Министерство науки и образования РФ
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Пояснительная записка к курсовой работе по теме «Графы» по дисциплине «Алгоритмы и структуры данных» Вариант 26

Выполнил студент гр.9308: Яловега Н.В.

Проверил: Колинько П.Г.

Санкт-Петербург, 2020 г.

Оглавление

Введение

Исследование алгоритмов, реализуемых с помощью графов.

1. Задание

Построение ширинного стягивающего леса для неориентированного графа.

2. Формализация задания

Стягивающее дерево — ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины.

Т.к в задании не сказано, что граф должен быть связным, то мы будем находить стягивающий лес — подграф, состоящий из объединения стягивающих деревьев для каждой его компоненты связности.

Ширинный стягивающий лес — стягивающий лес, который построен при помощи обхода графа серией поисков в ширину.

Таким образом, для каждой компоненты связности, запускаем обход в ширину, строя стягивающее дерево для данной компоненты связности. Стягивающим лесом будет являться объединение стягивающих деревьев каждой компоненты связности.

3. Обоснование выбора способа представления графа в памяти ЭВМ

Для реализации данного алгоритма было принято представлять граф в памяти в виде списов смежности. Т.к для составления стягивающего леса, нам нужно обходить граф, то сложность обхода для такого представления будет линейной O(n+m). Если же использовать матрицу смежности, то сложность будет O(n^2). Кроме того, данный способ удобен для представления графов, в которых количество ребер не очень велико. Количество вершин ограничено сверху числом 26 (количество букв английского алфавита для наименования вершин). Максимальное количество ребер может быть у полного графа на 26 вершинах. Количество ребер для полного графа оценивается формулой n(n-1)/2 = 26*25/2 = 325. Тогда для хранения такого графа понадобится n+m памяти 325+26 = 351, что меньше, чем при использовании матрицы смежности, для которой необходимо n^2 памяти 26^2 = 676. Можно сэкономить некоторый объем памяти если воспользоваться тем фактом, что матрица смежности для неориентированного графа симметрична. Тогда нам достаточно будет хранить лишь верхний (или нижний) треугольник. В таком случае матрица смежности для плотного графа будет выйгрывать по памяти, но разница будет не очень существенная. Но при этом теряется простота использования этого метода хранения, и, что более критично, для разреженных графов время исполнения остается по своей сути квадратичным.

Для хранения стягиващего леса в памяти ЭВМ было принято использовать массив ребер. Используя данный способ представления нам достаточно хранить m пар чисел, где m — количество ребер, и число вершин леса n (для рассмотрения случаев, когда лес содержит вершины, которые не соединены ребрами с другими). Основная операция, которая используется в алгоритме — вставка нового ребра. Сложность вставки ребра, используя массив ребер, - O(1). Кроме того, это экономая структура для хранения дерева. Оценим количество ребер, которое необходимо хранить. Для каждой компоненты связности графа мы вычисляем стягивающее дерево. Количество ребер этого дерева будет на 1 меньше числа вершин в компоненте (по определению дерева). Тогда если в графе k компонент связности, то количество ребер в стягивающем лесе будет m = n-k. В худшем случае получим, что ребер будет на 1 меньше, чем вершин исходного графа (одна компонента связности). Значит в худшем случае, используя массив ребер, потребуется 2*m + 1 = 2*n -1 ячеек памяти. Если использовать матрицу смежности, которая тоже обладает константной сложностью вставки ребра в дерево, потребуется хранить n^2 элементов, что не является хорошим способом для хранения разреженного графа.

4. Временная сложность функций

1. **Ввод графа.**

Сложность: O(*n^2*), где n — количество вершин графа. Для ввода графа сначала заполняется матрица смежности, после чего она преобразуется в списки смежности. Это сделано для того, чтоб устранить возможную неполноту ввода пользователя.

2. **Генерация случайного графа.**

Сложность: O(*n^2*), где n — количество вершин графа. Для генерации графа матрица смежности заполняется случайными 1 и 0 симметрично относительно главной диагонали, после чего она преобразуется в списки смежности.

3. **Построение стягивающего дерева для компоненты связности.**

Сложность: O(*n*+*m*), где n и m — количество вершин и ребер компоненты связности. Обход подграфа в ширину.

