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

Кафедра О7 «Информационные системы и программная инженерия»

Практическая работа №2
по дисциплине «Структуры и организация данных»
на тему «Нелинейные структуры данных»
вариант 5

Выполнил:
Студент Вяткин Н. А.
Группа О722Б

Преподаватель:
Гладевич А. А

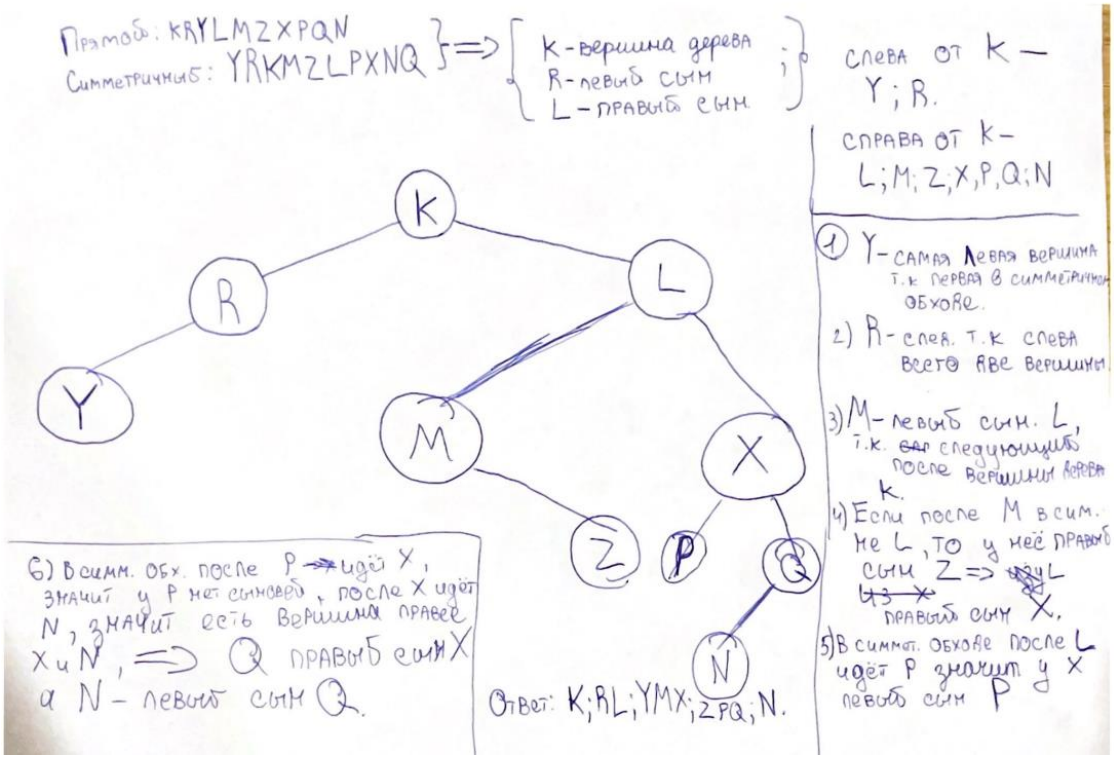
Санкт-Петербург
2023 г.

Задание 1.

В прямом порядке обхода бинарного дерева была получена последовательность узлов KRYLMZXPQN, а в симметричном - YRKMZLPXNQ. Какая последовательность узлов получится при обходе этого дерева по уровням? В строку ответа запишите последовательность посещения узлов без пробелов, разделяя разноуровневые узлы точкой с запятой, например A;BC;DEF;GHIJK.

Ответ: K;RL;YMX;ZPQ;N

Полученное дерево:



Ответ: K;RL;YMX;ZPQ;N

Задание 2.

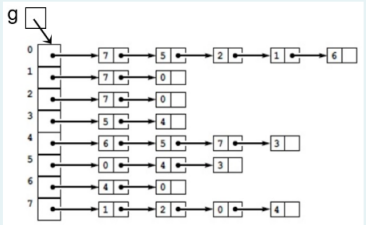
Вопрос 1
Верно
Баллов: 2,00 из 2,00
Отметить вопрос

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

В первом классе граф представлен матрицей весов, структура хранения которой – двумерный массив элементов типа unsigned int.

