Балтийский государственный технический университет
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра И5 «Информационные системы и программная инженерия»

Практическая работа №2
по дисциплине «Структуры и организация данных»
на тему «Нелинейные структуры данных»

Выполнил:
Студент Альков В.С.
Группа Номер_группы

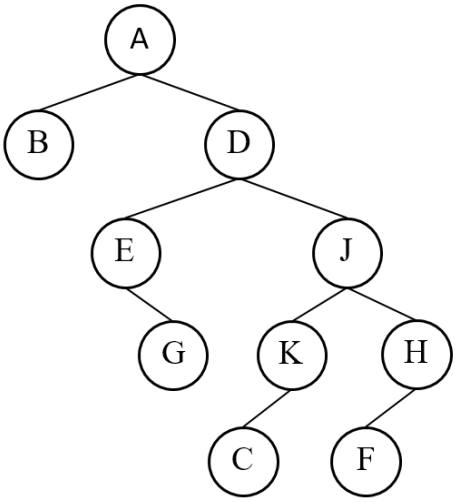
Преподаватель:
Полухин А.Л.

Санкт-Петербург
2021 г.

Задание 1.

В прямом порядке обхода бинарного дерева была получена последовательность узлов ABDEGJKCHF, а в симметричном - BAEGDCKJFH. Какова последовательность обхода этого дерева по уровням?

Полученное дерево:



Ответ: A;BD;EJ;GKH;CF

Задание 2.

Вопрос 1

Верно

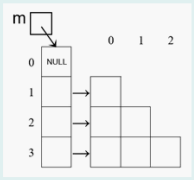
Баллов: 3,00 из 3,00

Отметить вопрос

В некоторой библиотеке есть два класса, в которых определены одинаковые операции для работы с неориентированным взвешенным графом.

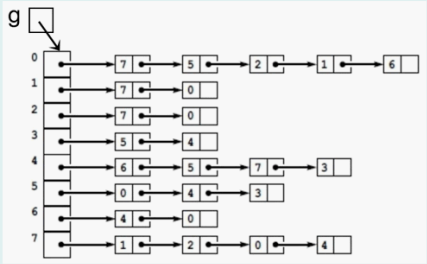
В первом классе граф представлен матрицей весов, структура хранения которой – треугольная динамическая матрица, схематичное изображение которой приведено на рисунке. Элементы матрицы могут принимать целые неотрицательные значения, указатель m объявлен так:

```
unsigned int ** m;
```



Во втором классе граф представляется списками смежности, структура хранения каждого из которых – односвязный линейный список, значением поля данных каждого элемента является номер вершины и вес дуги. Указатели на головы списков размещаются в массиве. Схематичное изображение структуры хранения в целом приведено на рисунке ниже. Указатель g объявлен так:

```
struct node
{
    short int number_node;
    unsigned int weight;
    node* next;
} **g;
```



Укажите минимальный порядок графа, для хранения которого в объекте второго класса будет выделяться гарантированно меньше памяти, чем в объекте первого класса, если степень любой вершины не будет превышать пяти, sizeof (int) = 4 байта, размер каждого указателя - 8 байт, а поля структуры располагаются вплотную друг к другу.

Ответ: 37

Проверить

Решение задачи:

Представление графа с помощью матрицы весов.

n – кол-во вершин графа;

m – указатель на массив указателей, указатель на строки матрицы, занимает 8 байт;

*m – указатель на массив элементов типа unsigned int, указатель на строку матрицы, занимает 8 байт;

i * j - размер полной матрицы, где i – число строк, j – число столбцов.

В нашем случае у полной матрицы i = j = n.

Треугольная матрица состоит из половины полной и не хранит главную диагональ, то есть путь из вершины в саму себя. Главная диагональ состоит из n элементов. В итоге размер треугольной матрицы = (n*n-n)/2 = n*(n-1) / 2. Каждый элемент матрицы имеет тип unsigned int и занимает 4 байта, следовательно, элементы матрицы занимают n*(n-1) / 2 * 4 байт.

Также память занимает указатель на все строки матрицы (m) и n указателей на каждую строку матрицы(*m). **Следовательно, граф занимает в памяти 8 + 8*n+n(n-1)/2*4 байт.**

Представление графа с помощью списков смежности.

n – кол-во вершин графа;

g – указатель на массив указателей, занимает 8 байт;

*g – указатель на голову списка, занимает 8 байт;

Элемент списка – структурный тип с полями: short int - занимает 2 байта; unsigned int – занимает 4 байта, указатель – занимает 8 байт. Следовательно, каждый элемент списка занимает 2+4+8 = 14 байт. Максимальное кол-во элементов в списке = 5, так как степень вершины не превышает 5 по условию. Следовательно, список занимает 5*14 = 70 байт. Количество списков, как и указателей на голову списка, n штук, поэтому в общем они занимают n*(8+70) = n*78 байт.

В итоге граф занимает 8 + n*78 байт.

Поиск минимального порядка графа

При n = 1:

первое представление занимает 8+8*1+1(1-1)/2*4 = 16 байт;

второе представление занимает 8+1*78 = 86 байт.

При n = 2:

первое представление занимает 8+8*2+2(2-1)/2*4 = 28 байт;

второе представление занимает 8+2*78 = 164 байт.

При n = 36:

первое представление занимает 8+8*36+36(36-1)/2*4 = 2816 байт;

второе представление занимает 8+36*78 = 2816 байт.

При n = 37:

первое представление занимает 8+8*37+37(37-1)/2*4 = 2968 байт;

второе представление занимает 8+37*78 = 2894 байт.

