|  |  |
| --- | --- |
| Описание: для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«Российский государственный технический университет радиотехники, электроники и информатики»**  **МГТУ МИРЭА** | |
| Институт «Комплексной безопасности и специального приборостроения» | |
| Кафедра «Прикладные информационные технологии**»** (КБ-2) | |

**Утверждаю**

Заведующий кафедрой

|  |
| --- |
| **ЗАДАНИЕ**  **на выполнения курсовой работы** |
| **по дисциплинам** |
| **«Языки программирования и Дискретная математика»** |
| Студент: Маркеев Арсений Сергеевич Группа: БИСО-05-18  **1.Тема: Ориентированные и неориентированные графы.**  **2.Исходные данные:**  **3.Перечень вопросов, подлежащих разработке, и обязательного графического материала:**  **4.Срок представления к защите курсовой работы:**  Задание на курсовую работу выдал:  Задание на курсовую работу получил: |
|  |

|  |  |
| --- | --- |
| Описание: для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«Российский государственный технический университет радиотехники, электроники и информатики»**  **МГТУ МИРЭА** | |
| Институт «Комплексной безопасности и специального приборостроения» | |
| Кафедра «Прикладные информационные технологии**»** (КБ-2)  **Протокол заседания комиссии по защите курсового проекта**    от №  Состав комиссии: Выжигин А.Ю., Русаков А.М., Мерсов А.А.  Утверждена распоряжением заведующего кафедрой от  Слушали защиту курсовой работы (Ориентированные и неориентированные графы):  по дисциплинам: Языки программирования и Дискретная математика  студента группы: Маркеева Арсений Сергеевича  Во время защиты курсовой работы были заданы следующие вопросы  1.  2.  3. | |
|  | |

Итоговая(комплексная) оценкавыполнения и защиты курсовой работы

Члены комиссии:

**Содержание**

Цель курсовой работы……………………………………………………………4

Исходные данные варианта задания…………………………………………….4

Теоритическое введение………………………………………………………….5

Алгоритм решения задачи……………………………………………………..6-8

Листинг программы…………………………………………………………...8-21

Анализ результатов…………………………………………………………..22-25

Вывод……………………………………………………………………………..26

**Цель курсовой работы**

Цель курсовой работы по дисциплине «Языки программирования» состоит в закреплении и углублении знаний и навыков, полученных при изучении дисциплины. Курсовая работа предполагает выполнение задания повышенной сложности по проектированию, разработке и тестированию программного обеспечения, а также оформлению сопутствующей документации.

**Исходные данные варианта задания**



**Теоритическое введение**

Метод представления графа в данной задачи это матрица смежности.

Матрица смежности — таблица, где как столбцы, так и строки соответствуют вершинам графа. В каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот). Недостатком является требования к памяти — очевидно, квадрат количества вершин.

Для решения данной задачи подходят несколько алгоритмов:

1. Модифицированный алгоритм Дейкстры.
2. Модифицированный алгоритм Флойда-Уоршела.
3. Поиск в глубину.
4. Поиск в ширину.

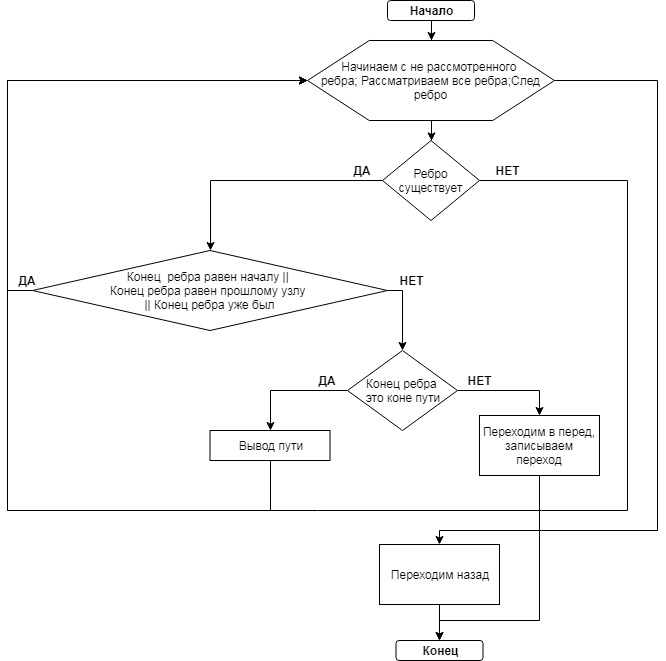
У алгоритма Флойда-Уоршела затрачиваемое время его работы равно N^3 (N – количество рассматриваемых вершин). Также этот алгоритм нужно дорабатывать для работы с разными типами графов. Алгоритм Дейкстры быстрее алгоритма Флойда-Уоршела, однако этот алгоритм тоже нужно модифицировать для работы с разными типами графов. Осталось 2 алгоритма обхода графов. Из них самый понятный и удобный для меня оказался обход графа в глубину (DFS).