4. **Построение стягивающего леса.**

Сложность: O(*n*+*m*), где n и m — количество вершин и ребер графа.Так как мы храним массив посещенных вершин, то мы не будем начинать обход из уже посещенных вершин, следовательно каждую вершину мы пройдем лишь один раз и сложность будет равна сложности обхода в ширину.

5. **Вывод леса.**

Сложность: O(*m*). Вывод m ребер в цикле.

5. Контрольные примеры

В таблице 1 представлены контрольные примеры, которые использовались для тестирования программы.

Таблица 1: Контрольные примеры

№	Входные данные		Результат
	Количество вершин	Множества смежности	Список ребер
1	11	a:b b:a c:dg d:fe e:g f:de g:ce h:ij i:hj j:hi k:0	a b c d c g d e d f h i h j
2	5	a: bcde b: ae c: ae d: ae e: abcd	a b a c a d a e
3	6	a: bcdf b: af c: adf d: acef e: d f: abcd	a b a c a d a f d e
4	8	a: bcde b: acde c: abde d: abce e: abcd f: gh g: fh h: fg	a b a c a d a e f g f h
5	3	a: 0 b: 0 c: 0	Лес не содержит ребер.

На рисунках 1 и 2 представлено графическое представление исходного графа и результата для примера 1. Данный пример используется в программе, как демонстрационный.