Ответ: 37

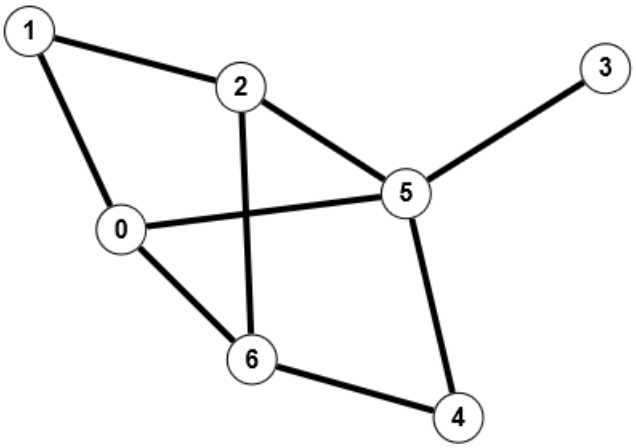
Задание 3. Написать программы для решения задачи.

Уровень сложности – **повышенный**. Граф представляется двумя способами (матрицей смежности или весов и списками смежности), для тестирования программы требуется создать файлы с описанием графа одним способом (только матрицей или только списками), при считывании данных из файла заполнять параллельно обе структуры хранения (и реализующую представление матрицы смежности (весов), и реализующую представление списков смежных вершин). При выборе структур хранения руководствоваться требованием разумной экономии памяти.

Найти диаметр графа, т.е. максимум расстояний между всеми парами его вершин.

Тестовый граф:

1)



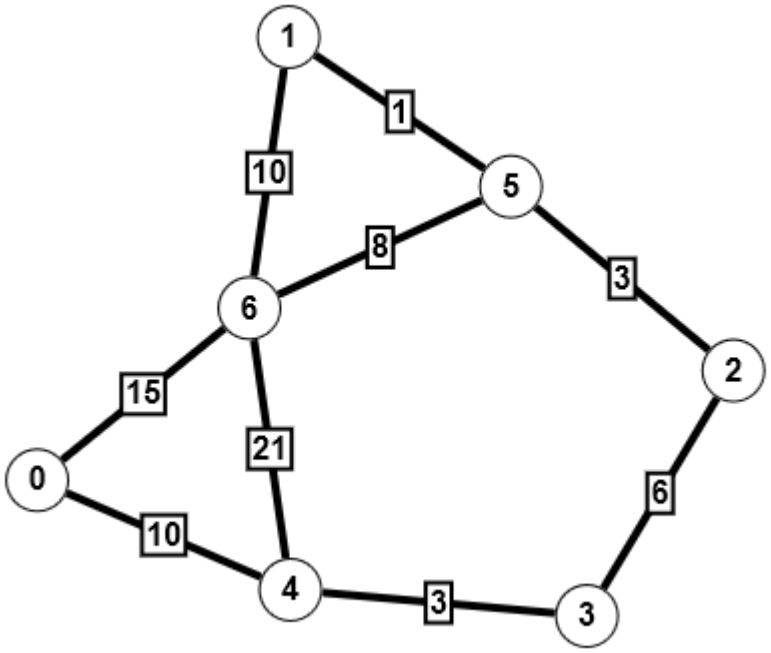
Матрица смежности							
	0	*1*	*2*	*3*	*4*	*5*	*6*
0	0	1	0	0	0	1	1
1	1	0	1	0	0	0	0
2	0	1	0	0	0	1	1
3	0	0	0	0	0	1	0
4	0	0	0	0	0	1	1
5	1	0	1	1	1	0	0
6	1	0	1	0	1	0	0

Списки смежности	
0	1 5 6
1	0 2
2	1 5 6
3	5
4	5 6
5	0 2 3 4
6	0 2 4

Матрица расстояний							
	0	*1*	*2*	*3*	*4*	*5*	*6*
0	0	1	2	2	2	1	1
1	1	0	1	3	3	2	2
2	2	1	0	2	2	1	1
3	2	3	2	0	2	1	3
4	2	3	2	2	0	1	1
5	1	2	1	1	1	0	2
6	1	2	1	3	1	2	0

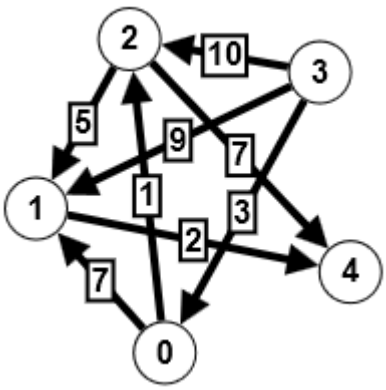
Диаметр
3

2)



Матрица смежности								Списки смежности				Матрица расстояний								Диаметр
	0	*1*	*2*	*3*	*4*	*5*	*6*	0	4(10)	6(15)			*0*	*1*	*2*	*3*	*4*	*5*	*6*	23
0	0	0	0	0	10	0	15	*1*	5(1)	6(10)		*0*	0	23	19	13	10	22	15	
1	0	0	0	0	0	1	10	*2*	3(6)	5(3)		*1*	23	0	4	10	13	1	9	
2	0	0	0	6	0	3	0	*3*	2(6)	4(3)		*2*	19	4	0	6	9	3	11	
3	0	0	6	0	3	0	0	*4*	0(10)	3(3)	6(21)	*3*	13	10	6	0	3	9	17	
4	10	0	0	3	0	0	21	*5*	1(1)	2(3)	6(8)	*4*	10	13	9	3	0	12	20	
5	0	1	3	0	0	0	8	*6*	0(15)	1(10)	4(21)	5(8)	*5*	22	1	3	9	12	0	8
6	15	10	0	0	21	8	0						*6*	15	9	11	17	20	8	0	

3)



