Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №9

по теме «Поиск расстояний в графе»

Выполнили:

Студенты группы 21ВВ2

Назаров Е.А.

Макаров И.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза, 2022

**Цель работы:**

Решение задач, использующих графовые модели представления, изучение различных методов решения и анализа.Поиск кратчайшего пути в графе.

**Лабораторные работы:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

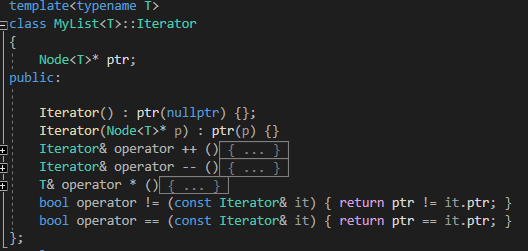
**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

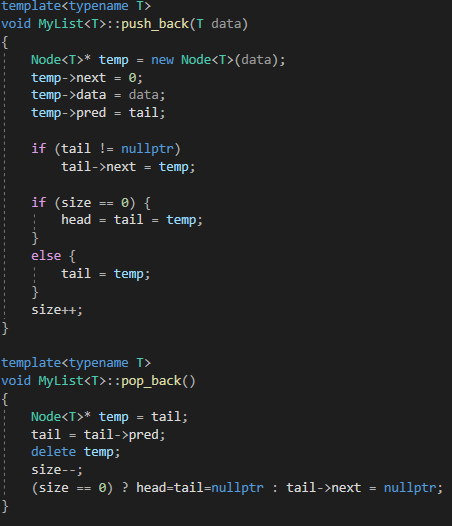
**Пояснительный текст к программе:**



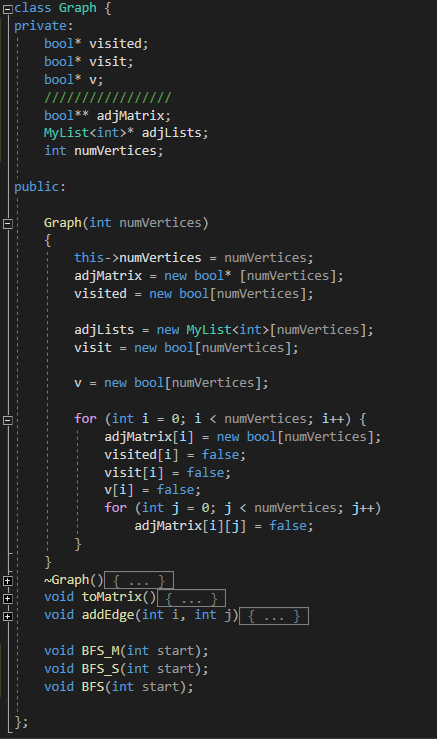
Был реализован динамический шаблонный двухсвязный список. Класс MyList представляет собой целостный класс с описанными функциями в нем. В классе был реализован класс Node элемента, содержащий в себе всю информацию и класс Iterator для перемещения по списку.



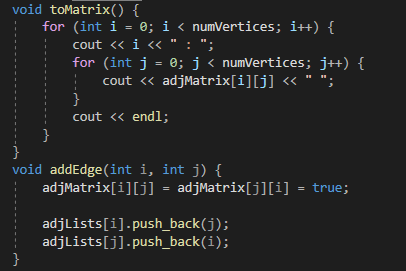
Содержит в себе в большей степени операторов перегрузки для хождения от одного элемента класса к другому. Функции, возвращающие начало и конец были вынесены в класс MyList.



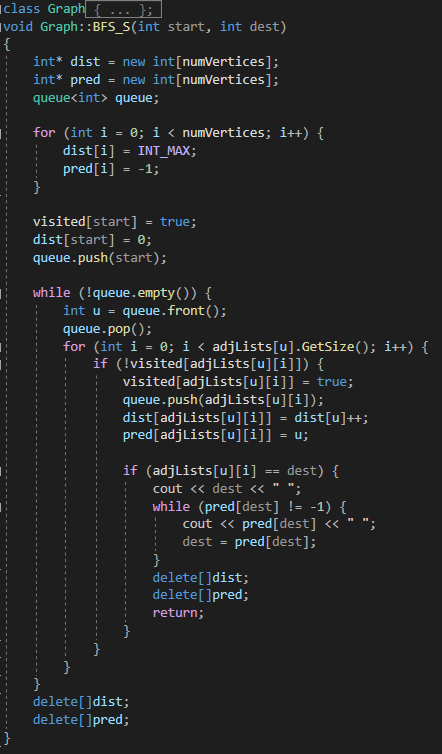
Функции добавления в конец списка(push\_back) и удаления последнего элемента списка(pop\_back).