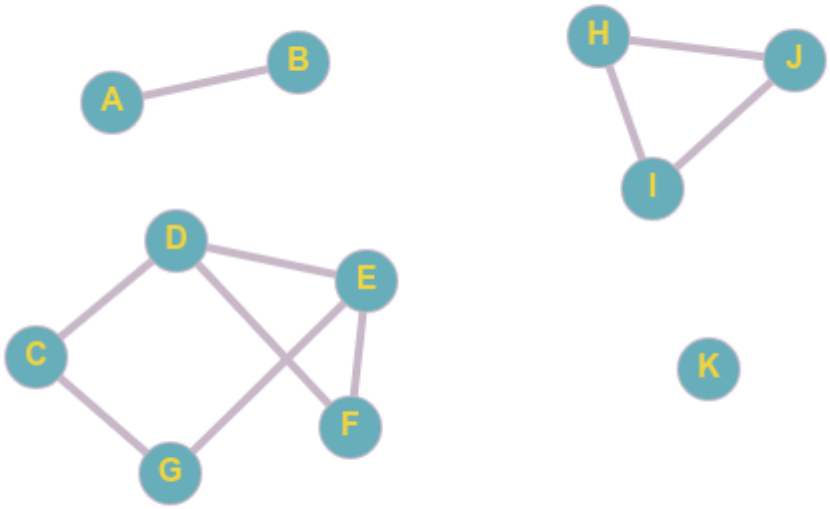

Рисунок 1: Исходный граф теста 1

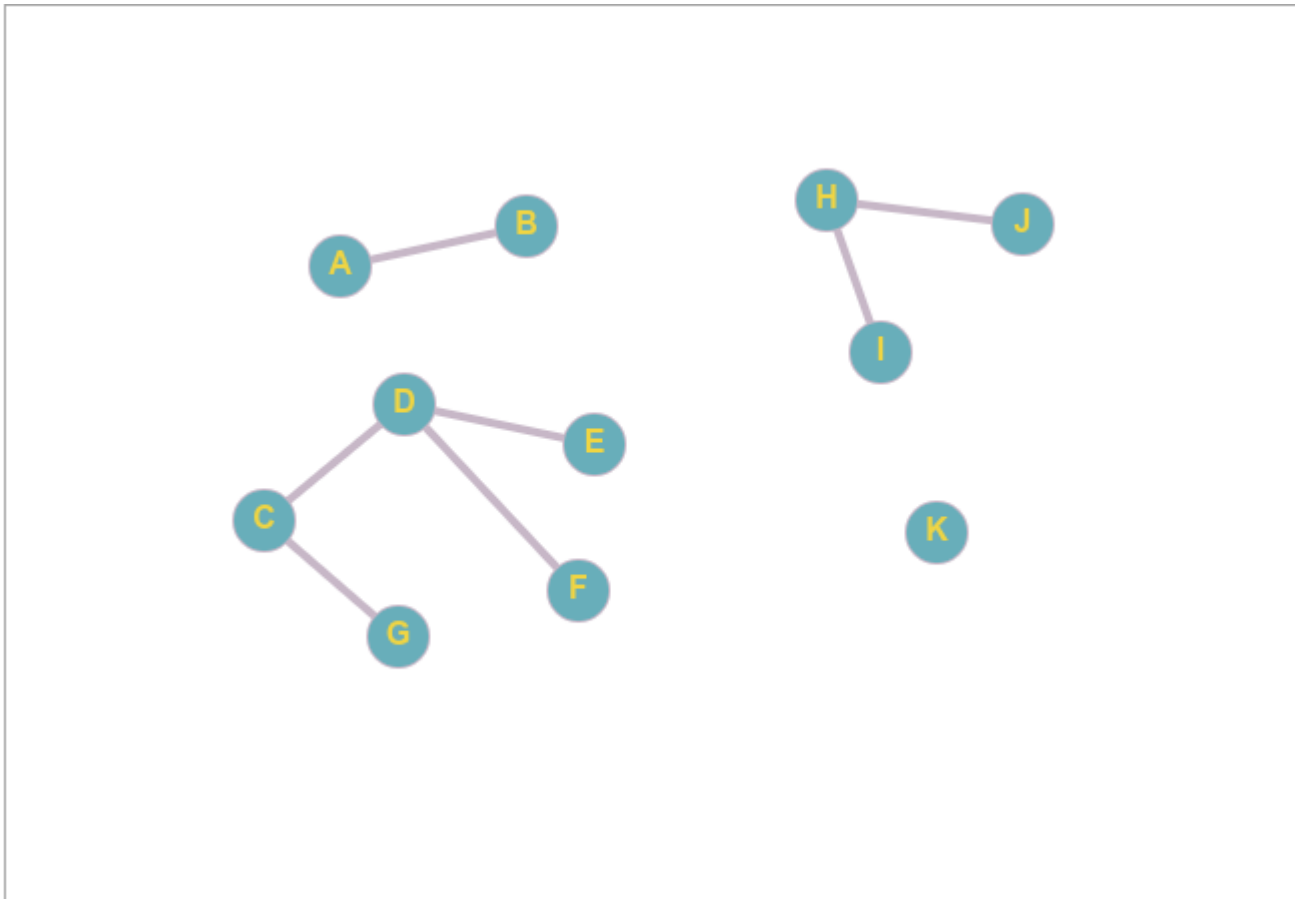

Рисунок 2: Ширинный стягивающий лес теста 1

6. Тестирование программы

На скриншотах ниже представлены результаты прогона программы для различных тестов из таблицы 1. Так же рассмотренны случаи некорректного ввода данных и неверного диапазона входных значений.