Матрица смежности	Списки смежности	Матрица расстояний	Диаметр

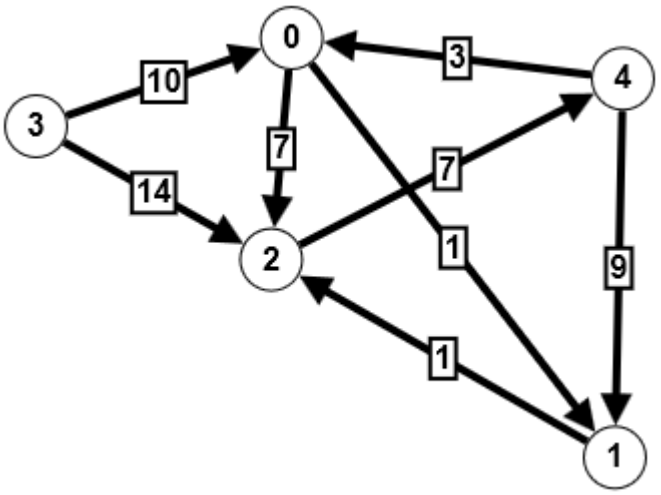
	0	1	2	3	4
0	0	7	1	0	0
1	0	0	0	0	2
2	0	5	0	0	7
3	3	9	10	0	0
4	0	0	0	0	0

0	1(7)	2(1)
1	4(2)	
2	1(5)	4(7)
3	0(3)	1(9) 2(10)
4		

	0	1	2	3	4
0	∞	6	1	∞	8
1	∞	∞	∞	∞	2
2	∞	5	∞	∞	7
3	3	9	4	∞	11
4	∞	∞	∞	∞	∞

11

4)



Матрица смежности					
	0	1	2	3	4
0	0	1	7	0	0
1	0	0	1	0	0
2	0	0	0	0	7
3	10	0	14	0	0
4	3	9	0	0	0

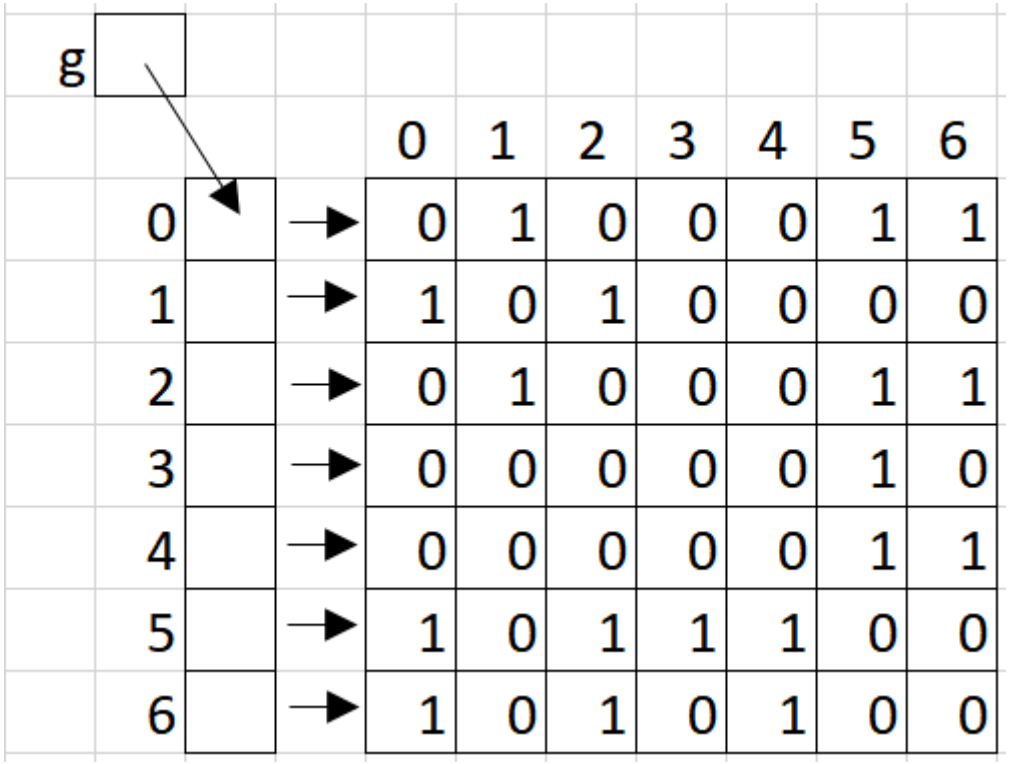
Списки смежности		
0	1(1)	2(7)
1	2(1)	
2	4(7)	
3	0(10)	2(14)
4	0(3)	1(9)

Матрица расстояний					
	0	1	2	3	4
0	0	1	2	∞	9
1	11	0	1	∞	8
2	10	11	0	∞	7
3	10	11	12	∞	19
4	3	4	5	∞	0

