        {
            int node = scc[j];
            if (node < min)
            {
                currScc = scc;
                min = node;
            }
        }
    }
    return currScc;
}

/**
 * @return Vector[]::Integer - представляющая структуру смежности компонента
 * сильной связности с наименьшей вершиной в просматриваемом в данный момент
 * подграфе
 */
vector<vector<int> > getAdjList(vector<int> nodes)

```

```

{
    vector<vector<int> > lowestIdAdjacencyList;
    if (nodes.size() > 0)
    {
        lowestIdAdjacencyList.clear();
        if(lowestIdAdjacencyList.size()==0)lowestIdAdjacencyList.resize(adjList.s
size());
        //          for (int i = 0; i < lowestIdAdjacencyList..size(); i++) {
        //              lowestIdAdjacencyList[i] = new Vector();
        for (int i = 0; i < nodes.size(); i++)
        {
            int node = nodes[i];
            for (int j = 0; j < adjList[node].size(); j++)
            {
                int succ = adjList[node][j];
                if (std::find(nodes.begin(), nodes.end(), succ) != nodes.end())
                {
                    lowestIdAdjacencyList[node].push_back(succ);
                }
            }
        }
        return lowestIdAdjacencyList;
    }
}

/**
 * Ищет компоненты сильной связности, достижимые из заданного узла
 *
 * @param root - корень
 */
void getStrongConnectedComponents(int root)
{
    sccCounter++;
    lowlink[root] = sccCounter;
    number[root] = sccCounter;
    visited[root] = true;
    stack.push_back(root);
    for (int i = 0; i < adjList[root].size(); i++)
    {
        int w = adjList[root][i];
        if (!visited[w])
        {
            getStrongConnectedComponents(w);
            lowlink[root] = min(lowlink[root], lowlink[w]);
        }
        else if (number[w] < number[root])
        {
            if (std::find(stack.begin(), stack.end(), w) != stack.end())
            {
                lowlink[root] = min(lowlink[root], number[w]);
            }
        }
    }
}

// найдена компонента сильной связности - SCC
if ((lowlink[root] == number[root]) && (stack.size() > 0))
{
    int next = -1;
    vector<int> scc;

    do {
        next = stack[stack.size() - 1];

```

```

        stack.erase(stack.begin()+ stack.size() - 1);
        scc.push_back(next);
    } while (number[next] > number[root]);

    // простые компоненты сильной связанности с одним узлом не будут
добавлены
    if (scc.size() > 1)
    {
        currentSCCs.push_back(scc);
    }
}

/**
 * Вектор смежности исходного графа
 */
vector<vector<int> > adjListOriginal;

/**
 * Список смежности просматриваемого в данный момент подграфа
 */
vector<vector<int> > adjList;

/**
 * Вспомогательный объект для поиска SCC
 */
vector<bool> visited;

/**
 * Вспомогательный объект для поиска SCC
 */
vector<int> stack;

/**
 * Вспомогательный объект для поиска SCC
 */
vector<int> lowlink;

/**
 * Вспомогательный объект для поиска SCC
 */
vector<int> number;

/**
 * Вспомогательный объект для поиска SCC
 */
int sccCounter = 0;

/**
 * Вспомогательный объект для поиска SCC
 */
vector<vector<int> > currentSCCs;

};

#endif /* STRONGCONNECTEDCOMPONENTS_H */

```

8.2.4. Листинг «Utils.h»

```

#ifndef GLOBAL_H
#define GLOBAL_H

#include <list>
#include <string>
#include <vector>
#include <sstream>
#include <iostream>
#include <algorithm>

using namespace std;

vector<vector <int> > getAdjacencyList(vector< vector <bool> > adjacencyMatrix)
{
    vector<vector <int> > LR;
    for (int i = 0; i < adjacencyMatrix.size(); i++)
    {
        vector < int > vx;
        for (int j = 0; j < adjacencyMatrix[i].size(); j++)
        {
            if (adjacencyMatrix[i][j])
            {
                vx.push_back(j);
            }
        }
        LR.push_back(vx);
    }
    return LR;
}

void split(const string &s, char delim, vector<string> &elems)
{
    stringstream ss(s);
    string item;
    while (getline(ss, item, delim))
    {
        elems.push_back(item);
    }
}

vector<string> split(const string &s, char delim)
{
    vector<string> elems;
    split(s, delim, elems);
    return elems;
}

