

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ГРАФЫ 1

Отчет по лабораторной работе №6

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Студент гр. 431-3

_____ А.В. Гурулёв
«__» _____ 2023 г.

Проверил: профессор кафедры АСУ, д.т.н.

_____ А.Н. Горитов
«__» _____ 2023 г.

Томск 2023

1 Задание на лабораторную работу

Напишите программу, которая в неориентированном связном графе находит сильно связные компоненты.

Для представления графа в программе использовать списки смежности.

Данные о графе вводятся из файла.

Программа должна вывести для каждой сильно связной компоненте графа множество вершин, входящих в найденную компоненту сильной связности графа.

После завершения работы с динамическими структурами данных необходимо освободить занимаемую ими память.

2 Алгоритм решения задачи

1. Инициализируем граф;
2. Передаем в метод импорта из файла, путь к файлу;
3. Открываем файл;
4. Переносим данные из файла в граф;
5. Создаем массив, где присваиваем каждому узлы начальный вес "0";
6. Запускаем рекурсивную функцию Component, куда передаем начальные данные (Начало – 1 узел; Предыдущий узел -1(узла нет); Указатель на массив с весами)
7. Получаем массив, где каждый индекс – узел, а его значения – группа компоненты;
8. Выводим компоненты;

3 Листинг программы

Для source.cpp:

```
#include <iostream>
#include <locale.h>
#include "MyGraph.h"
using namespace std;
using namespace MyTask;
int main()
{
    setlocale(LC_ALL, "rus");
    MyGraph graph;
    graph.ImportFromFile("File.txt");
    graph.Components();
}
```

```
    system("pause");  
    return 0;  
}  
cout << table.find_elem("fifteen") << endl;  
cout << endl;  
    table.print_table();  
    system("pause");  
    return 0;  
}
```

Для MyGraph.h:

```
#pragma once
#include "MyList.h"
namespace MyTask
{
    class MyGraph
    {
    private:
        MyList** edges;
        int count;
        int Component(int _start, int last, int* _way);
    public:
        MyGraph();
        ~MyGraph();
        void ImportFromFile(const char* path);
        int GetCount();
        void Components();
    };
}
```

Для MyList.h:

```
#pragma once
#include "MyQueue.h"
namespace MyTask
{
    class MyList
    {
    private:
        struct Node
        {
            int num;
            Node* next;
        };
        Node* head;
        int size;
    public:
        MyList();
        ~MyList();
        void Push(int _num);
        int Head();
        int GetSize();
        MyQueue* GetQueue();
        bool Find(int _num);
    };
}
```

Для MyQueue.h:

```
#pragma once
namespace MyTask
{
    class MyQueue
    {
    private:
        struct Node
        {
            int num;
            Node* next;
        };

        Node* head;
    public:
        MyQueue();
        ~MyQueue();
        void Push(int _num);
        int Pop();
    };
}
```

Для MyGraph.cpp:

```
#include <iostream>
#include <fstream>
#include "MyList.h"
#include "MyQueue.h";
#include "MyGraph.h"
using namespace std;
namespace MyTask
{
    MyGraph::MyGraph()
    {
        edges = nullptr;
        count = 0;
    }
    MyGraph::~MyGraph()
    {
        delete[] edges;
        count = 0;
    }
    void MyGraph::ImportFromFile(const char* path)
    {
        ifstream f(path);
        if (!f.is_open())
        {
            cout << endl << "Ошибка чтения файла" << endl;
            return;
        }
        f >> count;
        edges = (MyList**)malloc(sizeof(MyList*) * count);
        for (int i = 0; i < count; i++)
        {
```

```

        edges[i] = new MyList();
        int buf = 0;
        f >> buf;
        while (buf != -1)
        {
            edges[i]->Push(buf);
            f >> buf;
        }
    }
}

int MyGraph::GetCount()
{
    return count;
}

int MyGraph::Component(int _start, int last, int* _way)
{
    MyQueue* ways = edges[_start-1]->GetQueue();
    int next = 0;
    int myValue = _start;
    _way[_start-1] = myValue;
    while (next = ways->Pop())
    {
        if (next == last)
        {
            continue;
        }
        if (_way[next-1])
        {
            if(_way[next-1] < myValue)
            {
                myValue = _way[next-1];
            }
        }
    }
}

```



```

        }
        continue;
    }
    //Идем к следующему узлу
    int nextValue = Component(next, _start, _way);
    //Проверяем нужно ли менять вес
    if (nextValue < myValue)
    {
        myValue = nextValue;
    }
}
//Возвращаем значение
_way[_start-1] = myValue;
return myValue;
}

void MyGraph::Components()
{
    //Массив весов(и обозначение не пройденных) узлов
    int* way = (int*)malloc(sizeof(int) * count);
    for (int i = 0; i < count; i++)
    {
        way[i] = 0;
    }
    Component(1, -1, way);
    //Считаем количество компонент
    int* comp = (int*)malloc(sizeof(int) * count);
    int countComp = 0;
    for (int i = 0; i < count; i++)
    {
        bool flag = true;
        for (int j = 0; j < countComp; j++)

```

```

        {
            if (comp[j] == way[i])
            {
                flag = false;
                break;
            }
        }
        if (flag)
        {
            comp[countComp++] = way[i];
        }
    }
    //Выводим компоненты
    for (int i = 0; i < countComp; i++)
    {
        cout << endl << "Компонента " << i+1 << ":";
        for (int j = 0; j < count; j++)
        {
            if (way[j] == comp[i])
            {
                cout << " " << j+1;
            }
        }
    }
    cout << endl;
    //Чистим лишнее
    delete[] comp;
    delete[] way;
}
}