Простой путь - это последовательность вершин, в которой каждая вершина соединена со следующей ребром и не одна вершина не повторяется. Иными словами путь, который не содержит циклы.

**Алгоритм решения задачи**

Нам дают 2 вершины. Одну мы берем за начало пути, и ищем существующее ребро идущее из этого узла. Если мы нашли это ребро, то проверяем его, если найденное ребро это петля, или ребро в прошлый узел или образующее цикл (ребро идущее в узел в котором мы уже были). Если все эти условия не выполняются, то проверяем ребро ведет в конечный узел – да, выводим путь и переходим назад – нет, переходим по этому ребру в следующий узел. Однако если ребер из узла больше нет то переходим в прошлый узел и ищем ребра идущие после пройденного. Выполняем данный алгоритм пока не рассмотрим все пути.

Блок-схема алгоритма (SearchWay):



Для решения задачи я использовал 2 класса.

Класс Graf:

Содержит: Динамическую матрицу смежности, динамический массив, кол-во узлов в графе.

Имеет конструктор который создает матрицу смежности заданной длинны.

Методы:

1. Enter() – выводит в консоль матрицу смежности
2. int getQuantNode() – возвращает кол-во узлов.
3. setEdge(int ind1, int ind2, bool val) – записывает в матрицу смежности ребро из *ind1* в *ind2*.
4. setNameNode(int ind, string Name) – записывает в массив имен имя *i* вершины.
5. int getIndNode(string Name) – возвращает индекс узла.
6. string getNameNode(int ind) - возвращает имя узла.
7. bool CheckEdge(int s, int f) – возвращает наличие ребра из *s* в *f*.
8. ChangeEdge() – запрашивает начало и конец изменяемого ребра, и предлагает на выбор: 1)Добавить ребро 2)Удалить ребро 3) Изменить направление.
9. AddNode() – добавляет узел.
10. DeleteNode() – удаляет узел.

Класс Way:

Содержит: список индексов.

Методы:

1. PWay getHead() – возвращает указатель на начало списка.
2. PWay CreateWay(int NewInd) – возвращает указатель на созданный элемент списка.
3. AddAfter(PWay p,PWay NewWay) – добавить в список после заданного.
4. AddFirst(PWay NewWay) – добавить в начало списка.
5. Operator +(PWay NewWay) - добавить в конец списка.
6. int BackWay() – стерает последний элемент списка и возвращает предпоследний индекс.
7. EnterSpis() – выводит список.
8. bool CheckNode(int N) – возвращает наличие данного узла в списке.
9. int getLast() – возвращает последний индекс.

**Листинг программы**

#include <iostream>

#include <cstring>

#include <stdlib.h>

#include <string.h>

#include <fstream>

#include <conio.h>

using namespace std;

char NameFile[30];

string start,end;

int NumEdge = 0; // Номер ребра с которого начинаем

int ActualNode;

int ReadFile(){

cout << "Введите имя файла (.tgf)";cin >> NameFile;

ifstream fin;

fin.open(NameFile);

if(!fin.is\_open()){

cout << "Ошибка" << endl;

return 0;

}

int Quant = 0;

string str = "";

while(str != "#"){

getline(fin,str);

Quant++;

}

Quant--;

fin.close();

return Quant;

}

class Graf{

protected:

bool \*\*Matr; //Матрица смежности

string \*MasLabel; //Массив имен

int QuantNode; //Кол-во узлов

public:

Graf(int q){

QuantNode = q;

MasLabel = new string[QuantNode];

Matr = new bool\*[QuantNode];

for(int i = 0;i < QuantNode;i++){

Matr[i] = new bool[QuantNode];