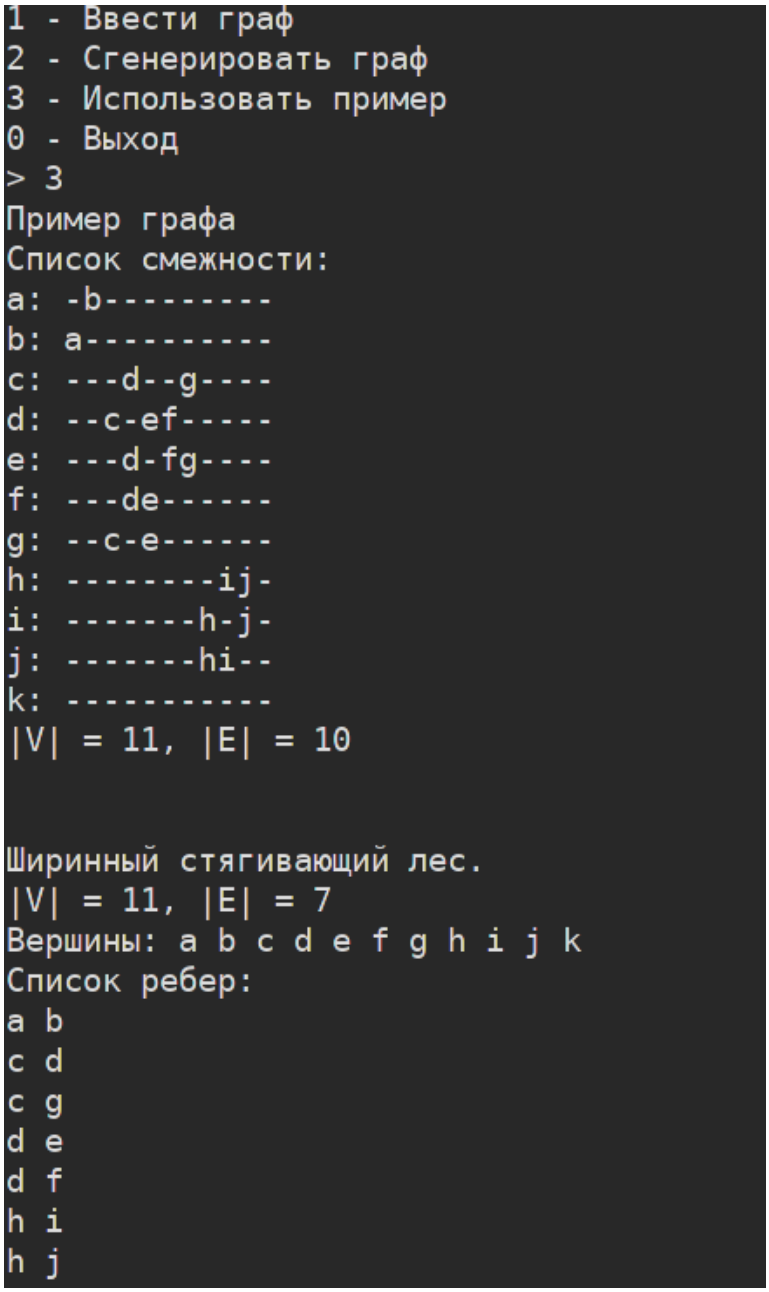

Рисунок 3: Тест 1

```
Введите количество вершин графа (число от 1 до 26): 8

Введите множества смежности (строки букв a до z)
v[a] = bcde
v[b] = acde
v[c] = abde
v[d] = abce
v[e] = abcd
v[f] = gh
v[g] = fh
v[h] = gf

Введенный граф
Список смежности:
a: -bcde---
b: a-cde---
c: ab-de---
d: abc-e---
e: abcd----
f: ------gh
g: -----f-h
h: -----fg-
|V| = 8, |E| = 13


Ширинный стягивающий лес.
|V| = 8, |E| = 6
Вершины: a b c d e f g h
Список ребер:
a b
a c
a d
a e
f g
f h
```

Рисунок 4: Тест 4

```
Введите количество вершин графа (число от 1 до 26): 3

Введите множества смежности (строки букв a до z)
v[a] = 0
v[b] = 0
v[c] = 0

Введенный граф
Список смежности:
a: ---
b: ---
c: ---
|V| = 3, |E| = 0


Ширинный стягивающий лес.
|V| = 3, |E| = 0
Вершины: a b c
Лес не содержит ребер.
```

Рисунок 5: Тест 5

```
Введите количество вершин графа (число от 1 до 26): -10

Число вершин не должно быть отрицательным или 0
Введите количество вершин графа (число от 1 до 26): ф

Некорректный ввод. Попробуйте еще раз
Введите количество вершин графа (число от 1 до 26): 10000

Число вершин не должно превышать 26
Введите количество вершин графа (число от 1 до 26): _
```

Рисунок 6: Некорректный ввод

```
1 - Ввести граф
2 - Сгенерировать граф
3 - Использовать пример
0 - Выход
> 100
Такого пункта не существует, повторите ввод!
```

Рисунок 7: Некорректный ввод

Вывод

В ходе выполнения работы был исследован алгоритм построения ширинного стягивающего леса для неориентированного графа. При тестировании программы с различными данными ошибки не были обнаружены.