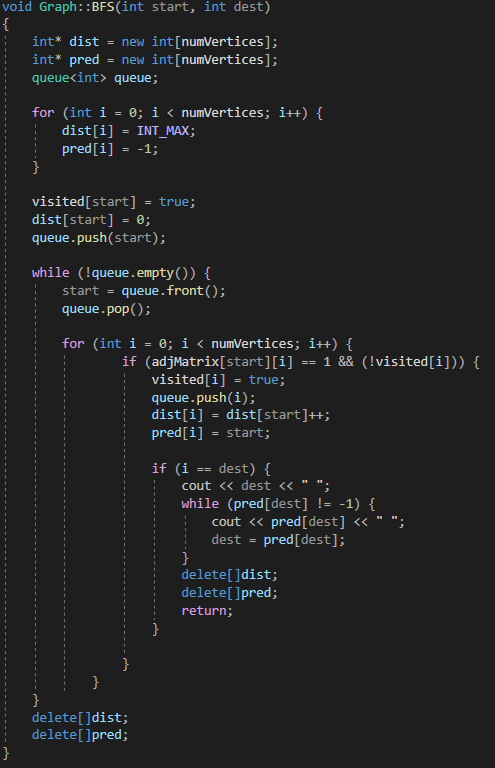
Класс содержащий в себе информацию о графе. В конструктор поступает количество вершин, после чего выделяется динамическая память для всех нужных элементов и массивов. Деструктор же освобождает эту память в конце программы.



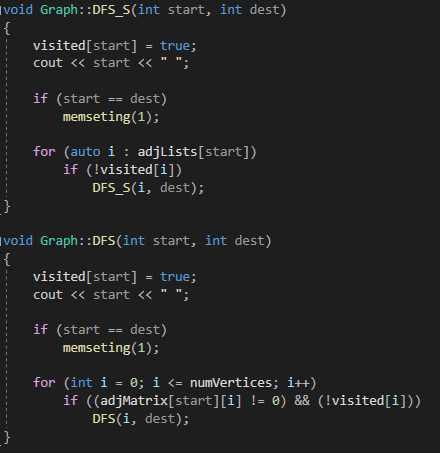
Функции вывода матрицы смежности на экран(toMatrix) и добавления связей между вершинами(addEdge).



Функция представляет основную идею функцию BFS(проход графа в ширину) из 8 лабораторной работы работающую на списке смежности. Смысл алгоритма состоит в том, что мы запоминаем расстояние, сохраняя все вершины в массиве. Когда мы доходим до конечной вершины(которую пользователь вводит сам), мы выводим массив и на результат получаем кратчайший маршрут.



Аналогично выше приведенной функции, только за основу алгоритма стала матрица смежности.



Функции представляют основную идею функцию DFS(проход графа в глубину) из 7 лабораторной работы. В отличие от выше приведенных функций, эти находят не кратчайший путь от заданных точек, а первый попавшийся для программы.

**Тестирование программы:**

|  |  |
| --- | --- |
| Количество вершин | Реализация |
| |  |  |  |  | | --- | --- | --- | --- | | Матрица смежности  BFS | Список смежности  BFS | Матрица смежности  DFS | Список смежности  DFS | |
| 100 | |  |  |  |  | | --- | --- | --- | --- | | 0.001 | 0 | 0.002 | 0.001 | |
| 200 | |  |  |  |  | | --- | --- | --- | --- | | 0.001 | 0.001 | 0.003 | 0.002 | |
| 400 | |  |  |  |  | | --- | --- | --- | --- | | 0.01 | 0.006 | 0.016 | 0.01 | |
| 1 000 | |  |  |  |  | | --- | --- | --- | --- | | 0.015 | 0.012 | 0.045 | 0.02 | |

Вывод: По приведенным выше замеров можно огласить результат: самый быстрой по реализация функцией оказалась BFS(Список смежности), самой медленной стала функция DFS(Матрица смежности)