}

for(int i = 0;i < q;i++){

for(int j = 0;j < q;j++){

Matr[i][j] = 0;

}

}

}

void Enter(){

cout << " ";

for(int i = 0;i < QuantNode;i++){

cout << MasLabel[i] << " ";

}

cout << endl;

for(int i = 0;i < QuantNode;i++){

cout << MasLabel[i] << "|";

for(int j = 0;j < QuantNode;j++){

cout << Matr[i][j] << " ";

}

cout << endl;

}

}

int getQuantNode(){

return QuantNode;

}

int getIndNode(string Name){

int ind = -1;

for(int i = 0;i < QuantNode;i++){

if(Name == MasLabel[i]){

ind = i;

}

}

if(ind > -1){return ind;}

cout << "Узел с именем " << Name << " не найден\n";

exit(0);

}

string getNameNode(int ind){

return MasLabel[ind];

}

void setEdge(int ind1, int ind2, bool val){

Matr[ind1][ind2] = val;

}

void setNameNode(int ind, string Name){

MasLabel[ind] = Name;

}

bool CheckEdge(int s, int f){

if(Matr[s][f] == 1){

return 1;

}

else{

return 0;

}

}

void ChangeEdge(){

Enter();

string NameStart,NameEnd; // Имена узлов

int s = -1,e = -1; // Индексы узлов

cout << "\nКакое ребро изменьть:.\n";

cout << "Начало:"; cin >> NameStart;

cout << "Конец:"; cin >> NameEnd;

s = getIndNode(NameStart);

e = getIndNode(NameEnd);

cout << "1)Добавить ребро\n2)Удалить ребро\n3)Изменить напрвление ребра.\n(Для выхода нажмите любую другую клавишу)\n";

switch(getche()){

case '1':// Добавление ребра

Matr[s][e] = 1;

system("cls");

cout << "Ребро (" << NameStart << "->" << NameEnd<< ") добавленно.\n\n";

break;

case '2':// Удаление ребра

Matr[s][e] = 0;

system("cls");

cout << "Ребро (" << NameStart << "->" << NameEnd << ") удаленно.\n\n";

break;

case '3':// Изменение направления

system("cls");

if(Matr[s][e] == 0){cout << "Дуги нет.\n\n";}

else{

Matr[s][e] = 0;

Matr[e][s] = 1;

cout << "Направление измененно.\n\n";

}

break;

}

}

void AddNode(){

string NameNewNode;// Имя узла

cout << "\nВведите имя узла:"; cin >> NameNewNode;

bool BufMatr[QuantNode][QuantNode];// Копия матрицы

string BufMasLabel[QuantNode];// Копия массива

for(int i = 0; i < QuantNode;i++){ // Копируем значения массива имен и матрицы

BufMasLabel[i] = MasLabel[i];

for(int j = 0; j < QuantNode;j++){

BufMatr[i][j] = Matr[i][j];

}

}

for(int i = 0; i < QuantNode;i++){ // Удаляем матрицу

delete [] Matr[i];

}

delete [] MasLabel;// Удаляем массив

QuantNode++;// Увеличиваем на 1 кол-во узлов

MasLabel = new string[QuantNode];// Создаем новый массив

Matr = new bool\*[QuantNode];// Создаем новую матрицу

for(int i = 0;i < QuantNode;i++){

Matr[i] = new bool[QuantNode];

}

// Записываем значения из старой матрицы в новую

for(int i = 0; i < QuantNode;i++){

if(i < QuantNode-1){

MasLabel[i] = BufMasLabel[i];

}

else{

MasLabel[i] = NameNewNode;

}

for(int j = 0; j < QuantNode;j++){

if(i < QuantNode-1 && j < QuantNode-1){

Matr[i][j] = BufMatr[i][j];

}

else{

Matr[i][j] = 0;

}

}

}

}

void DeleteNode(){

string NameNode;// Имя узла

int IndNode;

bool flag = 0;

cout << "\nВведите имя узла:"; cin >> NameNode;

for(int i = 0;i < QuantNode;i++){// Поиск индексов узлов

if(MasLabel[i] == NameNode){flag = 1;IndNode = i;}

}

if(!flag){

cout << "Узел не найден.\n\n";

return;