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;
```



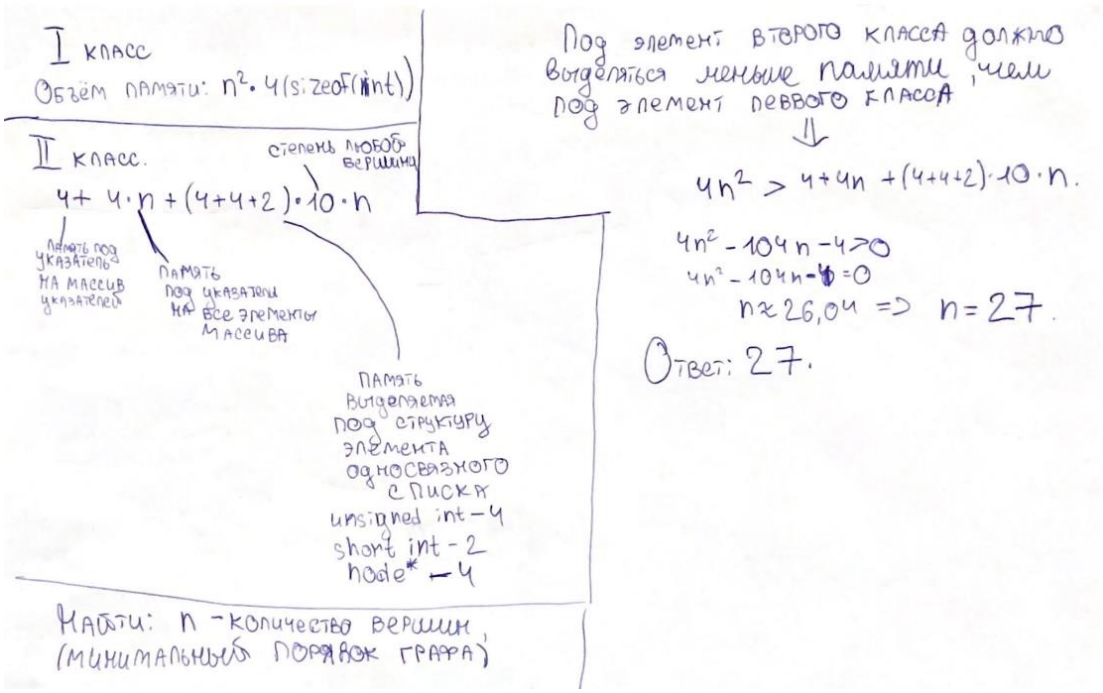
Укажите минимальный порядок графа, для хранения которого в объекте второго класса будет выделяться гарантированно меньше памяти, чем в объекте первого класса, если степень любой вершины не будет превышать десяти, sizeof(int) = 4 байта, размер каждого указателя - 4 байта, а поля структуры располагаются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

Ответ: 27

Проверить

Верно
Баллы за эту попытку: 2,00/2,00.

Решение задачи:



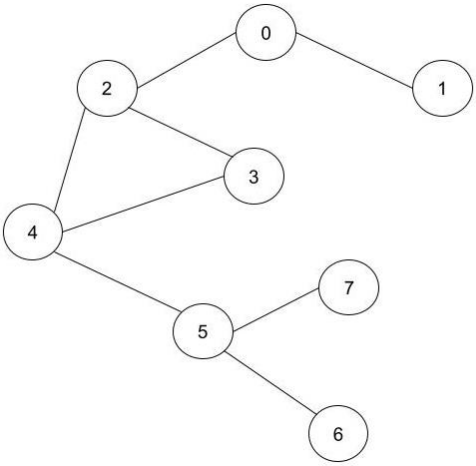
Ответ: 27

Задание 3. Написать программы для решения задачи.

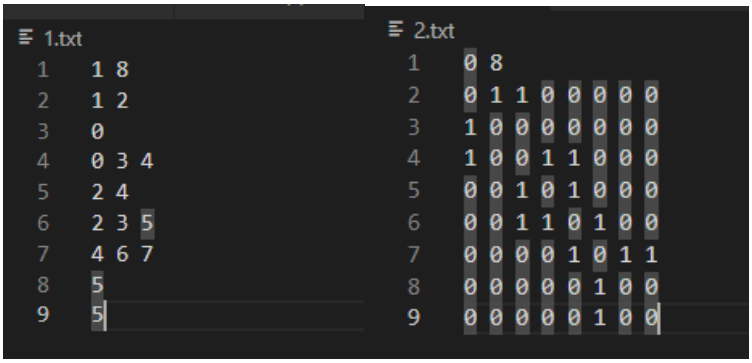
Повышенный. Граф представляется двумя способами (матрицей смежности или весов и списками смежности). Для каждого представления требуется написать отдельную программу решения задачи, используя алгоритм, наиболее подходящий для используемой структуры хранения. Для тестирования программ требуется создать файлы с описанием графа одним способом (только матрицей или только списками), обе программы должны 2 уметь заполнять структуры хранения, считывая файлы, как содержащие матрицы смежности, так и содержащие списки смежных вершин. При выборе структур хранения руководствоваться требованием разумной экономии памяти.

Известно, что заданный граф – не дерево. Напишите программу для 3 проверки, можно ли удалить из него одну вершину (вместе с инцидентными ей ребрами), чтобы в результате получилось дерево.

Тестовый граф 1:

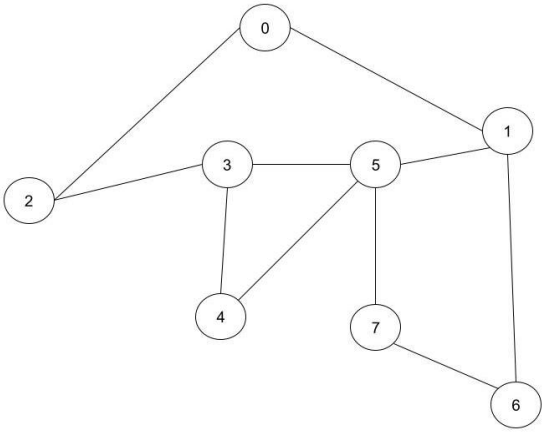


Первая строка – первая цифра (0 или 1) – тип представления графа (1 – список смежности, 0 – матрица смежности). Второе число – количество вершин в графе.



Ожидаемый результат: Можно, при удалении вершины 3.

Тестовый граф 2:



3.txt

```
1 8
1 2
0 5 6
0 3
2 4 5
3 5
1 3 4 7
1 7
5 6
```

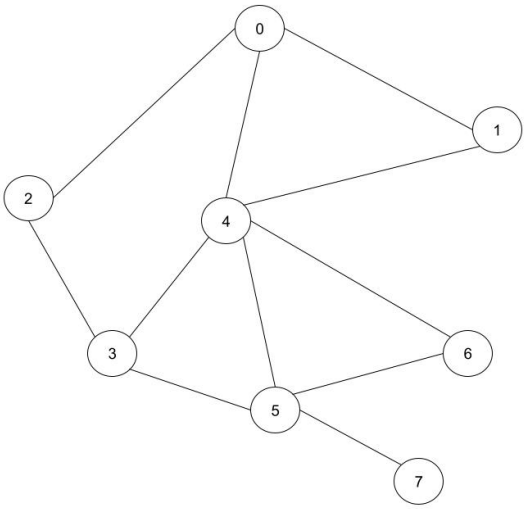
4.txt

```
0 8
0 1 1 0 0 0 0 0
1 0 0 0 0 1 1 0
1 0 0 1 0 0 0 0
0 0 1 0 1 1 0 0
0 0 0 1 0 1 0 0
0 1 0 1 1 0 0 1
0 1 0 0 0 0 0 1
0 0 0 0 0 1 1 0
```

Первая строка – первая цифра (0 или 1) – тип представления графа (1 – список смежности, 0 – матрица смежности). Второе число – количество вершин в графе.

Ожидаемый результат: Можно, при удалении вершины 5.

Тестовый граф 3:



5.txt

```
1 8
1 2 4
0 4
0 3
2 4 5
0 1 3 5 6
3 4 6 7
4 5
5
```

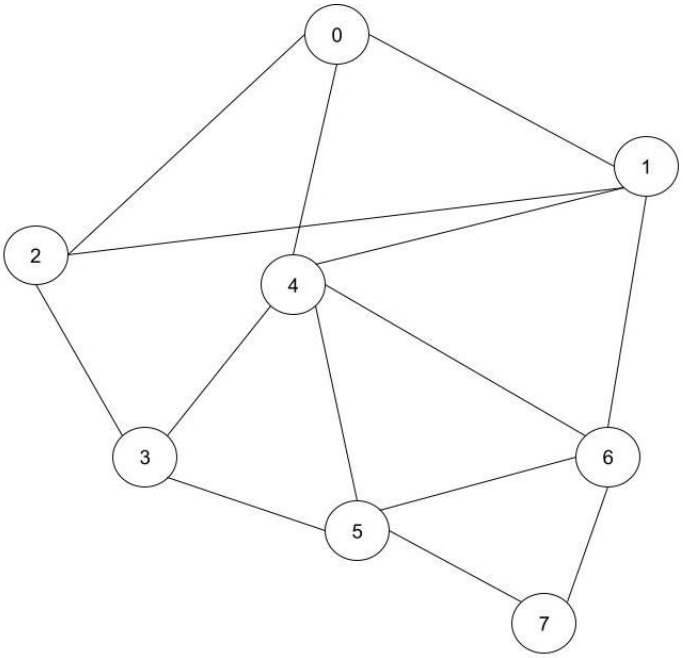
6.txt

```
0 8
0 1 1 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0
0 0 1 0 1 1 0 0
1 1 0 1 0 1 1 0
0 0 0 1 1 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 0 1 0 0
```

Первая строка – первая цифра (0 или 1) – тип представления графа (1 – список смежности, 0 – матрица смежности). Второе число – количество вершин в графе.

Ожидаемый результат: Можно, при удалении вершины 4.

Тестовый граф 4:



7.txt