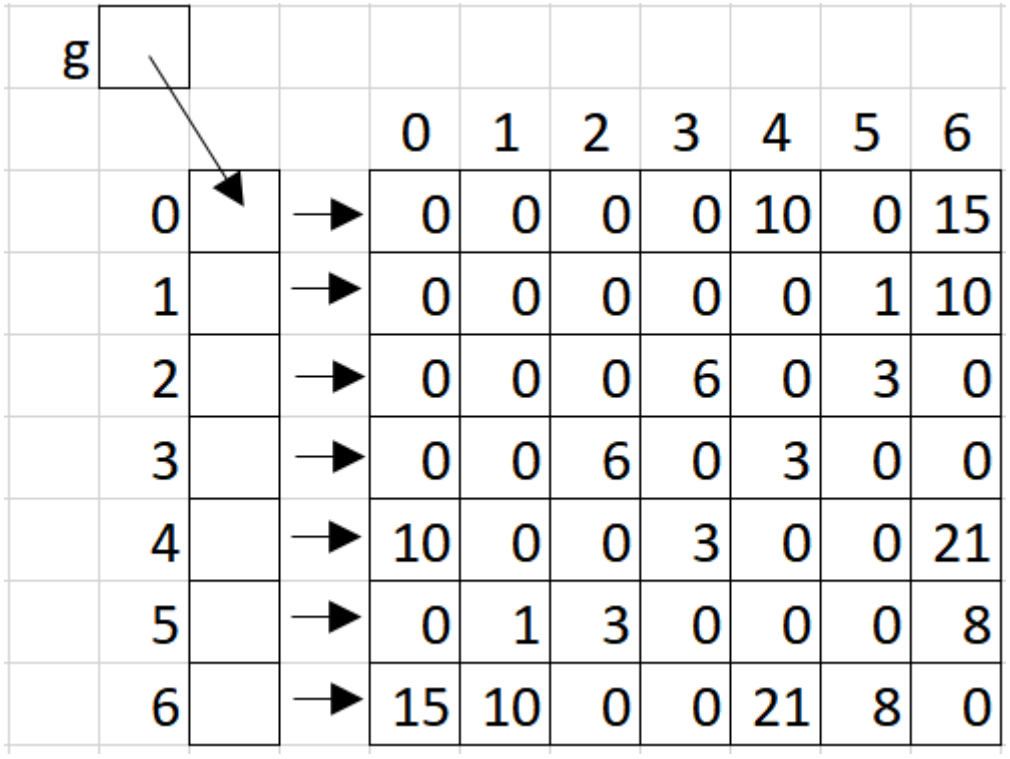
Диаметр: 19

Структура хранения для представления графа матрицей смежности или весов: Динамическая матрица. Так как матрица динамическая, нет ограничения на кол-во вершин, можно при необходимости добавить вершину, перевыделив память под матрицу большего размера и перенеся данные в нее из прошлой. Реализация графа с помощью матрицы смежности предполагает, что нужно хранить все связи между вершинами, то есть даже если связи между двумя вершинами нет, нужно хранить 0. Поэтому, используя структуру хранения список, занималось бы много памяти под хранение указателей на нулевые ячейки и из нулевых ячеек на следующую.

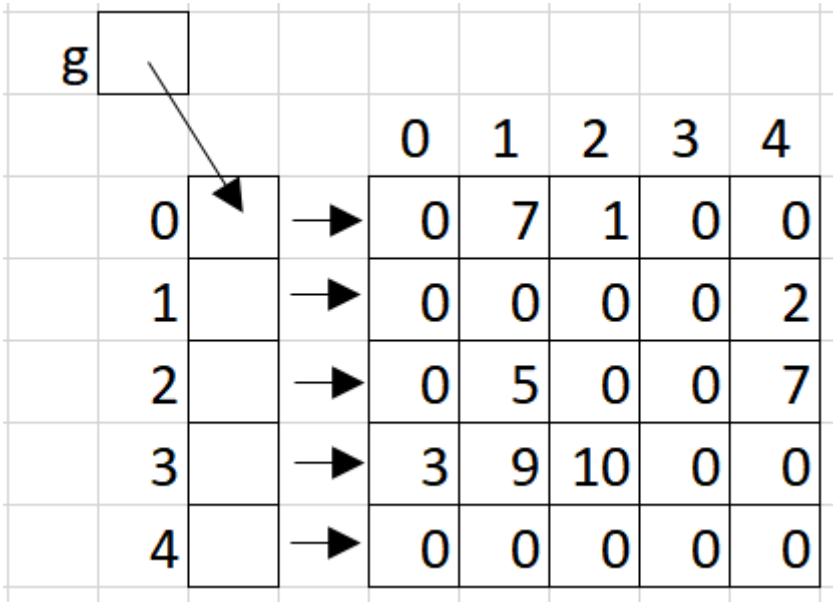
Тестовый граф 1


g
0 1 2 3 4 5 6
0 0 1 0 0 0 1 1
1 1 0 1 0 0 0 0
2 0 1 0 0 0 1 1
3 0 0 0 0 0 1 0
4 0 0 0 0 0 1 1
5 1 0 1 1 1 0 0
6 1 0 1 0 1 0 0

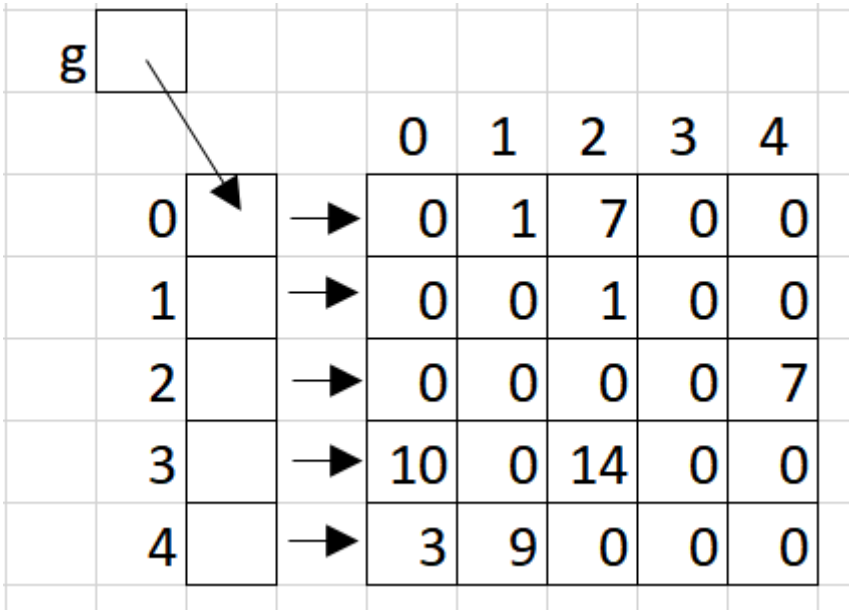
Граф 2


g
0 1 2 3 4 5 6
0 0 0 0 0 10 0 15
1 0 0 0 0 0 1 10
2 0 0 0 6 0 3 0
3 0 0 6 0 3 0 0
4 10 0 0 3 0 0 21
5 0 1 3 0 0 0 8
6 15 10 0 0 21 8 0

