Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по теме «Обход графа в ширину»

Выполнили:

Студенты группы 21ВВ2

Назаров Е.А.

Макаров И.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза, 2022

**Цель работы:**

Изучить основные алгоритмы обхода графа и научиться решать задачи обхода графа на основе поиска в ширину.

**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 обхода в ширину, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

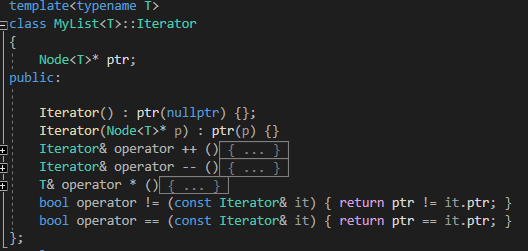
**Задание 2\***

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

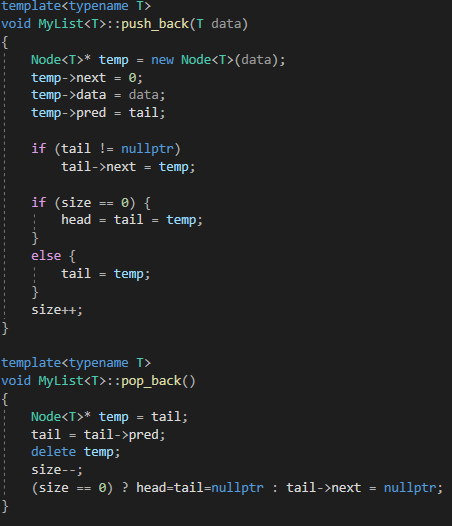
**Пояснительный текст к программе:**



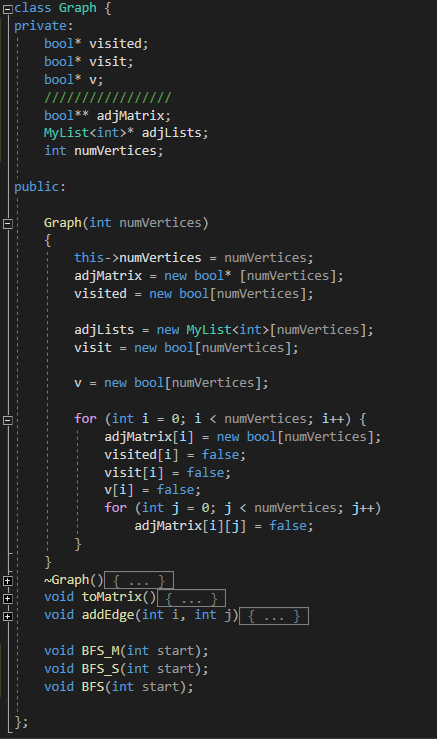
Был реализован динамический шаблонный двухсвязный список. Класс MyList представляет собой целостный класс с описанными функциями в нем. В классе был реализован класс Node элемента, содержащий в себе всю информацию и класс Iterator для перемещения по списку.



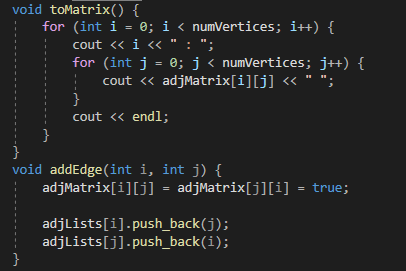
Содержит в себе в большей степени операторов перегрузки для хождения от одного элемента класса к другому. Функции, возвращающие начало и конец были вынесены в класс MyList.



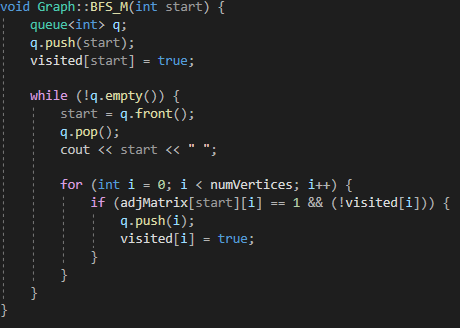
Функции добавления в конец списка(push\_back) и удаления последнего элемента списка(pop\_back).



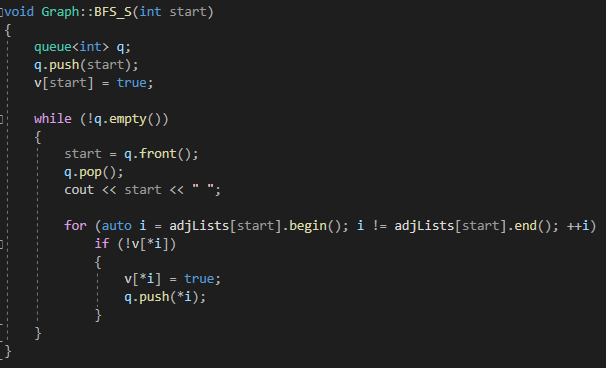
Класс содержащий в себе информацию о графе. В конструктор поступает количество вершин, после чего выделяется динамическая память для всех нужных элементов и массивов. Деструктор же освобождает эту память в конце программы.



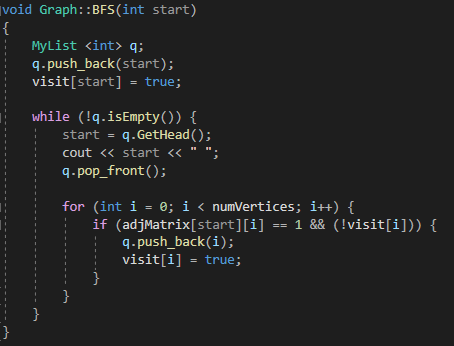
Функции вывода матрицы смежности на экран(toMatrix) и добавления связей между вершинами(addEdge).