```
0 8
0 1 1 0 1 0 0 0
1 0 1 0 1 0 1 0
1 1 0 1 0 0 0 0
0 0 1 0 1 1 0 0
1 1 0 1 0 1 1 0
0 0 0 1 1 0 1 1
0 1 0 0 1 1 0 1
0 0 0 0 0 1 1 0
```

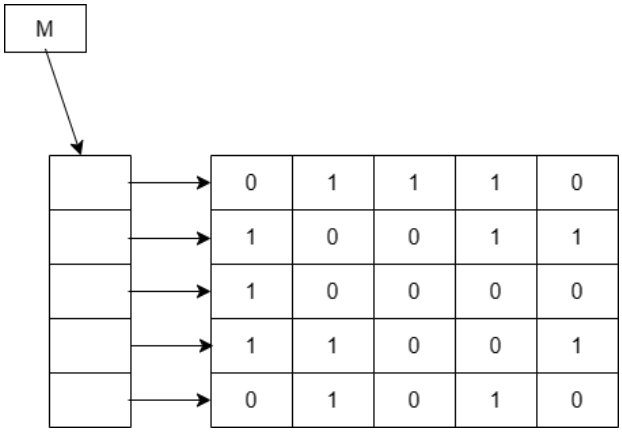
8.txt

```
1 8
1 2 4
0 2 4 6
0 1 3
2 4 5
0 1 3 5 6
3 4 6 7
1 4 5 7
5 6
```

Первая строка – первая цифра (0 или 1) – тип представления графа (1 – список смежности, 0 – матрица смежности). Второе число – количество вершин в графе.

Ожидаемый результат: Нельзя.

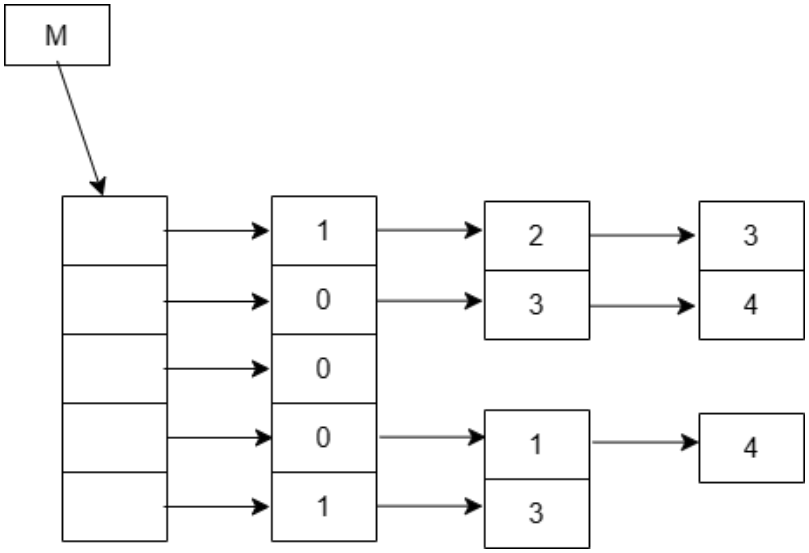
Структура хранения для представления графа матрицей смежности или весов:



. Структура хранения матрицы: динамическая двумерный массив, где 0 означает отсутствие ребра, а 1 наличие. Память выделятся под массив при чтении файла, для всех вершин. Выбрана данная структура хранения, так она будет занимать меньше памяти чем

связная структура хранения матрицы смежности. Представление графа в виде матрицы смежности удобно при проверке существования данной дуги, а также при добавлении и удаление дуг.

Структура хранения для представления графа списками смежных вершин:



Объём, выделяемой памяти, зависит от количества вершин в графе и ребер, он будет пропорционален их количеству, и экономия памяти будет происходить при динамическом выделение памяти под нужное количество вершин, и количество смежных вершин, при большем количестве вершин, данная структура будет занимать меньше памяти, чем другая структура хранения. Такая структура хранения позволяет быстро переходить к следующей вершине при обходе в глубину и поиске циклов при определении является ли граф деревом.

Применяемые алгоритмы:

Используется рекурсивный алгоритм обхода в глубину.

Текст программы:

Программа для матрицы смежности:

main.cpp

```
#include <iostream>
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
#include "func.h"

using namespace std;

int inputVerToRem(int);
bool Can(int **&, int);
int **arrayCopy(int **&, int);

int main()
{
    setlocale(LC_ALL, "Rus");
    ifstream in("5.txt"); // поток чтения
```

```
        int **x = NULL;       // двумерный массив для хранения графа
        int numberVerties, l; // количество вершин в графе
        bool flag = false;    // для понимания что в файле
        in.seekg(0, ios::beg);
        in >> l;
        in >> numberVerties;
        if (l == 0)
        {
            cout << "В файле матрица смежности" << endl;
            x = readMatrix(in, numberVerties);
        }
        if (l == 1)
        {
            cout << "В файле список смежных вершин" << endl;
            x = ListToMatrix(in, numberVerties);
        }
        if (x != NULL)
            showMatrix(x, numberVerties);
        bool *P = new (nothrow) bool[numberVerties];
        if (!P)
        {
            cout << "Ошибка выделения памяти" << endl;
            return 1;
        }
        memset(P, false, sizeof(P));
        if (isTree(x, numberVerties, P))
            cout << "Дерево" << endl;
        else
            cout << "Не дерево" << endl;
        cout << endl;
        cout << endl;
        if (Can(x, numberVerties))
        {
            cout << "Можно" << endl;
        }
        else
            cout << "Нельзя" << endl;
        delete[] x;
        return 0;
    }

    bool Can(int **&x, int numberVerties) // функци, которая проверяет можно
ли удалить из графа вершину и получиьь дерево
    {
        bool flag = false; // переменная которая показывает можно или нельзя
        int **newX = NULL; // массив из которого будем удалять
        int j;
        for (int i = 0; i < numberVerties; i++)
        {
            newX = arrayCopy(x, numberVerties);       // копируем граф в новый
двумерный массив из которого будем удалять
            newX = removeVert(i, newX, numberVerties); // удаляем i вершину из
графа
            int n = numberVerties - 1;                  // уменьшаем размер
матрицы скопиравнного графа
            bool *P = new (nothrow) bool[n];            // массив в котором
помечается какие вершины посещаются при обходе в глубину
            if (!P)
            {
                cout << "Ошибка выделения памяти" << endl;
                return false;
            }
            memset(P, false, n * sizeof(bool)); // заполняем массив  false,
```