}

bool BufMatr[QuantNode][QuantNode];// Копия матрицы

string BufMasLabel[QuantNode];// Копия массива

for(int i = 0; i < QuantNode;i++){ // Копируем значения массива имен и матрицы

BufMasLabel[i] = MasLabel[i];

for(int j = 0; j < QuantNode;j++){

BufMatr[i][j] = Matr[i][j];

}

}

for(int i = 0; i < QuantNode;i++){ // Удаляем матрицу

delete [] Matr[i];

}

delete [] MasLabel;// Удаляем массив

QuantNode--;// Уменьшаем на 1 кол-во узлов

MasLabel = new string[QuantNode];// Создаем новый массив

Matr = new bool\*[QuantNode];// Создаем новую матрицу

for(int i = 0;i < QuantNode;i++){

Matr[i] = new bool[QuantNode];

}

// Записываем значения из старой матрицы в новую

int k = 0;

for(int i = 0; i <= QuantNode;i++){

if(i != IndNode){

MasLabel[k] = BufMasLabel[i];

int d = 0;

for(int j = 0; j <= QuantNode;j++){

if(j != IndNode){

Matr[k][d] = BufMatr[i][j];

d++;

}

}

k++;

}

}

}

};

Graf ObjGraf(ReadFile());

class Way{

private:

struct NodeWay{

int nod;

NodeWay \*next;

};

typedef NodeWay \*PWay;

PWay Head = NULL;

public:

PWay getHead(){

return Head;

}

PWay CreateWay(int NewInd){

PWay NewWay = new NodeWay;

NewWay->nod = NewInd;

NewWay->next = NULL;

return NewWay;

}

void AddAfter(PWay p,PWay NewWay){

NewWay->next = p->next;

p->next = NewWay;

}

void AddFirst(PWay NewWay){

NewWay->next = Head;

Head = NewWay;

}

int BackWay(){

PWay q = Head;

PWay p = Head->next;

if(p != NULL){

while(p->next) {

q = q->next;

p = p->next;

}

q->next = NULL;

return q->nod;

}

else {

Head = NULL;

return 0;

}

}

void EnterSpis(){

PWay q = Head;

while(q != NULL){

cout << ObjGraf.getNameNode(q->nod) <<"->";

q= q->next;

}

cout << end <<endl;

}

bool CheckNode(int N){

PWay q = Head;

while(q != NULL){

if(q->nod == N) {

return true;

}

q = q->next;

}

return false;

}

int getLast(){

PWay q = Head;

PWay p = q->next;

if(p != NULL){

while(p->next) {

q = q->next;

p = p->next;

}

}

return q->nod;

}

void operator +(PWay NewWay){

if(Head == NULL){

AddFirst(NewWay);

return;

}

PWay q = Head;

while(q->next) q = q->next;

AddAfter(q,NewWay);

}

};

Way ObjWay;

void Fill(char NameF[] , int QuantNode){

ifstream fin;

fin.open(NameF);

string str;

for(int i = 0;i < QuantNode;i++){

fin >> str;

fin >> str;

ObjGraf.setNameNode(i,str);

}

fin >> str;

int buf1,buf2;

while(fin >> buf1 && fin >> buf2){

ObjGraf.setEdge(buf1 - 1, buf2 - 1, 1);

}

fin.close();

}

void SearchWay(int NEdge, int lM){

for(int i = NEdge;i < lM;i++){

if(ObjGraf.CheckEdge(ActualNode,i)){

if(ActualNode == i || i == ObjWay.getLast() || ObjWay.CheckNode(i)){/\*"Пропуск"\*/}

else if(i == ObjGraf.getIndNode(end)){

ObjWay.EnterSpis();

}

else{

ActualNode = i; //Переходим в перед

ObjWay + ObjWay.CreateWay(i); // Записываем переход

NumEdge = 0; //Начинаем с 0 ребра

return;

}

}

}

NumEdge = ActualNode + 1;// Начинаем с ребра после предедущего

ActualNode = ObjWay.BackWay(); // Записываем прошлый узел

}

void First(){

string f = ObjGraf.getNameNode(1);

cout << "Идекс первой вершины:" << ObjGraf.getIndNode(f) << endl;

}

void Next(string v, int i){

int ind = ObjGraf.getIndNode(v);

for(int j = i; j < ObjGraf.getQuantNode();j++){

if(ObjGraf.CheckEdge(ind,j)){

cout << "Индекс смежной вершины:" << j << endl;

return;