Граф 3



Граф 4

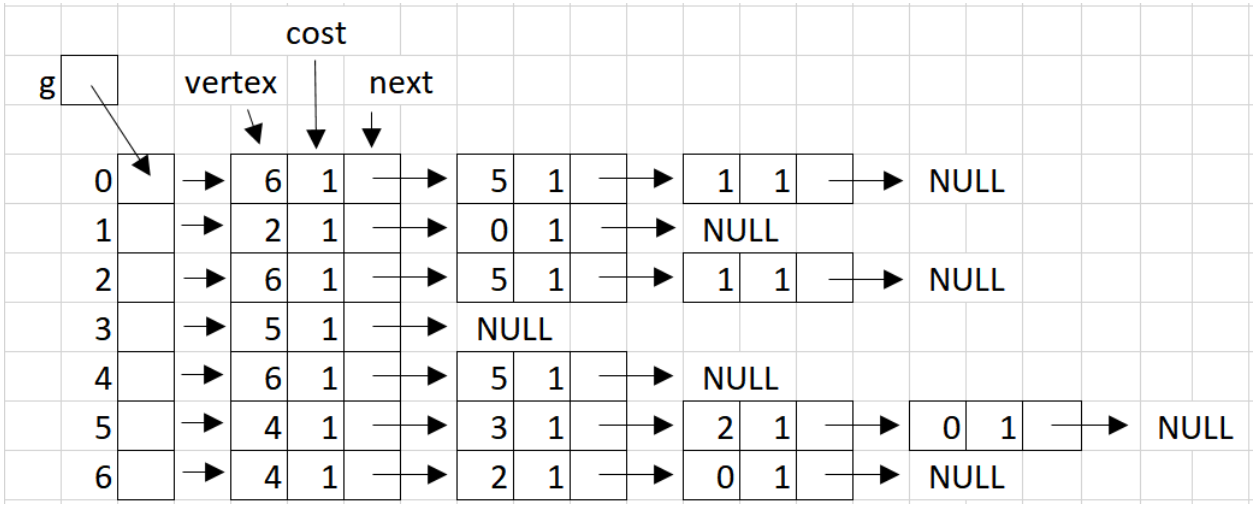


Структура хранения для представления графа списками смежных вершин:

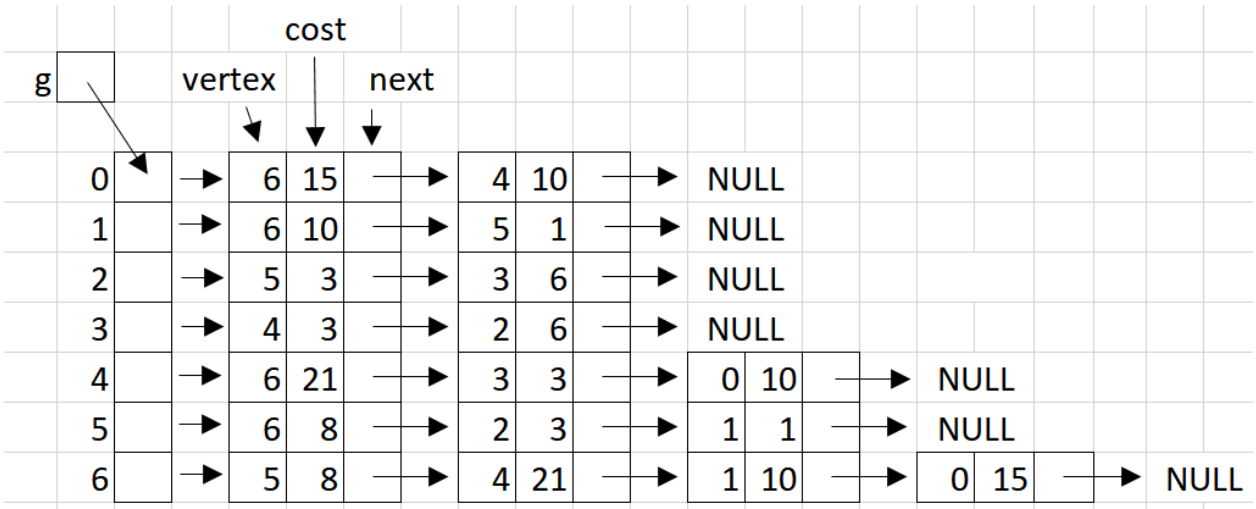
Динамический массив списков.

Так как массив динамический можно добавлять вершины. Реализация графа списком смежных вершин предполагает хранение индекса вершины смежной с данной и стоимость пути. Нет необходимости хранить сведение о том, что связи между вершинами нет в виде 0, как это надо в матрице смежности.