```
ибо обход еще не начали
            if (isTree(newX, n, P))              // проверяем являет ли граф
после удаления вершины деревом
            {
                cout << "Номер вершины удалив которую мы можем получить дерево:
" << i << endl;
                return true; // если да завершаем проверку
            }
            delete[] newX; // удаляем созданные массивы
            delete[] P;
        }
        return flag;
    }

    int **arrayCopy(int **&x, int numberVerties) // копирование массива
    {
        int **G;
        int i, j;
        G = new (nothrow) int *[numberVerties]; // выделяем память
        if (!G)
        {
            cout << "Ошибка выделения памяти" << endl;
            return NULL;
        }
        for (i = 0; i < numberVerties; i++)
        {
            G[i] = new (nothrow) int[numberVerties]; // выделяем память
            if (!G[i])
            {
                cout << "Ошибка выделения памяти" << endl;
                return NULL;
            }
        }
        for (i = 0; i < numberVerties; i++)
            for (j = 0; j < numberVerties; j++)
                G[i][j] = x[i][j]; // заполняем значениями из исходного
массива
        return G;
    }
```

ReadFile.cpp

```
#ifndef ReadFile_CPP
#define ReadFile_CPP
#include <iostream>
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
#include <string>
#include "func.h"
using namespace std;

// прочитать матрицу смежности из файла
int **readMatrix(ifstream &in, int numberVerties)
{
    string s;
    int **G;
    int i, j;
    getline(in, s);
    s.clear();
    G = new (nothrow) int *[numberVerties];
    if (!G)
    {
```

```
        cout << "Ошибка выделения памяти" << endl;
        return NULL;
    }
    for (i = 0; i < numberVerties; i++)
    {
        G[i] = new (nothrow) int[numberVerties];
        if (!G[i])
        {
            cout << "Ошибка выделения памяти" << endl;
            return NULL;
        }
    }
    for (i = 0; i < numberVerties; i++)
        for (j = 0; j < numberVerties; j++)
        {
            in >> G[i][j];
        }
    return G;
}
// прочитать список смежности в матрицу смежности
int **ListToMatrix(ifstream &in, int numberVerties)
{
    string s;
    int **G;
    int i, j;
    G = new (nothrow) int *[numberVerties];
    getline(in, s);
    cout << s << endl;
    s.clear();
    if (!G)
    {
        cout << "Ошибка выделения памяти" << endl;
        return NULL;
    }
    for (i = 0; i < numberVerties; i++)
    {
        G[i] = new (nothrow) int[numberVerties];
        if (!G[i])
        {
            cout << "Ошибка выделения памяти" << endl;
            return NULL;
        }
        getline(in, s);
        int countS = search_count(s);
        int *arr = strToArr(s, countS);
        s.clear();
        if (arr[0] == -1) // если строка пустая
        {
            memset(G[i], 0, sizeof(int) * numberVerties);
        }
        else
        {
            memset(G[i], 0, sizeof(int) * numberVerties);
            for (int k = 0; k < countS; k++)
                G[i][arr[k]] = 1;
        }
        delete[] arr;
    }
    return G;
}

// показ матрицы
void showMatrix(int **&G, int numberVerties)
```

```
{
    int i, j;
    for (i = 0; i < numberVerties; i++)
    {
        for (j = 0; j < numberVerties; j++)
        {
            cout << G[i][j] << " ";
        }
        cout << endl;
    }
}

#endif
```

FunctionsForMatrix.cpp

```
#ifndef FunctionsForMatrix_CPP
#define FunctionsForMatrix_CPP
#include <iostream>
#include "func.h"

using namespace std;

int next(int start, int cur, int N, int **&Matrix) // получение индекса
следующей смежно вершины
{
    cur++;
    while (cur < N && !Matrix[start][cur])
        cur++;
    if (cur < N)
        return cur;
    return -1;
}

bool DFSR(int start, int **&Matrix, int N, bool *P, char parent)
{
    int nextVer;                                  // следующая вершины
    P[start] = true;                              // делаем посещенной вершну
    nextVer = next(start, parent, N, Matrix); // получаем следующую вершину
    while (nextVer != -1)                          // пока следующая вершина
не -1
    {
        if (!P[nextVer]) // если вершина не посещна то продожаем с нее
обход
        {
            if (!DFSR(nextVer, Matrix, N, P, start)) // если DFS false ,
значит у следующей вершины следующая вершина посещена и не равна родительской
                return false;
        }
        else if (nextVer != parent) // если следующая вершина посещена и
не равна родительской возвращаем false
            return false;
        nextVer = next(start, nextVer, N, Matrix);
    }
    return true;
}

bool isTree(int **&graph, int n, bool *&discovered)
{
    // чтобы отслеживать, открыта вершина или нет
    // логический флаг для хранения, является ли graph деревом или нет
    bool isTree = true;
    // Выполняем обход в глубину из первой вершины
```

```
        isTree = DFSR(0, graph, n, discovered, 0); // если у графа есть вершины которые имеют как-бы несколько родителей
        for (int i = 0; isTree && i < n; i++)
        {
            // любая неоткрытая вершина означает, что graph несвязен
            if (!discovered[i])
            {
                isTree = false;
            }
        }
        return isTree;
    }

    int **removeVert(int del, int **g, int numberVer) // удаление вершины перемещением удаляемой вершины в конец, и уменьшением размера массива
    {
        int i, j;
        for (i = del + 1; i < numberVer; i++)
            for (j = 0; j < numberVer; j++)
                g[i - 1][j] = g[i][j];
        for (j = del + 1; j < numberVer; j++)
            for (i = 0; i < numberVer; i++)
                g[i][j - 1] = g[i][j];
        return g;
    }

    #endif
```

func.h

```
    #ifndef func_H
    #define func_H
    #include <iostream>
    #include <string>
    #include <memory.h>
    #include <limits>
    #include <locale.h>
    #include <fstream>
    using namespace std;

    int search_count(string);   // подчсет количества элементов в строке для чтения
    int *strToArr(string, int); // превращаем строку в массив

    int **readMatrix(ifstream &, int);   // чтение матрицы из файла
    int **ListToMatrix(ifstream &, int); // чтение списка из файла в матрицу смежности
    void showMatrix(int **&, int);       // показ матрицы

    int next(int, int, int, int **&);           // переход к следующей вершине в dfs
    bool DFSR(int, int **&, int, bool *, char); // рекурсивный обход в глубину, который возвращает false, если следующая посещенная вершина не родительская

    bool isTree(int **&, int, bool *&); // является ли граф деревом
    int **removeVert(int, int **, int); // удаление вершины из графа

    #endif
```

func.cpp

```
    #ifndef func_CPP
    #define func_CPP
    #include <iostream>
    #include <string>
```

```
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
#include "func.h"
using namespace std;
// количество чисел в строке
int search_count(string s)
{
    bool flag = false;
    int i = 0, count = 0;
    while (s[i])
    {
        if (isdigit(s[i]))
        {
            if (!flag)
            {
                count++;
                flag = true;
            }
        }
        else
            flag = false;
        i++;
    }
    return count;
}
// строку в массив
int *strToArr(string s, int n)
{
    int *a = new (nothrow) int[n];
    int i = 0, j = 0;
    string ch;
    if (!a)
    {
        cout << "Ошибка выделения памяти" << endl;
        return NULL;
    }
    while (s[i])
    {
        ch.push_back(s[i]);
        if (s[i] == ' ' || s[i] == '\0')
        {
            a[j++] = stoi(ch);
            ch.clear();
        }
        i++;
    }
    a[j] = stoi(ch);
    return a;
}