#endif /* GLOBAL_H */

```

8.3. Файл с тестами

```

#include "Utils.h"
#include "SCCResult.h"
#include "StrongConnectedComponents.h"
#include "ElementaryCyclesSearch.h"
#include <time.h>

int main(void)
{
    // time test variables

```

```

clock_t t;
double timeCreate, timeProcess, timePrint, s=0;
int n=0, testCount=1000;

const int N = 16;
vector<string> nodes(N);
vector < vector<bool> > adjMatrix(N, vector<bool>(N));

// initialization
for (int i = 0; i < N; i++)
{
    nodes[i] = "Node " + std::to_string(i);
}

string SR = "1 5 4\n2 6 5 4 8\n3 7 6 10 9 13 12\n6 10 9\n7 11 15 14 13\n8 4 5 6
2\n9 10 6\n11 15 14\n12 8 9 10 11 7 3\n13 9 10 6 7\n14 10 11";
// "1 2 3\n2 3 1\n3 1 2\n3 2";
// "1 2\n3 2\n3 4 5 6 7\n4 5 6 7 3\n5 6 3\n6 3\n6 7 3"
// "1 5 4\n2 6 5 4 8\n3 7 6 10 9 13 12\n6 10 9\n7 11 15 14 13\n8 4 5 6 2\n9 10
6\n11 15 14\n12 8 9 10 11 7 3\n13 9 10 6 7\n14 10 11";

vector<string> lines = split(SR, '\n');

// Creating an Adjacency Matrix
for (int itime=0; itime<testCount; itime++){
    t=0;
    t=clock();
    for (int i = 0; i < lines.size(); i++)
    {
        vector<string> components = split(lines[i], ' ');

        for (int i = 0; i < components.size() - 1; i++)
        {
            int x = atoi(components[i].c_str());
            int y = atoi(components[i + 1].c_str());
            adjMatrix[x][y] = true;
        }
    }
    t=clock()-t;
    s+=t;
}
timeCreate = (s/testCount)*0.017;

// Run algorithm
for (int itime=0; itime<testCount; itime++){
    t=0;
    t=clock();
    ElementaryCyclesSearch ecs = ElementaryCyclesSearch(adjMatrix, nodes);
    vector<vector<string> > cycles = ecs.getElementaryCycles();
    t=clock()-t;
    s+=t;
}
timeProcess = (s/testCount)*0.017;

ElementaryCyclesSearch ecs = ElementaryCyclesSearch(adjMatrix, nodes);
vector<vector<string> > cycles = ecs.getElementaryCycles();

// Print result
for (int itime=0; itime<testCount; itime++)
{
    t=0;
    t=clock();

```

```

    for (int i = 0; i < cycles.size(); i++)
    {
        vector<string> cycle = cycles[i];
        for (int j = 0; j < cycle.size(); j++)
        {
            string node = cycle[j];
            if (j < cycle.size() - 1)
            {
                cout << node << " -> ";
            }
            else
            {
                cout << node;
            }
        }
        cout << endl;
    }
    t=clock()-t;
    s+=t;
}
timePrint = (s/testCount)*0.017;

cout << "\n=====\\n" << endl;
cout << "Creating an Adjacency Matrix time:" << endl;
cout << timeCreate << endl;
cout << "Run algorithm time:" << endl;
cout << timeProcess << endl;
cout << "Print result time:" << endl;
cout << timePrint << endl;
cout << "=====\\n" << endl;

return 0;
}

```

8.4. Программный файл для отображения графа

```

import matplotlib.pyplot as plt
import networkx as nx

# create a directed multi-graph
G = nx.MultiDiGraph()
rad = 0

G.add_edges_from([
    (0, 1),
    (1, 5),
    (2, 6),
    (3, 7),
    (4, 5),
    (4, 8),
    (5, 4),
    (5, 6),
    (6, 2),
    (6, 5),
    (6, 7),
    (6, 10),
    (7, 3),
    (7, 6),
    (7, 11),
    (8, 4),
    (8, 9),

```

```
(9, 10),
(9, 13),
(10, 6),
(10, 9),
(10, 11),
(11, 7),
(11, 15),
(12, 8),
(13, 9),
(13, 12),
(14, 10),
(14, 12),
(15, 14),
])

plt.figure(figsize=(16,8)) # plotting a frame
nx.draw(G, connectionstyle=f'arc3, rad = {rad}', with_labels = True) # plotting the
graph
plt.show() # pause before exiting

G.clear()
```