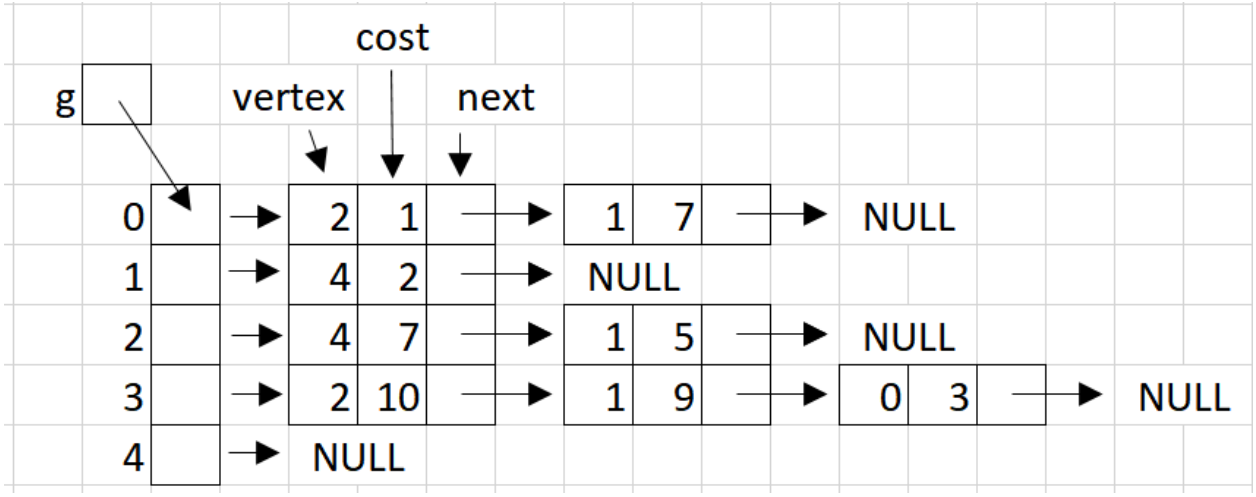
граф 1



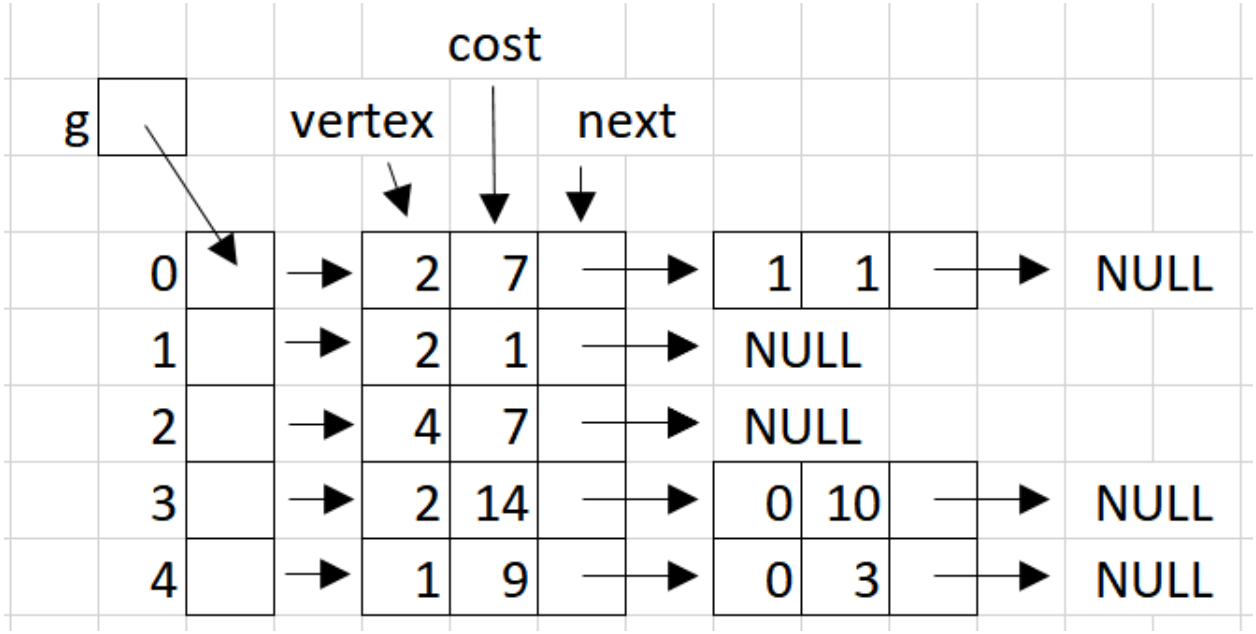
граф 2



граф 3



граф 4



Применяемые алгоритмы:

1) Нахождение эксцентриситета веришины, используя Алгоритм Дейкстры – поиск кратчайшего пути между вершиной и всеми остальными вершинами.

Входные данные: граф, индекс вершины.

Выходные данные: эксцентриситет вершины, максимальное кратчайшее расстояние между одной вершиной со всеми остальными.

2) Нахождение диаметра – максимальный эксцентриситет.

Входные данные: граф;

Выходные данные: диаметр.