}

}

}

void Vertex(string v,int i){

int ind = ObjGraf.getIndNode(v);

if(ObjGraf.CheckEdge(ind,i)){

cout << "Вершина:" << ObjGraf.getNameNode(i) << endl;

}

}

int main() {

// setlocale(LC\_ALL,"Rus");

string V;

int I;

Fill(NameFile,ObjGraf.getQuantNode());

while(true){

ObjGraf.Enter();

cout << "1)Найти все простые пути.\n2)Изменение ребер.\n3)Добавить узел.\n4)Удалить узел.\n5)FIRST\n6)NEXT\n7)VERTEX\n0)Выход.\n";

switch(getche()){

case '1':

cout << "Начало пути:"; cin >> start;

cout << "Конец пути:"; cin >> end;

system("cls");

ActualNode = ObjGraf.getIndNode(start); //Расматриваемый узел

ObjWay + ObjWay.CreateWay(ActualNode); //Записываем узел в начало

NumEdge = 0;

while(ObjWay.getHead() != NULL){

SearchWay(NumEdge,ObjGraf.getQuantNode());

}

break;

case '2':

system("cls");

ObjGraf.ChangeEdge();

break;

case '3':

system("cls");

ObjGraf.AddNode();

break;

case '4':

system("cls");

ObjGraf.DeleteNode();

break;

case '5':

system("cls");

First();

break;

case '6':

system("cls");

cout << "Введите вершину:";cin >> V;

cout << "Введите индекс:";cin >> I;

Next(V,I);

break;

case '7':

system("cls");

cout << "Введите вершину:";cin >> V;

cout << "Введите индекс:";cin >> I;

Vertex(V,I);

break;

case '0':

exit(0);

break;

}

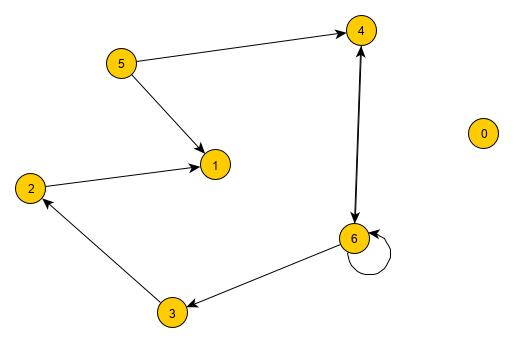
}

system("pause");

}

**Анализ результатов**

Граф 1 (7 узлов, 8 ребер)



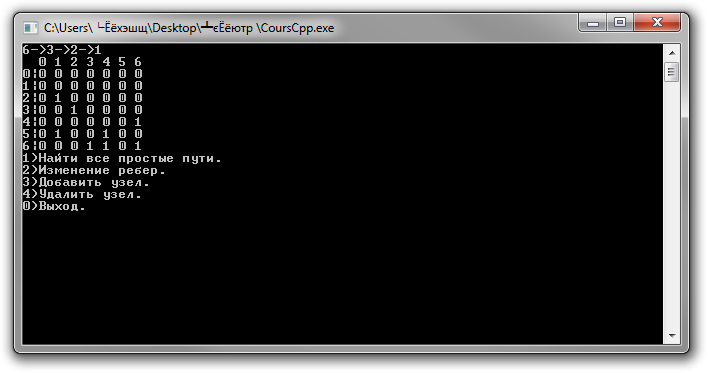
Пример:

Начало – 6. Конец – 1.