**Результат работы программы:**

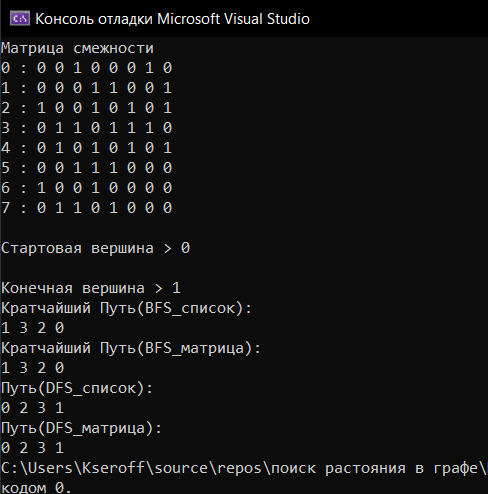


Рисунок 1- результат работы на 8 вершин

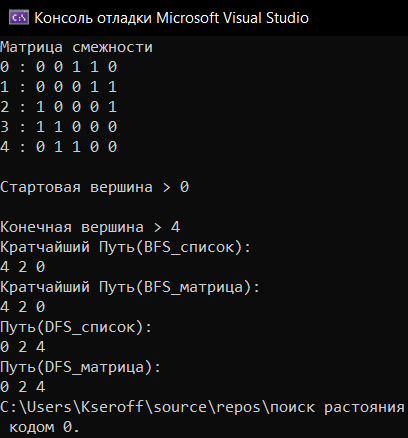


Рисунок 2- результат работы на 5 вершин

**Листинг:**

**Lab\_9.cpp**

#include <iostream>

#include <list>

#include <queue>

using namespace std;

template<typename T>

class MyList {

public:

class Iterator;

MyList() : size(0), head(nullptr), tail(nullptr) {}

~MyList() {}

T GetSize() { return size; }

bool isEmpty() { return(head == nullptr); }

void push\_back(T data);

T& operator[](const T index);

Iterator begin() { return Iterator(head); }

Iterator end() { return Iterator(nullptr); }

private:

template<typename T>

struct Node {

T data;

Node\* next;

Node\* pred;

Node(T data = T(), Node\* pNext = nullptr, Node\* pPred = nullptr) {

this->data = data;

this->next = pNext;

this->pred = pPred;

}

};

int size;

Node<T>\* head;

Node<T>\* tail;

};

template<typename T>

class MyList<T>::Iterator

{

Node<T>\* ptr;

public:

Iterator() : ptr(nullptr) {};

Iterator(Node<T>\* p) : ptr(p) {}

Iterator& operator ++ ()

{

if (ptr)

{

ptr = ptr->next; //Может быть еще и здесь кинуть исключение

return \*this;

}

}

Iterator& operator -- ()

{

if (ptr)

{

ptr = ptr->pred;

return \*this;

}

else

{

ptr = tail;

return \*this;

}

}

T& operator \* ()

{

if (ptr)

return ptr->data;

}

bool operator != (const Iterator& it) { return ptr != it.ptr; }

bool operator == (const Iterator& it) { return ptr == it.ptr; }

};

template<typename T>

void MyList<T>::push\_back(T data)

{

Node<T>\* temp = new Node<T>(data);

temp->next = 0;

temp->data = data;

temp->pred = tail;

if (tail != nullptr)

tail->next = temp;

if (size == 0) {

head = tail = temp;

}

else {

tail = temp;

}

size++;

}

template<typename T>

T& MyList<T>::operator[](const T index)

{

int counter\_1 = 0, counter\_2 = size;

Node<T>\* current\_1 = this->head;

Node<T>\* current\_2 = this->tail;

while (current\_1 != nullptr || current\_2 != nullptr)

{

if (counter\_1 == index)

return current\_1->data;

if (counter\_2 == index + 1)

return current\_2->data;

current\_1 = current\_1->next;

current\_2 = current\_2->pred;

counter\_2--;

counter\_1++;

}

}

class Graph {

private:

bool\* visited;

/////////////////

bool\*\* adjMatrix;

MyList<int>\* adjLists;

int numVertices;

public:

Graph(int numVertices)

{

this->numVertices = numVertices;

adjMatrix = new bool\* [numVertices];

visited = new bool[numVertices];

adjLists = new MyList<int>[numVertices];