Алгоритмы на графах просты для понимания (как минимум из-за возможности визуализации). На основе применения теории графов построено огромное количество самых разных технологий

Список используемых источников

1. Колинько П.Г. Пользовательские структуры данных / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.
2. Поздняков С.Н, Рыбин С.В Дискретная математика / Учебник для студ. Вузов — Издательский центр «Академия» 2008 — 448с
3. Скиена С. С Алгоритмы. Руководство по разработке. — 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2018. — 720 с.: ил

Приложение 1 (Исходный текст программы)

Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include <vector>
#include <list>
#include <queue>
#include <string>
#include <iostream>

using namespace std;

class Forest
{
   int n; // количество вершин
   int m; // количество ребер
   vector <pair<int, int>> edges; // список ребер
   public:
      Forest(int maxV): n(maxV) {};
      void print();
   friend class Graph;
};
class Graph
{
   int n; // количество вершин
   int m; // количество ребер
   vector <list<int>> LIST; // список смежности
   void spanning_tree(int, bool*, vector<pair<int, int>>*, int*); // построение сгивающего дерева
   public:
      Graph(); // пример графа
      Graph(int); // ввод графа
      Graph(int, char); // генерирование случайного графа
      Forest spanning_forest(); // постоение ширинного стягивающего леса
};

#endif
```

Graph.cpp

```
#include "graph.h"

char name_node(char c) {return c+'a';}



Graph :: Graph(): n(11), m(10)
/*
   Пример графа.
*/
{
   LIST.resize(n);
   LIST[0].push_back(1);
   LIST[1].push_back(0);
   LIST[2].push_back(3);LIST[2].push_back(6);
   LIST[3].push_back(2);LIST[3].push_back(4);LIST[3].push_back(5);
   LIST[4].push_back(3);LIST[4].push_back(5);LIST[4].push_back(6);
   LIST[5].push_back(3);LIST[5].push_back(4);
   LIST[6].push_back(2);LIST[6].push_back(4);
   LIST[7].push_back(8);LIST[7].push_back(9);
   LIST[8].push_back(7);LIST[8].push_back(9);
   LIST[9].push_back(7);LIST[9].push_back(8);


   cout << "Пример графа\nСписок смежности:\n";
   cout << "a: -b---------\n";
   cout << "b: a----------\n";
   cout << "c: ---d--g----\n";
   cout << "d: --c-ef-----\n";
   cout << "e: ---d-fg----\n";
   cout << "f: ---de------\n";
   cout << "g: --c-e------\n";
   cout << "h: --------ij-\n";
   cout << "i: -------h-j-\n";
   cout << "j: -------hi--\n";
   cout << "k: ----------\n";
   cout << "|V| = 11, |E| = 10\n";

}
```

```
Graph :: Graph(int maxV): n(maxV), m(0)
/*
   Ввод графа на n вершинах.
*/
{
   int G[n][n];
   string s;


   // обнуление матрицы
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         G[i][j] = 0;

   cout << "\nВведите множества смежности (строки букв a до z)\n";


   // заполнение матрицы смежности
   int k = 0;
   do
   {
      cout << "v[" << name_node(k) << "] = ";
      cin >> s;
      for (auto i : s)
         if (isalpha(i))
         {
            int j = tolower(i) - 'a';
            G[k][j] = G[j][k] = 1;
         }
      k++;
   }
   while(k < n);

   cout << "\nВведенный граф\nСписок смежности:";
   // заполнение списка смежности
   m = 0;
   LIST.resize(n);
   for (int i = 0; i < n; ++i)
```

```
    {
        int f = 0;
        cout << '\n' << name_node(i) << ": ";
        for (int j = 0; j < n; ++j)
            if (G[i][j])
            {
                ++f;
                LIST[i].push_back(j);
                cout << name_node(j);
            }
            else cout << '-';
        m += f;
    }

    cout << "\n|V| = " << n << ", |E| = " << m/2 << '\n';
}



Graph :: Graph(int maxV, char c): n(maxV), m(0)
/*
    Генерация случайного графа на n вершинах.
*/
{
    int G[n][n];

    // генерация случайной матрицы смежности
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j)
            if (i == j)
                G[i][j] = G[j][i] = 0;
            else
                G[i][j] = G[j][i] = rand()%2;

    cout << "\nСгенерированный граф\nСписок смежности:";
    // заполнение списка смежности
    m = 0;
    LIST.resize(n);
    for (int i = 0; i < n; ++i)
    {
```