#endif
```

Программа для списка смежности

main.cpp

```
#include <iostream>
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
#include "spisok.h"
#include "func.h"
```

```
using namespace std;

bool DFS(Graf const &, int, bool *&, int); // обход в глубину
bool Tree(Graf const &, int, bool *&);     // проверка на дерево
bool Can(Graf const &);                    // можно ли удалив вершину
получить дерево

int main()
{
    setlocale(LC_ALL, "Rus");
    ifstream in("5.txt");
    int numberVerties;
    bool flag = false;
    bool *P = NULL;
    Graf nG;
    Graf x(in);
    x.showGraf();
    numberVerties = x.numberVerties;
    P = new (nothrow) bool[numberVerties];
    if (!P)
    {
        cout << "Ошибка выделения памяти" << endl;
        return 1;
    }
    memset(P, false, sizeof(P));
    if (Tree(x, numberVerties, P))
        cout << "Это дерево" << endl;
    else
        cout << "Это не дерево" << endl;
    delete[] P;
    cout << endl;
    cout << endl;
    if (Can(x))
        cout << "Можно" << endl;
    else
        cout << "Нельзя" << endl;
    return 0;
}

bool Can(Graf const &x)
{
    bool *P;
    for (int i = 0; i < x.numberVerties; i++) // поочередно удаляем вершины
и проверяем дерево ли
    {
        Graf Ng(x);                           // конструктор копирования
        int k = i;                            // удаляемая вершины
        delVert(Ng, k);                       // удаляем вершину
        P = new (nothrow) bool[Ng.numberVerties]; // массив посещенных
вершин
        if (!P)
        {
            cout << "Ошибка выделения памяти" << endl;
            return 1;
        }
        memset(P, false, sizeof(P));      // заполнячем пустотой, так как
еще не посещали
        if (Tree(Ng, Ng.numberVerties, P)) // если дерево
        {
            cout << "Номер вершины, удалив которую можно получить дерево:
" << i << endl;
            return true;
```

```
            }
            delete[] P; // очищаем  память, для Ng работает деструктор
        }
        return false;
    }

    bool DFS(Graf const &graf, int start, bool *&P, int parent) // рекурстивный
обход в глубину
    {
        Node *temp = graf.graf[start].head; // для перемещения к следующей
вершины
        P[start] = true;                    // помечаем посещнной
        while (temp != NULL)
        {
            int nextVer = temp->data; // получаем следующею вершину
            if (nextVer == -1)          // если эта вершина с которой начали
обход не имеет смежных
                return false;
            if (!P[nextVer]) // если вершина не посещена продолжаем с нее обход
            {
                if (!DFS(graf, nextVer, P, start))
                {
                    return false;
                }
            }
            else if (nextVer != parent) // если следующая вершина посещена и
она равна не родителю, значит есть цикл и этот граф не может быть деревом
            {
                return false;
            }
            temp = temp->next; // переходим к следующему элемнту
        }
        return true;
    }

    bool Tree(Graf const &graph, int n, bool *&discovered)
    {
        // чтобы отслеживать, открыта вершина или нет
        // логический флаг для хранения, является ли graph деревом или нет
        // Выполняем обход в глубину из первой вершины
        bool isTree;
        isTree = true;
        isTree = DFS(graph, 0, discovered, -1);
        for (int i = 0; isTree && i < n; i++)
        {
            if (!discovered[i]) // если есть не посещнные или вершины без
смежжных ребер
            {
                isTree = false;
            }
        }
        return isTree;
    }
```

spisok.h

```
#ifndef Spisok_h
#define Spisok_h
#include <iostream>
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
using namespace std;
```

```
// запись односвязного списка
class Node
{
public:
    short int data;
    Node *next;

public:
    Node(short int x) // конструктор записи односвязного списка
    {
        this->data = x;
        this->next = NULL;
    }
};

class spis
{
public:
    Node *head, *tail; // начало и конец

public:
    spis();
    ~spis();
    void addBack(short int); // добавление элемента в конец односвзяного
списка
    bool isEmpty();          // проверка на пустоту
    void showSpis();         // показ списка
    bool findThis(int);      // поиск элемента
    void deleteHead();       // удаление элемента с начала
};

class Graf
{
public:
    int numberVerties; // количество вершин в графе
    spis *graf;        // массив односвязных списков

public:
    Graf();                          // конструктор без параметров
    Graf(int);                       // конструктор с параметрами
    Graf(const Graf &x);             // конструктор копирования
    ~Graf();                         // деструктор
    void showGraf();                 // показ графа
    Graf(ifstream &);                // чтение графа из файла
    Graf &operator=(const Graf &); // прегрузка оператора присваивания
};

void delVert(Graf &, int);    // удаление вершины из графа
Node *findTodel(spis &, int); // поиск позиции элемента в списке для
удаления
void delFrom(spis &, int);    // удаление из односвязного списка
bool Empty(int *, int);      // проверка массива на наличе ребер в вершины
void newInd(spis &, int);    // перерасчет индексов после удаления вершины
int countEl(spis &);             // подсчет количества смежных вершин в
односвязном списке
bool findK(spis &, int);      // провекрка наличия элемента в односвязном
списке

#endif
```

spisok.cpp

```
#ifndef Spisok_CPP
```

```
#define Spisok_CPP
#include "spisok.h"
#include "func.h"
#include <string>
using namespace std;

// функции списка
// конструткор списка без параметров
spis::spis()
{
    this->head = this->tail = NULL;
}
// проверкак списка на пустоту
bool spis::isEmpty()
{
    return (!head);
}
// деструктор списка
spis::~spis()
{
    // cout << "Destructor spis" << endl;
    while (head != NULL)
    {
        deleteHead();
    }
    head = NULL;
}
// добавление элемента в конец односвязного списка
void spis::addBack(short int x)
{
    Node *newNode = new (nothrow) Node(x);
    if (isEmpty())
        head = newNode;
    if (tail != NULL)
        tail->next = newNode;
    tail = newNode;
}
// показ списка
void spis::showSpis()
{
    Node *temp = head;
    while (temp != NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// проверка на наличие заданного элемента в списке
bool spis::findThis(int x)
{
    Node *temp = head;
    bool flag = false;
    while (temp != NULL)
    {
        if (temp->data == x)
            flag = true;
        temp = temp->next;
    }
    return flag;
}
// удаление элемента с головы односвзяного списка
void spis::deleteHead()
```

```
    {
        if (isEmpty()) // если список пуст
            return;
        if (head == tail) // если один элемент
        {
            delete tail;
            head = tail = NULL;
            return;
        }
        // если не один
        Node *temp = head;
        head = temp->next;
        delete temp;
    }

