|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Интеллектуальные системы информационной безопасности»

**ОТЧЕТ   
о выполнении лабораторной работы №2**

**«Графы» по дисциплине   
«Алгоритмы и структуры данных»**

**Вариант № 97**

Выполнил: студент 2 курса

группы БСБО-06-20

шифр: 20Б1097

Халиков Александр Александрович  
*(фио студента)*

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

Москва 2021 г.

**Задание на лабораторную работу № 2.**

**Вариант № 97.**

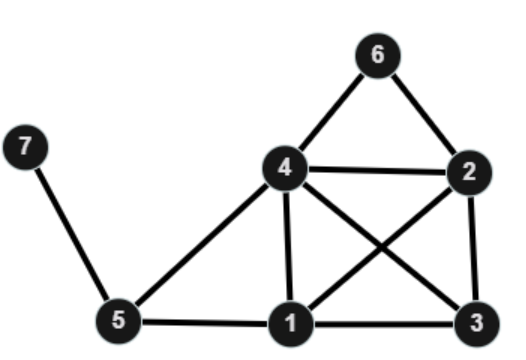
**Алгоритм:** Методом обхода в глубину вычислить цикломатическую сложность

графа

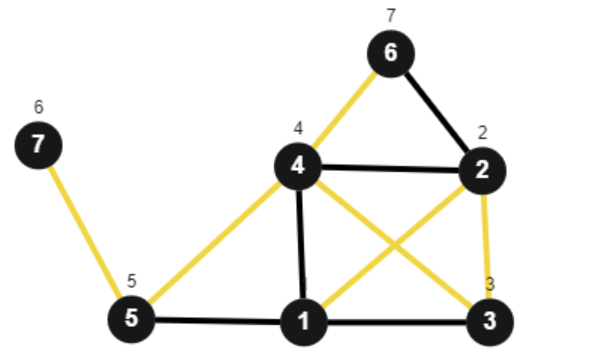
**Способ представления графа:** Список дуг

1. **Нарисованый граф с выполненным алгоритмом**

**Нарисованный граф:**



**Выполнение алгоритма поиска в глубину:**



Получается последовательность вида: 1->2->3->4->5->7->6. Это если алгоритм начинать с первой вершины. Чтобы найти цикломатическое число графа необходимо знать количество вершин, поэтому каждый раз прибавляем к нашему счетчику вершин +1.

Далее, каждый раз когда мы находим вершину и записываем ее, надо убавить счетчик дуг на один(чтобы не проходить два раза по одной дуге), а когда находим дугу счетчик увеличивается на один.

После подсчета по формуле для нахождения цикломатического числа графа получаем: 12+1-7=6 (12 дуг, 1 граф, 7 вершин). Получаем сколько ребер на графе нужно удалить, чтобы в нем не осталось ни одого цикла.

**Способ представления графа (список дуг)**

 Список дуг представлен списком элементов, каждый из которых содержит два целых числа— номера концов дуги. В данной реализации список не представлен в виде отдельного описания класса.

Операция добавления дуги в этом представлении — это просто операция добавления одного элемента в конец списка, и реализуется она сравнительно эффективно. Напротив, при выполнении операции поиска дуги может потребоваться просмотреть весь список. Разумеется, настолько же неэффективной будет и операция удаления дуги.

Граф сверху в программе заполняется так (добавляются все ребра и в одну и в другую строну, чтобы получилась дуга, так как граф неориентирован):

ArcGraph g = ArcGraph(8);

//добавление ребер

g.addArc(1, 2);

g.addArc(1, 3);

g.addArc(1, 4);

g.addArc(1, 5);

g.addArc(2, 3);

g.addArc(2, 1);

g.addArc(2, 4);

g.addArc(2, 6);

g.addArc(3, 2);

g.addArc(3, 1);

g.addArc(3, 4);

g.addArc(4, 6);

g.addArc(4, 2);

g.addArc(4, 3);

g.addArc(4, 1);

g.addArc(4, 5);

g.addArc(4, 7);

g.addArc(5, 1);

g.addArc(5, 4);

g.addArc(5, 7);

g.addArc(6, 2);

g.addArc(6, 4);

g.addArc(7, 4);

g.addArc(7, 5);

**Код программы**

#include <iostream>

// Дуга представлена элементом списка, содержащим номера

// концов дуги и указатель на следующий элемент списка

struct Arc {

int begin, end;

Arc\* next;

// Конструктор дуги.

