**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

**СОДЕРЖАНИЕ**

[**1.** **ТЕКСТ ЗАДАНИЯ** 3](#_Toc120623982)

[**2.** **МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ В ТЕРМИНАХ ТЕОРИИ МНОЖЕСТВ** 4](#_Toc120623983)

[**3.** **ВЫБОР И ОБОСНОВАНИЕ СПОСОБА ПРЕДСТАВЛЕНИЯ ДАННЫХ** 5](#_Toc120623984)

[**4.** **ОПИСАНИЕ АЛГОРИТМА И ОЦЕНКА ЕГО ВРЕМЕННОЙ СЛОЖНОСТИ** 6](#_Toc120623985)

[**5.** **НАБОР ТЕСТОВ И РЕЗУЛЬТАТЫ ПРОВЕРКИ АЛГОРИТМА НА ЭВМ** 7](#_Toc120623986)

[**6.** **ВЫВОДЫ** 8](#_Toc120623987)

[**7.** **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 9](#_Toc120623988)

[**8.** **ПРИЛОЖЕНИЕ A.** 10](#_Toc120623989)

[**8.1.** **Исходный текст программы для ЭВМ** 10](#_Toc120623990)

[**8.2.** **Файлы с тестами** 10](#_Toc120623991)

1. **ТЕКСТ ЗАДАНИЯ**
   1. **Формулировки общего задания**

Выполнить курсовую работу по предложенному преподавателем индивидуальному заданию (табл. П.2.3). Детальную постановку задачи взять из рекомендованных литературных источников. Реализовать алгоритм на языке С++ 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. **Формулировка индивидуального задания**

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

1. **МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ В ТЕРМИНАХ ТЕОРИИ МНОЖЕСТВ**
2. **ВЫБОР И ОБОСНОВАНИЕ СПОСОБА ПРЕДСТАВЛЕНИЯ ДАННЫХ**
3. **ОПИСАНИЕ АЛГОРИТМА И ОЦЕНКА ЕГО ВРЕМЕННОЙ СЛОЖНОСТИ**
4. **НАБОР ТЕСТОВ И РЕЗУЛЬТАТЫ ПРОВЕРКИ АЛГОРИТМА НА ЭВМ**
5. **ВЫВОДЫ**
6. **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**
7. П. Г. Колинько // «Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1»: пользовательские структуры данных» // Издательство: СПб.: СПбГЭТУ «ЛЭТИ», 2022г. – [64 с.]
8. Dale, Nell B. «C++ plus data stractures» // Издательство: Prentice-Hall, Inc., 2003г. – [816 с.]
9. Robert L. Kruse, Alexander J. Ryba // «Data Structures and Program Design in C++» // Издательство: Prentice-Hall, Inc., 2000г. – [717 с.]
10. **ПРИЛОЖЕНИЕ A.**
    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;

}

* 1. **Листинг программного кода h-файлов**
     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;

    /\*\*

     \* B-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];

            // Найти 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 \*/

* + 1. **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

* + 1. **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> > adjacencyList = getAdjList(nodes);

                    if (adjacencyList.size() > 0)

                    {

                        for (int j = 0; j < adjListOriginal.size(); j++)

                        {

                            if (adjacencyList[j].size() > 0)

                            {

                                return new SCCResult(adjacencyList, 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.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 связанности - 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 \*/

* + 1. **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 \*/

* 1. **Файл с тестами**