    // функции создания графа
    Graf::Graf(int V) // конструтор создания графа с параметром
    {
        this->numberVerties = V;
        if (!graf)
        {
            cout << "Ошибка выделения памяти" << endl;
            return;
        }
    }
    // конструктор графа без параметров
    Graf::Graf()
    {
        this->numberVerties = 0;
        graf = NULL;
    }
    // конструктор копирования
    Graf::Graf(const Graf &x)
    {
        this->numberVerties = x.numberVerties;                  // присваиваем
количество вершин
        this->graf = new (nothrow) spis[this->numberVerties]; // выделяем
память под массив списков
        if (!this->graf)
        {
            cout << "Ошибка выделения памяти" << endl;
            this->graf = NULL;
            return;
        }
        for (int i = 0; i < this->numberVerties; i++) // заполняем все
односвязные спсики
        {
            Node *temp = x.graf[i].head; // первый элемент односвязного спсика
            while (temp != NULL)         // пока есть элементы копируем
            {
                this->graf[i].addBack(temp->data);
                temp = temp->next;
            }
        }
    }
    // функции для чтения графа из файла и его заполнения и для удаления
    // деструктор графа
    Graf::~Graf()
    {
        for (int i = 0; i < numberVerties + 1; i++) // проходим по всем
элементам массива
        {
            graf[i].head = NULL;
```

```
        }
        delete[] this->graf; // удаляем граф
        graf = NULL;
    }

    Graf &Graf::operator=(const Graf &x)
    {
        this->numberVerties = x.numberVerties;                  // присваиваем
количество вершин
        this->graf = new (nothrow) spis[this->numberVerties]; // выделяем
память под массив списков
        if (!this->graf)
        {
            cout << "Ошибка выделения памяти" << endl;
            this->graf = NULL;
            return *this;
        }
        for (int i = 0; i < this->numberVerties; i++) // заполняем все
односвязные спсики
        {
            Node *temp = x.graf[i].head;
            while (temp != NULL)
            {
                this->graf[i].addBack(temp->data);
                temp = temp->next;
            }
        }
        return *this;
    }
    // показ графа
    void Graf::showGraf()
    {
        if (graf != NULL)
            for (int i = 0; i < numberVerties; i++)
            {
                cout << i << " :";
                graf[i].showSpis();
            }
        else
            cout << "Граф пуст" << endl;
    }
    // проверка строки матрицы на пустоту
    bool Empty(int *arr, int count)
    {
        for (int i = 0; i < count; i++)
            if (arr[i] != 0)
                return false;
        return true;
    }

    Graf::Graf(ifstream &in)
    {
        if (in.is_open()) // если файл есть
        {
            in.seekg(0, ios::beg);
            string s;
            int x;
            in >> x;                    // считываем тип представления графа в
файле
            in >> this->numberVerties; // считываем количество вершин графа
            getline(in, s);            // получаем строку и очищаем ее
            s.clear();
            if (x == 0)
```

```
            {
                cout << "В файле матрица смежности" << endl;
                this->graf = new (nothrow) spis[this->numberVerties]; //
выделяем память
                if (!this->graf)
                {
                    cout << "Ошибка выделения памяти" << endl;
                    return;
                }
                for (int i = 0; i < this->numberVerties; i++) // начинаем
построчно считывать файл и заполнять односвязные списки
                {
                    getline(in, s);                    // считали строку
                    int countS = this->numberVerties; // количество чисел в
строке матрицы смежности
                    int *arr = strToArr(s, countS);   // получаем массив чисел
из строки
                    if (arr == NULL)
                    {

                        cout << "Ошибка выделения памяти" << endl;
                        return;
                    }
                    if (Empty(arr, numberVerties)) // если все нули значит нет
смежных вершин
                    {
                        this->graf[i].addBack(-1);
                    }
                    else // если есть смежные вершины
                    {
                        for (int j = 0; j < countS; j++)
                        {
                            if (arr[j] == 1 && !this->graf[i].findThis(j)) //
если 1 в строке из матрицы и этого элемента еще нет в односвязном списке
                                this->graf[i].addBack(j);                  //
добовляем в конец
                        }
                    }
                    s.clear();    // очищаем строку
                    delete[] arr; // очищаем память
                }
            }
            if (x == 1) // если в файле список смежности
            {
                cout << "В файле список смежных вершин" << endl;
                this->graf = new (nothrow) spis[this->numberVerties]; //
выделяем память
                if (!this->graf)
                {
                    cout << "Ошибка выделения памяти" << endl;
                    return;
                }
                for (int i = 0; i < this->numberVerties; i++) // заполняем
списки
                {
                    getline(in, s);
                    int countS = search_count(s);    // получаем количество
смежных вершин
                    int *arr = strToArr(s, countS); // получем массив из строки
                    if (arr == NULL)
                    {
                        cout << "Ошибка выделения памяти" << endl;
                        return;
```

```
                    }
                    if (arr[0] == -1) // если -1 , то нет смежных вершин
                    {
                        this->graf[i].addBack(-1);
                    }
                    else // если есть смежные вершины
                    {
                        for (int j = 0; j < countS; j++)
                        {
                            if   (!this->graf[i].findThis(arr[j]))   //   если
элемента из массива еще нет в односвязном списке добавляем его
                                this->graf[i].addBack(arr[j]);
                        }
                    }
                    s.clear();    // очищаем
                    delete[] arr; // удаляем массив
                }
            }
        }
        else // если такого файла нет
        {
            cout << "Файл не найден" << endl;
        }
    }

    Node *findTodel(spis &x, int k) // поиск указателя на удаляемый элемент(то
есть предшествующий удаляемому)
    {
        Node *temp = x.head;
        bool flag = true;
        while (temp != NULL)
        {
            if (temp->data == k) // если первый элемент
                break;
            if (temp->next->data == k) // если следующий элемент односвязного
спсика равен ключу, то прекращаем поиск нашли нужное
            {
                break;
            }
            temp = temp->next;
        }
        return temp;
    }
    // удаление из односвязного списка с произвольной позиции
    void delFrom(spis &x, int k)
    {
        Node *node = findTodel(x, k); // находим предшествующий элемент для
удаляемого
        Node *temp;
        bool flag = true;
        if (x.head->data == k && x.head == x.tail) // если один элемент то
просто удаляем его
        {
            flag = false;
            delete x.tail;
            x.head = x.tail = NULL;
        }
        if (x.head->data == k && x.head == node) // если нужный элемент первый
        {
            flag = false;
            temp = node;
            x.head = node = node->next;
            delete temp;
```