Arc(int b, int e, Arc\* n=nullptr)

{

begin = b; end = e; next = n;

};

};

class ArcGraph {

public:

// Список дуг представлен, как обычно, указателями на первый и последний элементы списка

Arc\* first, \* last;

int cyclomaticNubmer;

int numberOfArcs;

int arcs;

// arcCount - счетчик числа дуг-элементов списка

int arcCount;

// vertexNumber - количество вершин графа, используемое в данном представлении только для контроля номеров вершин

int vertexNumber;

// Конструктор графа инициализирует пустой список и запоминает количество вершин графа.

ArcGraph(int n)

{

first = last = nullptr;

arcCount = 0;

vertexNumber = n;

}

// Функция подсчета количества вершин выдает запомненное значение

int vertexCoun() const { return vertexNumber-1; }

// Операции над дугами:

void addArc(int from, int to);

bool hasArc(int from, int to) const;

void getGraph();

int DFS(int start,int n);

int getNubmerOfArc();

};

//Реализация операции добавления дуги

void ArcGraph::addArc(int from, int to) {

// Сначала проверяем правильность задания номеров вершин.

if (from < 0 || to < 0 || from >= vertexNumber || to >= vertexNumber)

return;

Arc\* newArc = new Arc(from, to);

// Новая дуга добавляется в конец списка

if (last) last->next = newArc;

else first = newArc;

last = newArc;

arcCount++;

}

// Реализация операции проверки наличия дуги.

bool ArcGraph::hasArc(int from, int to) const {

// Необходимо просмотреть список дуг в поисках нужной.

for (Arc\* current = first; current; current = current->next) {

if (current->begin == from && current->end == to)

{

return true;

}

}

return false;

}

void ArcGraph::getGraph()

{

for (Arc\* current = first; current; current = current->next)

{

std::cout << current->begin << " " << current->end << std::endl;

}

}

int ArcGraph::DFS(int start,int n)

{

static int visited[8] = { 0 };

int j;

if (visited[start] == 0)

{

printf("%d", start);

visited[start] = 1;

for (j = 1;j < n;j++)

{

//numberOfVertex++;

if (hasArc(start, j) && visited[j] == 0)

{

numberOfArcs--;

DFS(j, n);

}

else if (hasArc(start, j))

{

numberOfArcs++; //Счетчик скролько всего ребер

}

}

}

return numberOfArcs+1-vertexCoun(); //

(наименьшее число рёбер, которое нужно удалить из данного графа, чтобы получить дерево (для связного графа))

}

int main()

{

ArcGraph g = ArcGraph(8);

//добавление ребер(дуг)

g.addArc(1,2);

g.addArc(1,3);

g.addArc(1,4 );

g.addArc(1, 5);

g.addArc(2, 3);

g.addArc(2, 1);

g.addArc(2, 4);

g.addArc(2, 6);

g.addArc(3, 2);

g.addArc(3, 1);

g.addArc(3, 4);

g.addArc(4, 6);

g.addArc(4, 2);

g.addArc(4, 3);

g.addArc(4, 1);

g.addArc(4, 5);

g.addArc(4, 7);

g.addArc(5, 1);

g.addArc(5, 4);

g.addArc(5, 7);

g.addArc(6, 2);

g.addArc(6, 4);

g.addArc(7, 4);

g.addArc(7, 5);

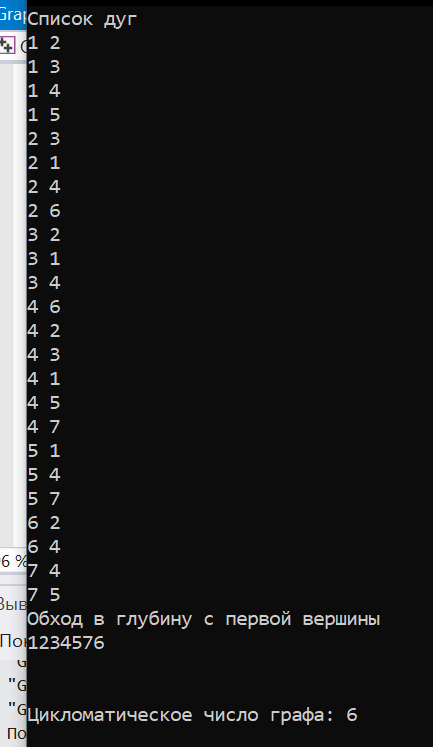
g.getGraph();

std::cout << "\n\n\n" << g.DFS(5, 8);

return 0;

}

**Скриншот работы программы**

****