```
        int f = 0;
        cout << '\n' << name_node(i) << ": ";
        for (int j = 0; j < n; ++j)
            if (G[i][j])
            {
                ++f;
                LIST[i].push_back(j);
                cout << name_node(j);
            }
            else cout << '-';
        m += f;
    }

    cout << "\n|V| = " << n << ", |E| = " << m/2 << '\n';
}


void Graph :: spanning_tree(int start_node, bool* visited, vector <pair<int, int>> *tree, int* m)
/*
    Поиск стягивающего дерева поиском в ширину.
*/
{
    queue<int> q;
    visited[start_node] = true;
    q.push(start_node);

    while (!q.empty())
    {
        int current_node = q.front();
        q.pop();

        for (auto adj_node : LIST[current_node])
            if (!visited[adj_node])
            {
                visited[adj_node] = true;
                q.push(adj_node);
                // cout << "Добавлено ребро: " << current_node << ' ' << adj_node << '\n';
                (*tree)[(*m)++] = (make_pair(current_node, adj_node));
            }
```

```
    }
}


Forest Graph :: spanning_forest()
/*
    Поиск стягивающего леса серией поисков в ширину.
*/
{

    Forest f(n); // т.к стягивающий лес, то количество вершин графа = количество вершин леса
    bool *visited = new bool[n]; // список посещенных вершин для обхода всех компонент

    f.m = 0;
    f.edges.resize(n-1);

    for (int node = 0; node < n; ++node)
        if (!visited[node]) // если вершина не была посещена, то
        {
            // запускаем обход из этой вершины
            spanning_tree(node, visited, &f.edges, &f.m);
        }

    f.edges.resize(f.m);
    delete [] visited;
    return f;
}


void Forest :: print()
/*
    Вывод леса.
*/
{
    cout << "\n\nШиринный стягивающий лес. \n";
    cout << "|V| = " << n << ", |E| = " << m << '\n';

    cout << "Вершины: ";
    for (int i = 0; i < n; ++i)
```

```
        cout << name_node(i) << " ";

    if (m > 0)
    {
        cout << "\nСписок ребер: \n";
        for (auto e:edges)
            cout << name_node(e.first) << " " << name_node(e.second) << '\n';
    }
    else
        cout << "\nЛес не содержит ребер.\n";
}
```

Main.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "graph.h"

using namespace std;

const int MAXV = 26; // максимальное количество вершин

short get_num_nodes();
void input_case();
void generate_case();
void example_case();

int main()
{
    srand(time(nullptr));
    int menu_item;

    do
    {
        cout << "1 - Ввести граф\n";
        cout << "2 - Сгенерировать граф\n";
        cout << "3 - Использовать пример\n";
        cout << "0 - Выход\n";
        cout << "> ";
```

```
        cin >> menu_item;
        switch(menu_item)
        {
            case 1:
                input_case();
                break;
            case 2:
                generate_case();
                break;
            case 3:
                example_case();
                break;
            case 0:
                cout << "";
                break;
            default:
                cout << "Такого пункта не существует, повторите ввод!\n";
        }
    }
    while(menu_item);

    return 0;
}

void input_case()
{
    int n = get_num_nodes();
    Graph g(n);
    Forest f = g.spanning_forest();
    f.print();
}


void generate_case()
{
    int n = get_num_nodes();
    Graph g(n, 0);
    Forest f = g.spanning_forest();
```

```
    f.print();
}

void example_case()
{
    Graph g;
    Forest f = g.spanning_forest();
    f.print();
}

short get_num_nodes()
/*
    Обработка ввода количества вершин
*/
{
    int n;

    do
    {
        cout << "Введите количество вершин графа (число от 1 до " << MAXV << "): ";
        cin >> n;
        if (cin.fail())
        {
            cin.clear();
            cin.ignore(32767,'\n');
            cout << "\nНекорректный ввод. Попробуйте еще раз\n";
        }
        else
        {
        if (n > MAXV)
            cout << "\nЧисло вершин не должно превышать " << MAXV << '\n';
        if (n < 1)
            cout << "\nЧисло вершин не должно быть отрицательным или 0\n";
        }

    } while(cin.fail() || n < 1 || n > MAXV);

    return n;
}
```