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

adjMatrix[i] = new bool[numVertices];

visited[i] = false;

for (int j = 0; j < numVertices; j++)

adjMatrix[i][j] = false;

}

}

~Graph() {

for (int i = 0; i < numVertices; i++)

delete[] adjMatrix[i];

delete[] adjMatrix;

delete[]visited;

}

void memseting(bool a) {

for (int i = 0; i < numVertices; i++)

visited[i] = a;

}

void toMatrix() {

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

cout << i << " : ";

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

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

}

cout << endl;

}

}

void addEdge(int i, int j) {

adjMatrix[i][j] = adjMatrix[j][i] = true;

adjLists[i].push\_back(j);

adjLists[j].push\_back(i);

}

void BFS\_S(int start, int dest);

void BFS(int start, int dest);

void DFS\_S(int st, int dest);

void DFS(int v, int dest);

};

void Graph::BFS\_S(int start, int dest)

{

int\* dist = new int[numVertices];

int\* pred = new int[numVertices];

queue<int> queue;

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

dist[i] = INT\_MAX;

pred[i] = -1;

}

visited[start] = true;

dist[start] = 0;

queue.push(start);

while (!queue.empty()) {

int u = queue.front();

queue.pop();

for (int i = 0; i < adjLists[u].GetSize(); i++) {

if (!visited[adjLists[u][i]]) {

visited[adjLists[u][i]] = true;

queue.push(adjLists[u][i]);

dist[adjLists[u][i]] = dist[u]++;

pred[adjLists[u][i]] = u;

if (adjLists[u][i] == dest) {

cout << dest << " ";

while (pred[dest] != -1) {

cout << pred[dest] << " ";

dest = pred[dest];

}

delete[]dist;

delete[]pred;

return;

}

}

}

}

delete[]dist;

delete[]pred;

}

void Graph::BFS(int start, int dest)

{

int\* dist = new int[numVertices];

int\* pred = new int[numVertices];

queue<int> queue;

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

dist[i] = INT\_MAX;

pred[i] = -1;

}

visited[start] = true;

dist[start] = 0;

queue.push(start);

while (!queue.empty()) {

start = queue.front();

queue.pop();

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

if (adjMatrix[start][i] == 1 && (!visited[i])) {

visited[i] = true;

queue.push(i);

dist[i] = dist[start]++;

pred[i] = start;

if (i == dest) {

cout << dest << " ";

while (pred[dest] != -1) {

cout << pred[dest] << " ";

dest = pred[dest];

}

delete[]dist;

delete[]pred;

return;

}

}

}

}

delete[]dist;

delete[]pred;

}

void Graph::DFS\_S(int start, int dest)

{

visited[start] = true;

cout << start << " ";

if (start == dest)

memseting(1);

for (auto i : adjLists[start])

if (!visited[i])

DFS\_S(i, dest);

}

void Graph::DFS(int start, int dest)

{

visited[start] = true;

cout << start << " ";

if (start == dest)

memseting(1);

for (int i = 0; i <= numVertices; i++)

if ((adjMatrix[start][i] != 0) && (!visited[i]))

DFS(i, dest);

}

int main()

{

setlocale(0, "");

srand(time(0));

int z = 8;

Graph c(z);

for (int i = 0; i < z; i++)

{

for (int j = i + 1; j < z; j++)

{

if (bool(rand() % 2))

c.addEdge(i, j);

}

}

cout << "Матрица смежности" << endl;

c.toMatrix();

int start, dest;

cout << endl << "Стартовая вершина > "; cin >> start;

cout << endl << "Конечная вершина > "; cin >> dest;

cout << "Кратчайший Путь(BFS\_список): " << endl;

c.memseting(0);

c.BFS\_S(start, dest);

cout<<endl << "Кратчайший Путь(BFS\_матрица):" << endl;

c.memseting(0);

c.BFS(start, dest);

cout<<endl << "Путь(DFS\_список): " << endl;

c.memseting(0);

c.DFS\_S(start, dest);

cout << endl << "Путь(DFS\_матрица):" << endl;

c.memseting(0);

c.DFS(start, dest);

return 0;

}

**Вывод:** мы научилисьрешать задач, использующие графовые модели представления, изучили различные методы решения и анализа. Реализовали поиск кратчайшего пути в графе с помощью прохода графа в ширину и поиск пути с помощью прохода графа в глубину.