Реализация графа матрицей смежности

```
#include <iostream>

class Graph1
{
    int **g, nodeCount;
public:
    Graph1(); //конструктор
    ~Graph1(); //деструктор
    void print(); //печать матрицы смежности
    void addNode(); //добавление вершины
    void addEdge(int vertex1, int vertex2, int cost); //добавление связяи
    int getEccentricity(int start); //эксцентриситет
    void getDiameter(); //диаметр

};

Graph1::Graph1()
{
//создание пустого графа
    g=NULL;
    nodeCount=0;
}
```

```
void Graph1::addNode()
{
//увеличиваем число вершин в графе
    nodeCount++;
    if(!g) //если граф пустой
    {
      //то выделяем память под одну вершину, связей у нее нет
        g=new int*[nodeCount];
        *g = new int[nodeCount];
        **g = 0;
    }
    else //в другом случае
        {
            //выделяем память под новое кол-во вершин
            int **temp = new int*[nodeCount], i, j;

            for(i=0; i<nodeCount-1; i++)
            {
                temp[i] = new int[nodeCount];
                //копируем матрицу смежности
                memcpy(temp[i], g[i], (nodeCount-1)*sizeof(int));
                //устанавливая новые связи на 0
                temp[i][nodeCount-1] = 0;
                //очищаем прошлую строку матрицы смежности
                delete[] g[i];
            };
            //освобождаем указатель на матрицу смежности
            delete[] g;
            //выделяем память под оставшеюся строку
            temp[i] = new int[nodeCount];
            //обнуляем связи
            for(j=0; j<nodeCount; j++)
                temp[i][j] = 0;
            //переставляем указатель на новую матрицу смежности
            g = temp;
        }
}

void Graph1::addEdge(int vertex1, int vertex2, int cost)
{
//проверка существования вершины
    if(vertex1<nodeCount&&vertex2<nodeCount)
//устанавливаем связь
        g[vertex1][vertex2] = cost;
}

Graph1::~Graph1()
{
//если граф не пуст
    if(g)
    {
        for(int i=0; i<nodeCount; i++)
        //освобождаем строки
            delete[] g[i];
        //освобождаем указатель на строки
        delete[] g;
    };
}


void Graph1::print()
{
    for(int i=0; i<nodeCount; i++)
```

```
        {
            for(int j=0; j<nodeCount; j++)
                printf("%-5d ", g[i][j]);
            std::cout<<"\n";
        };
    }

    int Graph1::getEccentricity(int start)
    {
    //distance – массив расстояний до вершин от заданной
        int distance[nodeCount], i, j, min, minIndex, res=0;
    //массив посещений вершин
        bool nodesVisit[nodeCount];
        for (i = 0; i < nodeCount; i++)
        {
            //задаем максимальное расстояние до каждой вершины
            distance[i] = INT_MAX;
            //ставим все вершины не посещенными
            nodesVisit[i] = false;
        };
        //ставим расстояние до самой себя равной 0
        distance[start] = 0;
        for (i = 0; i < nodeCount - 1; i++)
        {
            for (j = 0,  min = INT_MAX; j < nodeCount; j++)
                //находим непосещенную вершину с минимальным расстоянием
                if (!nodesVisit[j] && distance[j] <= min)
                {
                    min = distance[j];
                    minIndex = j;
                };
          //отмечаем ее посещенной
            nodesVisit[minIndex] = true;
            for (j = 0; j < nodeCount; j++)
                if (!nodesVisit[j] && g[minIndex][j] && distance[minIndex] !=
INT_MAX)
                    //если путь в вершину j через вершину minIndex короче
чем найденный раньше, то запоминаем его как минимальный
                    distance[j] = std::min(distance[j], distance[minIndex] +
g[minIndex][j]);
        }
        //находим максимальной расстояние
        for(i = 0; i<nodeCount; i++)
        {
            if(distance[i]!=INT_MAX&&distance[i]>res)
                res = distance[i];
        };
        return res;
    }

    void Graph1::getDiameter()
    {
        int res = 0;
    //находим максимальное расстояние из максимальных кратчайших каждой
вершины
        for(int i = 0; i<nodeCount; i++)
            res = std::max(res, this->getEccentricity(i));
        std::cout<<res<<"\n";
    }
```

Реализация графа списками смежности

```
    #include <iostream>
```

```
class Graph2;
//элемент списка
class Node
{
    int vertex, cost;
    Node* next;
public:
    Node(int x, int y, Node* z); //конструктор с параметрами, x – смежная
вершина, y – путь, z – след. элемент списка
    ~Node()
    {
        delete this->next;
    };
    //оператор для получения длины пути по индеку вершины
    int operator[](int x);
    friend Graph2;
};

Node::Node(int x, int y, Node* z)
{
    vertex = x;
    cost = y;
    next = z;
};

int Node::operator[](int x)
{
    Node* temp = this;
    int res = 0;
    while(temp)
    {
            //находи нужную вершину
        if(x==temp->vertex)
            res = temp->cost;
        temp = temp->next;
    };
    return res;
};

class Graph2
{
    Node** g;
    int nodeCount;
public:
    Graph2();
    ~Graph2();
    void print();
    void addNode();
    void addEdge(int vertex1, int vertex2, int cost);
    int getEccentricity(int start);
    void getDiameter();

};

Graph2::Graph2()
{
    g=NULL;
    nodeCount=0;
}


void Graph2::addNode()
{
```

```
        nodeCount++;
        //если граф пустой
        if(!g)
        {
            g=new Node*[nodeCount];
            *g = NULL;
        }
        else
            {
                //выделяем память под массив указателей на списки
                Node **temp = new Node*[nodeCount];
                int i, j;
                for(i=0; i<nodeCount-1; i++)
                    //переставляем указатели
                    temp[i] = g[i];
                delete[] g;
                temp[i] = NULL;
                g=temp;
            };
    }

    void Graph2::addEdge(int vertex1, int vertex2, int cost)
    {
        //проверка существования вершины и пути, если 0, то значит, что пути
нет
        if(vertex1<nodeCount&&vertex2<nodeCount&&cost!=0)
        {
            Node* temp = new Node(vertex2, cost, g[vertex1]);
            g[vertex1] = temp;
        };
    }

    Graph2::~Graph2()
    {
        if(g)
        {
            for(int i=0; i<nodeCount; i++)
                delete g[i];
            delete[] g;
        };
    }

    void Graph2::print()
    {
        Node* temp;
        for(int i = 0; i<nodeCount; i++)
        {
            temp = g[i];
            printf("%-2d", i);
            while(temp)
            {
                printf("%2d/%-2d ", temp->vertex, temp->cost);
                temp = temp->next;
            };
            printf("\n");
        };
    }
    //аналогично матрице смежности
    int Graph2::getEccentricity(int start)
    {
        int distance[nodeCount], i, j, min, minIndex, res=0;
        bool nodesVisit[nodeCount];
        for (i = 0; i < nodeCount; i++)
```

```
        {
            distance[i] = INT_MAX;
            nodesVisit[i] = false;
        };
        distance[start] = 0;
        for (i = 0; i < nodeCount - 1; i++)
        {
            for (j = 0,  min = INT_MAX; j < nodeCount; j++)
                if (!nodesVisit[j] && distance[j] <= min)
                {
                    min = distance[j];
                    minIndex = j;
                };
            nodesVisit[minIndex] = true;
            for (j = 0; j < nodeCount; j++)
                if (!nodesVisit[j] && (*g[minIndex])[j] && distance[minIndex]
!= INT_MAX)
                    distance[j] = std::min(distance[j], distance[minIndex] +
(*g[minIndex])[j]);
        }
        for(i = 0; i<nodeCount; i++)
        {
            if(distance[i]!=INT_MAX&&distance[i]>res)
                res = distance[i];
        };
        return res;
    }
    //аналогично матрице смежности
    void Graph2::getDiameter()
    {
        int res = 0;
        for(int i = 0; i<nodeCount; i++)
            res = std::max(res, this->getEccentricity(i));
        std::cout<<res<<"\n";
    }
```