Функция обхода графа в ширину для матрицы смежности. Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности, и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему.



Функция обхода графа в ширину для списка смежности. Алгоритм очень схож с выше перечисленным, но есть отличия. Сначала создаем queue для выполнения BFS, помечаем исходную вершину как обнаруженную. Ставим исходную вершину в queue и запускам цикл до тех пор, пока queue не станет пустой. В цикле мы сначала удаляем передний узел из очереди и печатаем его, потом запускаем цикл для работы над каждым ребром (v, u). Помечаем вершины как обнаруженные и ставим в queue.



Функция обхода графа в ширину для матрицы смежности, но очередь реализована самостоятельно с помощью класса MyList.

**Тестирование программы:**

|  |  |
| --- | --- |
| Количество вершин | Реализация |
| |  |  |  | | --- | --- | --- | | Матрица смежности  queue | Список смежности | Матрица смежности  MyList | |
| 100 | |  |  |  | | --- | --- | --- | | 0.013 | 0.012 | 0.011 | |
| 200 | |  |  |  | | --- | --- | --- | | 0.026 | 0.026 | 0.024 | |
| 400 | |  |  |  | | --- | --- | --- | | 0.076 | 0.067 | 0.063 | |
| 1 000 | |  |  |  | | --- | --- | --- | | 0.214 | 0.214 | 0.17 | |

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

queue)

**Результат работы программы:**

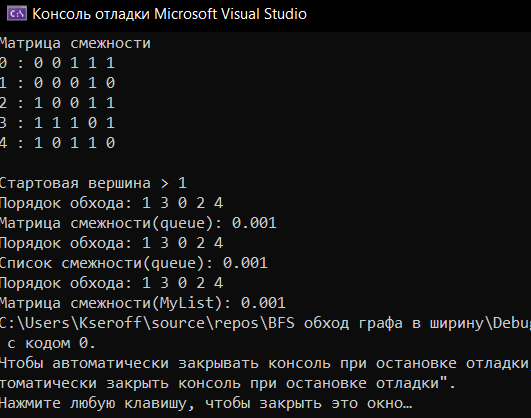


Рисунок 1- результат работы на 5 вершин

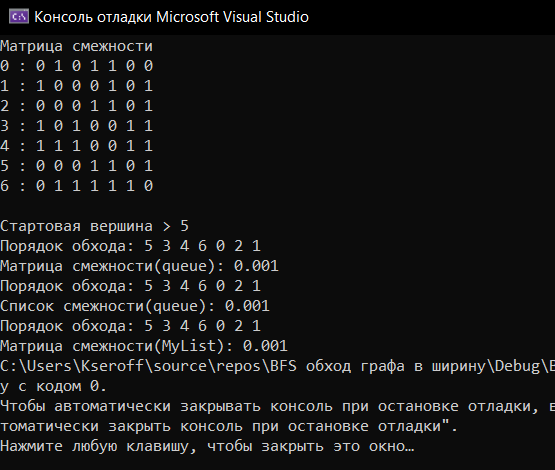


Рисунок 2- результат работы на 7 вершин

**Листинг:**

**BFS**

#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() { while (head != nullptr) pop\_front(); }

T GetSize() { return size; }

T GetHead() { return head->data; }

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

void push\_back(T data);

void pop\_front();

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; //Может быть еще и здесь кинуть исключение

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>

void MyList<T>::pop\_front()

{

Node<T>\* temp = head;

head = head->next;

delete temp;

size--;

(size == 0) ? head = tail = nullptr : head->pred = nullptr;

}

class Graph {

private:

bool\* visited;

bool\* visit;

bool\* v;

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

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];

visit = new bool[numVertices];

v = new bool[numVertices];

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

adjMatrix[i] = new bool[numVertices];

visited[i] = false;

visit[i] = false;

v[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;

delete[]visit;

delete[]v;

}

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\_M(int start);

void BFS\_S(int start);

void BFS(int start);

};

void Graph::BFS\_M(int start) {

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

start = q.front();

q.pop();

cout << start << " ";

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

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

q.push(i);

visited[i] = true;

}

}

}

}

void Graph::BFS\_S(int start)

{

queue<int> q;

q.push(start);

v[start] = true;

while (!q.empty())

{

start = q.front();

q.pop();

cout << start << " ";

for (auto i = adjLists[start].begin(); i != adjLists[start].end(); ++i)

if (!v[\*i])

{

v[\*i] = true;

q.push(\*i);

}

}

}

void Graph::BFS(int start)

{

MyList <int> q;

q.push\_back(start);

visit[start] = true;

while (!q.isEmpty()) {

start = q.GetHead();

cout << start << " ";

q.pop\_front();

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

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

q.push\_back(i);

visit[i] = true;

}

}

}

}

int main()

{

setlocale(0, "");

srand(time(0));

int z = 7;

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;

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

cout << "Порядок обхода: ";

double time1 = clock();

c.BFS\_M(start);

double time2 = clock();

cout << endl << "Матрица смежности(queue): "<< (time2 - time1) / CLOCKS\_PER\_SEC;

cout<<endl << "Порядок обхода: ";

time1 = clock();

c.BFS\_S(start);

time2 = clock();

cout << endl << "Список смежности(queue): " << (time2 - time1) / CLOCKS\_PER\_SEC;

cout<<endl << "Порядок обхода: ";

time1 = clock();

c.BFS(start);

time2 = clock();

cout << endl << "Матрица смежности(MyList): " << (time2 - time1) / CLOCKS\_PER\_SEC;

return 0;

}

**Вывод:** Изучили основные алгоритмы обхода графа и научились решать задачи обхода графа на основе поиска в ширину.