```
            }
            if (temp = node->next) // если не в начале
            {
                if (flag == true) // если не последний
                {
                    node->next->data  =  node->next->data;  // изменяем значение
элемента
                    node->next = node->next->next;        // переносим указатель
на следующий после удаляемого
                    delete temp;
                }
            }
            else // если в конце
            {
                if (flag == true)
                {
                    delete node;
                    node = NULL;
                }
            }
        }

        bool findK(spis &x, int k) // проверяем на наличе элемента в односвязном
списке
        {
            Node *temp = x.head;
            while (temp != NULL)
            {
                if (temp->data == k)
                    return true;
                temp = temp->next;
            }
            return false;
        }
        // узнаем количество элементов в односвязном спсике
        int countEl(spis &x)
        {
            int count = 0;
            Node *temp = x.head;
            while (temp != NULL)
            {
                count++;
                temp = temp->next;
            }
            return count;
        }

        void newInd(spis &x, int k) // изменяем индексы после удаления
        {
            Node *temp = x.head;
            while (temp != NULL)
            {
                if (temp->data != -1 && temp->data >= k) // если есть смежные
вершины и смежная вершина больше или равна удаляемой то необходимо изменить ее
индексы
                    temp->data--;                          // уменьшаем индекс на 1
                temp = temp->next;
            }
        }

        void delVert(Graf &x, int k)
        {
            int j, count = 0;
```

```
        spis *temp;
        int n = x.numberVerties - 1;
        for (j = 0; j < n; j++) // удаление элемнта из массива
        {
            temp = &x.graf[j];           // получаем адрес односвязного списка
            if (findK(*temp, k) && j < k) // если есть элемент в строке и эту
строку не надо переносить при удалении
            {
                if (countEl(*temp) != 1) // если не один элемент
                {
                    delFrom(*temp, k);
                }
                else // если один елемент, то делаем вершину без смежных
                {
                    delFrom(*temp, k);
                    x.graf[j].addBack(-1);
                }
            }
            if (j >= k) // если надо удалять и перетаскивать элементы
            {
                x.graf[j] = x.graf[j + 1]; // переносим удаляемый элемент
                temp = &x.graf[j];           // получаем адрес односвязного
списка
                if (findK(*temp, k))        // если есть элемент который надо
удалить
                {
                    if (countEl(*temp) > 1) // если не один элемент
                    {
                        delFrom(*temp, k);
                    }
                    else // если рдин элемент то делаем его без смежных вершин
                    {
                        x.graf[j].deleteHead();
                        x.graf[j].addBack(-1);
                    }
                }
            }
        }
        x.numberVerties--;                    // уменьшаем размер графа
        for (j = 0; j < x.numberVerties; j++) // пересчитываем индексы для
всех вершин
        {
            temp = &x.graf[j];
            newInd(*temp, k);
        }
    }

    #endif
```

func.h

```
    #ifndef func_H
    #define func_H
    #include <iostream>
    #include <string>
    #include <memory.h>
    #include <limits>
    #include <locale.h>
    #include <fstream>
    using namespace std;

    int search_count(string);   // подсчет количества элементов в строке
    int *strToArr(string, int); // строку в массив
```

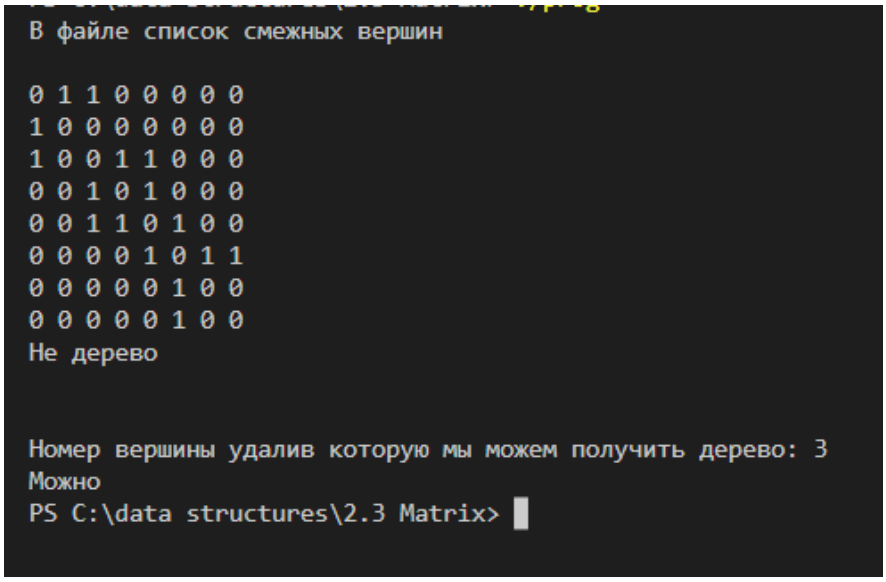
```
#endif
```

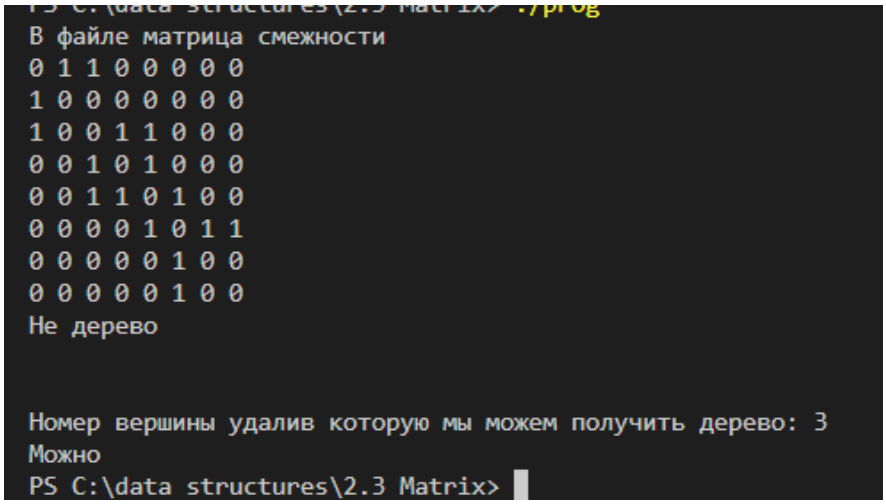
func.cpp

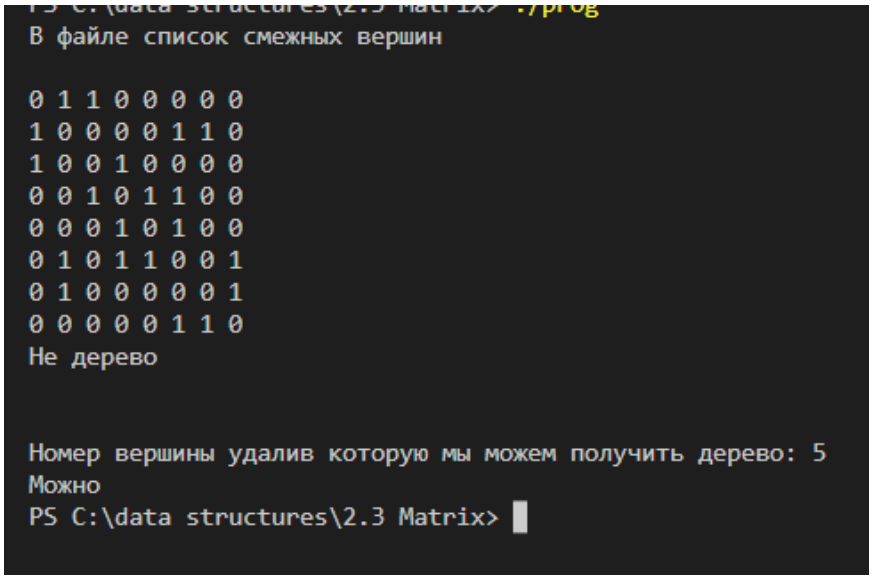
```
#ifndef func_CPP
#define func_CPP
#include <iostream>
#include <string>
#include <memory.h>
#include <limits>
#include <locale.h>
#include <fstream>
#include "func.h"
#include "spisok.h"
using namespace std;

int search_count(string s)
{
    bool flag = false;
    int i = 0, count = 0;
    while (s[i])
    {
        if (isdigit(s[i]))
        {
            if (!flag)
            {
                count++;
                flag = true;
            }
        }
        else
            flag = false;
        i++;
    }
    return count;
}
// строку в массив
int *strToArr(string s, int n)
{
    int *a = new (nothrow) int[n];
    int i = 0, j = 0;
    string ch;
    if (!a)
    {
        cout << "Ошибка выделения памяти" << endl;
        return NULL;
    }
    while (s[i])
    {
        ch.push_back(s[i]);
        if (s[i] == ' ' || s[i] == '\0')
        {
            a[j++] = stoi(ch);
            ch.clear();
        }
        i++;
    }
    a[j] = stoi(ch);
    return a;
}