Текст программы:

```
#include "graph1.cpp"
#include "graph2.cpp"
#include <fstream>
#include <iostream>
#include <locale.h>
#include <string.h>
int main(int argc, char* files[])
{

    int num, i=0, j=0;
    char filename[80], c;

    Graph1 a;
    Graph2 b;
    setlocale(LC_ALL, "rus");
    if(argc<2)
    {
        std::cout << "Имя файла:";
        std::cin.getline(filename, 80);
    }
    else
        strcpy(filename,files[1]);
    std::fstream f(filename);
    if(!f.is_open())
    {
```

```
            std::cout<<"Не удалось открыть файл";
            return 0;
        };
        while(f>>num)
        {
            //если первая строка, то добавляем вершины
            if(i==0)
            {
                a.addNode();
                b.addNode();
            };
            a.addEdge(i,j, num);
            b.addEdge(i,j, num);
            j++;
            while(f.get(c))
            {
                //проверка на цифру
                if(c>='0' && c<='9' || c=='-')
                {
                    f.unget();
                    break;
                }
                //разделение на строки
                if(c=='\n')
                {
                    i++;
                    j=0;
                    break;
                };
            };
        };
        f.close();
        std::cout<<"Матрица смежности:\n";
        a.print();
        std::cout<<"Списки вершин смежности:\n";
        b.print();
        std::cout<<"Диаметр:\n";
        a.getDiameter();
        b.getDiameter();
        return 0;
    }
```

Результаты работы программы:

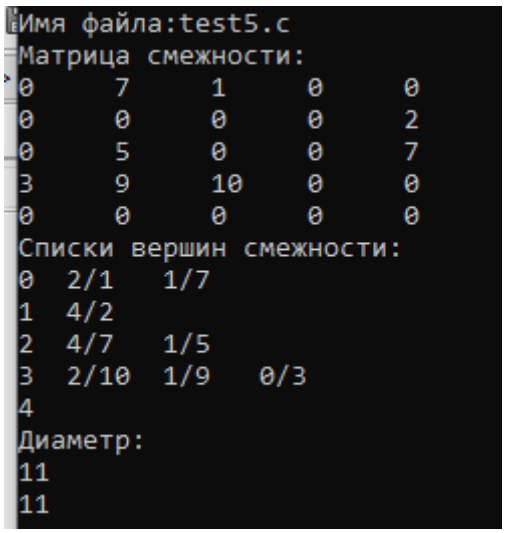
граф 1

```
Имя файла:test1.c
Матрица смежности:
0     1     0     0     0     1     1
1     0     1     0     0     0     0
0     1     0     0     0     1     1
0     0     0     0     0     1     0
0     0     0     0     0     1     1
1     0     1     1     1     0     0
1     0     1     0     1     0     0
Списки вершин смежности:
0  6/1   5/1   1/1
1  2/1   0/1
2  6/1   5/1   1/1
3  5/1
4  6/1   5/1
5  4/1   3/1   2/1   0/1
6  4/1   2/1   0/1
Диаметр:
3
3
```

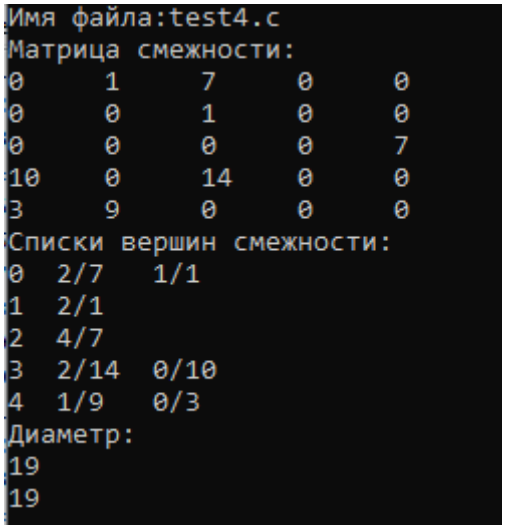
граф 2

```
Имя файла:test2.c
Матрица смежности:
0     0     0     0     10    0     15
0     0     0     0     0     1     10
0     0     0     6     0     3     0
0     0     6     0     3     0     0
10    0     0     3     0     0     21
0     1     3     0     0     0     8
15    10    0     0     21    8     0
Списки вершин смежности:
0  6/15  4/10
1  6/10  5/1
2  5/3   3/6
3  4/3   2/6
4  6/21  3/3   0/10
5  6/8   2/3   1/1
6  5/8   4/21  1/10  0/15
Диаметр:
23
23
```

граф 3


Имя файла:test5.c
Матрица смежности:
0 7 1 0 0
0 0 0 0 2
0 5 0 0 7
3 9 10 0 0
0 0 0 0 0
Списки вершин смежности:
0 2/1 1/7
1 4/2
2 4/7 1/5
3 2/10 1/9 0/3
4
Диаметр:
11
11

граф 4


Имя файла:test4.c
Матрица смежности:
0 1 7 0 0
0 0 1 0 0
0 0 0 0 7
10 0 14 0 0
3 9 0 0 0
Списки вершин смежности:
0 2/7 1/1
1 2/1
2 4/7
3 2/14 0/10
4 1/9 0/3
Диаметр:
19
19