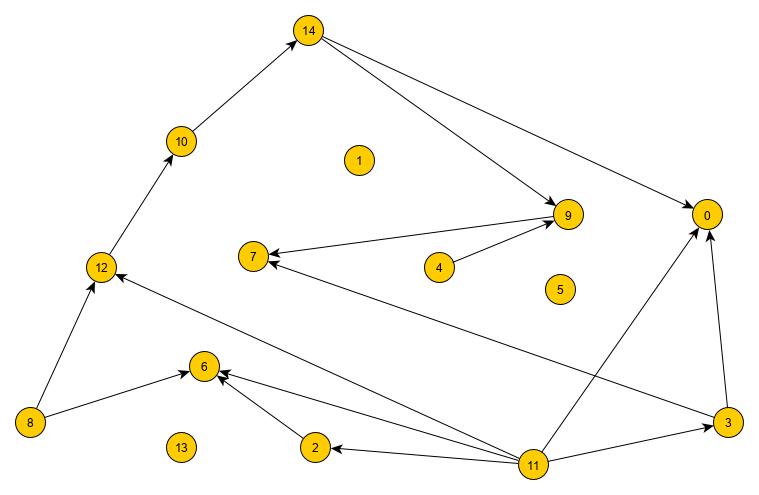
1. Начинаем с 6 узла. Ищем смежные узлы. Это узел 3. Это не петля, не прошлый узел и его мы не проходили. Он не конечный. Переходим вперед.
2. Узел 3. Ищем смежные узлы. Это Узел 2. Это не петля, не прошлый узел и его мы не проходили. Он не конечный. Переходим вперед.
3. Узел 2. Ищем смежные узлы. Это Узел 1. Это не петля, не прошлый узел и его мы не проходили. Этот узел конец. Записываем путь. (Вывод в консоль).
4. Продолжаем искать смежные узлы. У узла 2 нет больше смежных узлов. Переход назад.
5. Продолжаем искать смежные узлы. У узла 3 нет больше смежных узлов. Переход назад.
6. Продолжаем искать смежные узлы. Узел 4. Это не петля, не прошлый узел и его мы не проходили. Он не конечный. Переходим вперед.
7. Узел 4. У этого узла нет смежных вершин. Переход назад.
8. Продолжаем искать смежные узлы. Узел 6. Это петля. Пропускаем узел.
9. У узла 3 нет больше смежных узлов. Так как это начальный узел, то все доступные узлы рассмотрены.

Ответ: 6->3->2->1

Распечатка программы:

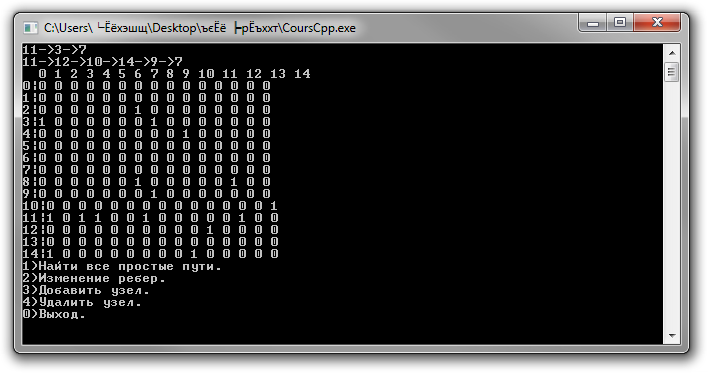


Граф 2 (15 узлов, 16 ребер)

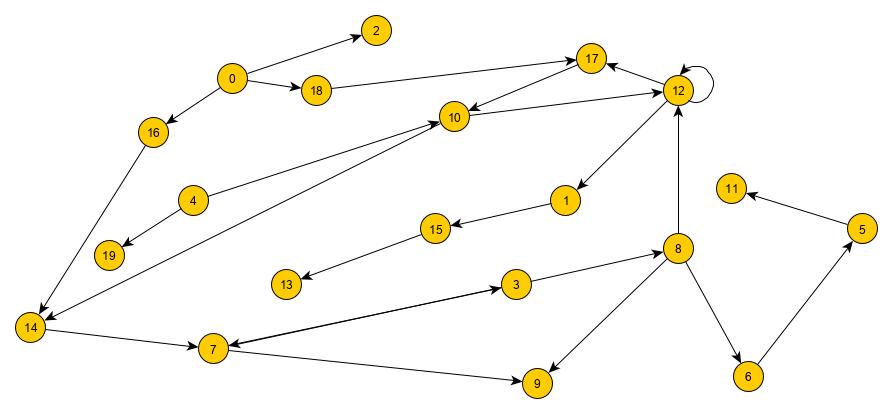


Начало – 11. Конец – 7.

Распечатка программы:



Граф 2 (20 узлов, 24 ребера)



Начало – 0. Конец – 14.

Распечатка программы:



**Вывод**

Исследовав теорию графов, получилось написать программу которая эффективно выполняет поставленную задачу, поиск простых вершин в графе. Программа обрабатывает любые типы графов.

Плюсы данной программы:

* Быстрый и эффективный алгоритм
* Полная настройка графа
* Принимает любой граф