#endif
```

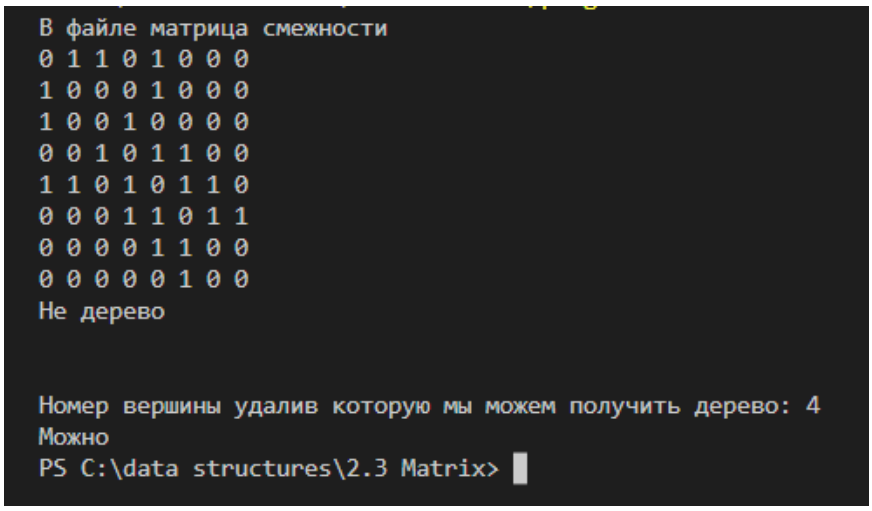
Результаты работы программы:

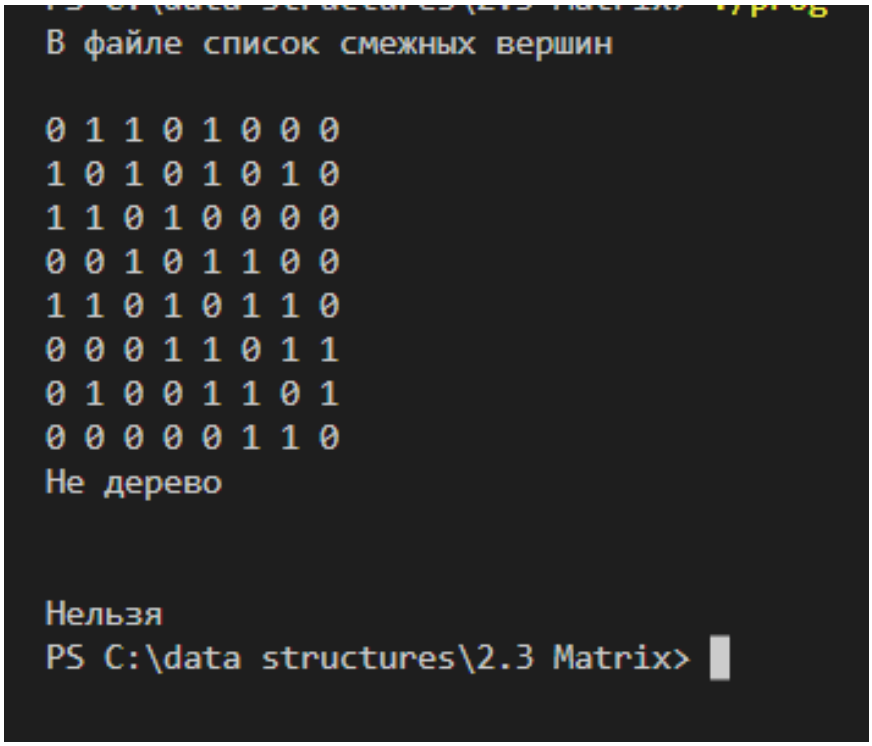
Тесты программы для матрицы смежности:








В файле матрица смежности
0 1 1 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0
0 0 1 0 1 1 0 0
1 1 0 1 0 1 1 0
0 0 0 1 1 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 0 1 0 0
Не дерево
Номер вершины удалив которую мы можем получить дерево: 4
Можно
PS C:\data structures\2.3 Matrix>


В файле список смежных вершин
0 1 1 0 1 0 0 0
1 0 1 0 1 0 1 0
1 1 0 1 0 0 0 0
0 0 1 0 1 1 0 0
1 1 0 1 0 1 1 0
0 0 0 1 1 0 1 1
0 1 0 0 1 1 0 1
0 0 0 0 0 1 1 0
Не дерево
Нельзя
PS C:\data structures\2.3 Matrix>

Тесты программы для списка смежности:

```
В файле список смежных вершин
0 :1 2
1 :0
2 :0 3 4
3 :2 4
4 :2 3 5
5 :4 6 7
6 :5
7 :5
Это не дерево


Номер вершины, удалив которую можно получить дерево: 3
Можно
PS C:\data structures\2.3 List>
```

```
PS C:\data structures\2.3 List> ./prog
В файле матрица смежности
0 :1 2
1 :0
2 :0 3 4
3 :2 4
4 :2 3 5
5 :4 6 7
6 :5
7 :5
Это не дерево


Номер вершины, удалив которую можно получить дерево: 3
Можно
PS C:\data structures\2.3 List>
```

```
PS C:\data structures\2.3 List> ./prog
В файле список смежных вершин
0 :1 2
1 :0 5 6
2 :0 3
3 :2 4 5
4 :3 5
5 :1 3 4 7
6 :1 7
7 :5 6
Это не дерево


Номер вершины, удалив которую можно получить дерево: 5
Можно
PS C:\data structures\2.3 List>
```

```
В файле список смежных вершин
0 :1 2 4
1 :0 4
2 :0 3
3 :2 4 5
4 :0 1 3 5 6
5 :3 4 6 7
6 :4 5
7 :5
Это не дерево


Номер вершины, удалив которую можно получить дерево: 4
Можно
PS C:\data structures\2.3 List>
```

```
В файле матрица смежности
0 :1 2 4
1 :0 2 4 6
2 :0 1 3
3 :2 4 5
4 :0 1 3 5 6
5 :3 4 6 7
6 :1 4 5 7
7 :5 6
Это не дерево


Нельзя
PS C:\data structures\2.3 List>
```