```

Для MyList.cpp:

```
#include <iostream>
#include "MyList.h"
#include "MyQueue.h"
using namespace std;
namespace MyTask
{
    MyList::MyList()
    {
        head = nullptr;
        size = 0;
    }
    MyList::~MyList()
    {
        while (head)
        {
            Node* next = head->next;
            delete[] head;
            head = next;
        }
        size = 0;
    }
    void MyList::Push(int _num)
    {
        Node* newNode = new Node();
        if (!newNode)
        {
            cout << endl << "Ошибка выделения памяти" << endl;
            return;
        }
    }
}
```

```

newNode->num = _num;
newNode->next = nullptr;
if (!head)
{
    head = newNode;
    size++;
    return;
}
Node* place = head;
while (place->next)
{
    place = place->next;
}
place->next = newNode;
size++;
}

int MyList::Head()
{
    return head->num;
}

int MyList::GetSize()
{
    return size;
}

MyQueue* MyList::GetQueue()
{
    MyQueue* QEdges = new MyQueue();
    Node* next = head->next;
    while (next)
    {
        QEdges->Push(next->num);
    }
}

```

```

        next = next->next;
    }
    return QEdges;
}
bool MyList::Find(int _num)
{
    Node* place = head;
    while (place)
    {
        if (place->num == _num)
        {
            return true;
        }
        place = place->next;
    }
    return false;
}
}

```

Для MyQueue.cpp:

```
#include <iostream>
#include <fstream>
#include "MyQueue.h"
using namespace std;
namespace MyTask
{
    MyQueue::MyQueue()
    {
        head = nullptr;
    }
    MyQueue::~MyQueue()
    {
        while (head)
        {
            Node* next = head->next;
            delete[] head;
            head = next;
        }
    }
    void MyQueue::Push(int _num)
    {
        Node* newNode = new Node();
        if(!newNode)
        {
            cout << endl << "Ошибка выделения памяти" << endl;
            return;
        }
        newNode->num = _num;
        newNode->next = nullptr;
```

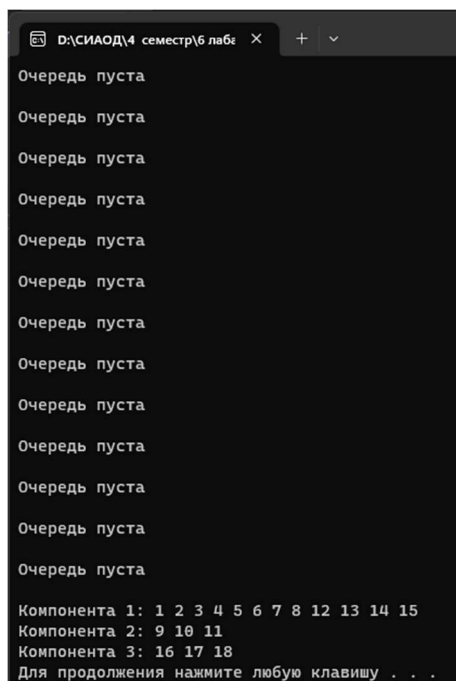
```

        if (!head)
        {
            head = newNode;
            return;
        }
        Node* place = head;
        while (place->next)
        {
            place = place->next;
        }
        place->next = newNode;
    }
    int MyQueue::Pop()
    {
        if (!head)
        {
            cout << endl << "Очередь пуста" << endl;
            return 0;
        }
        int out = head->num;
        Node* next = head->next;
        delete[] head;
        head = next;
        return out;
    }
}

```

4 Пример решения

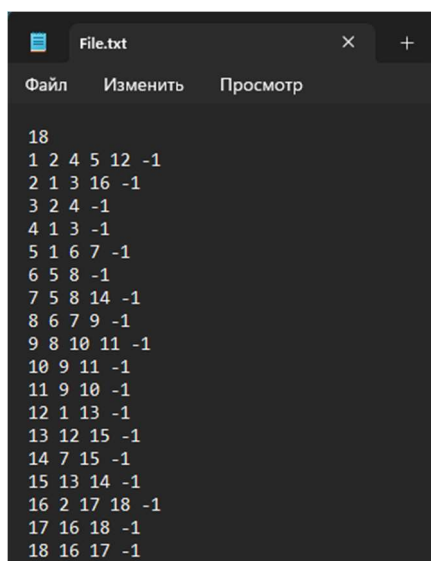
На рисунке 4.1 можно увидеть пример результата программы. Вывод состоит из сообщений о конце очереди(вызываются в процессе), а так же из списка компонент и их элементов.



```
ФАСИАОД\4 семестр\6 лаба
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Очередь пуста
Компонента 1: 1 2 3 4 5 6 7 8 12 13 14 15
Компонента 2: 9 10 11
Компонента 3: 16 17 18
Для продолжения нажмите любую клавишу . . .
```

Рисунок 4.1 - Результат выполнения программы

На рисунке 4.2 можно увидеть входные данные файла.



```
File.txt
Файл  Изменить  Просмотр
18
1 2 4 5 12 -1
2 1 3 16 -1
3 2 4 -1
4 1 3 -1
5 1 6 7 -1
6 5 8 -1
7 5 8 14 -1
8 6 7 9 -1
9 8 10 11 -1
10 9 11 -1
11 9 10 -1
12 1 13 -1
13 12 15 -1
14 7 15 -1
15 13 14 -1
16 2 17 18 -1
17 16 18 -1
18 16 17 -1
```

Рисунок 4.2 - Входные данные

5 Вывод

Я изучил как устроен АТД “Граф”, а именно способы его представления в программе, обходы, а так же, как можно обнаружить сильно связанные компоненты в неориентированном графе.