Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили:

студенты группы 22ВВВ1

Дунюшкин В.А.

Ивлюшин М.Д.

Проверили:

к.т.н, доцент Юрова О.В.

к.э.н, доцент Акифьев И.В.

Пенза 2023

**Цель работы:**

Цель научиться использовать алгоритм обхода графа в ширину

**Лабораторное задание:**

**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.

**Ход работы:**

Данный код реализует обход в ширину для графа, заданного матрицей смежности и списком смежности. Создается матрица смежности и список смежности на основе нее. Затем происходит обход в ширину для матрицы смежности и для списка смежности.

Алгоритм обхода в глубину можно реализовать как с помощью рекурсии, так и с использованием связанного списка

1. С помощью связанного списка(из ЛР№3):

- Создание структуры для хранение узла и класс для реализации очереди.

- Создаём массивы для отметки посещения вершин и для результата.

-Обходим все вершины.

-Если вершину ещё не обходили запускаем из неё обход в ширину.

-Пока очередь не опустеет извлекаем вершину из очереди и добавляем в результат.

-Проверяем все смежные вершины.

-Если смежная вершина не посещена помещаем в очередь.

2. С использованием алгоритма из ЛР№8:

- Проверяем с вершины i=0 была ли она посещена.

- Если нет, то создаётся очередь q и вершина добавляется в неё

- После устанавливается флаг в true, чтобы отметить вершину как посещённую.

- Далее цикл выполняется, пока очередь не станет пустой.

- На каждой итерации извлекается вершина v из очереди.

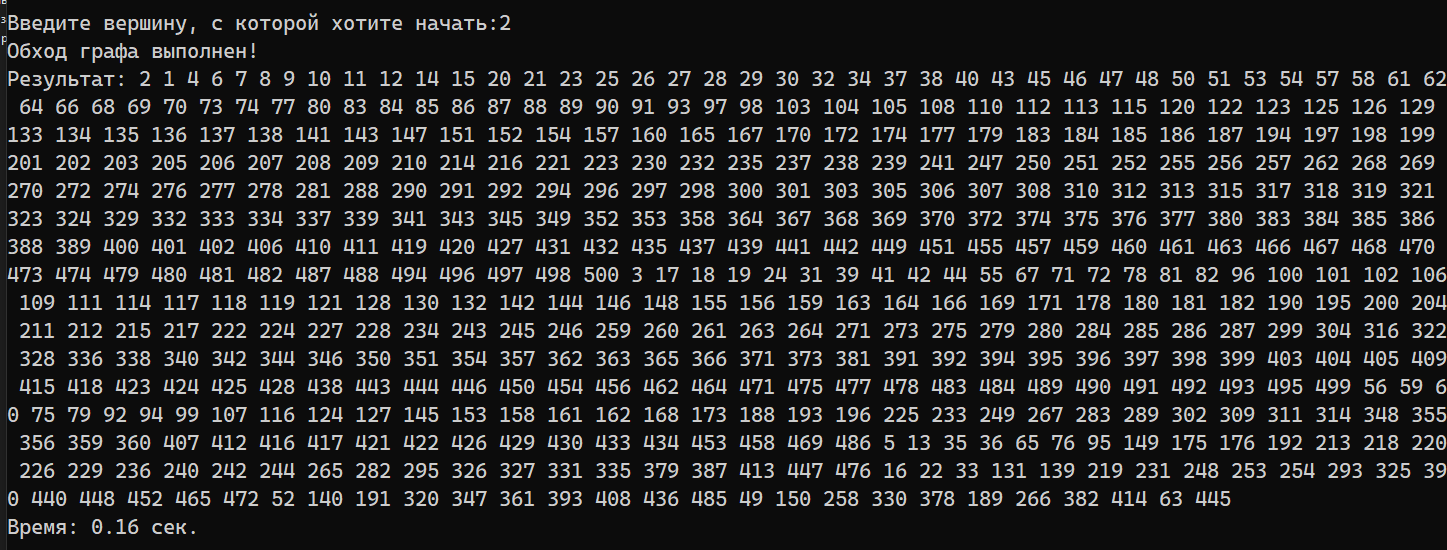
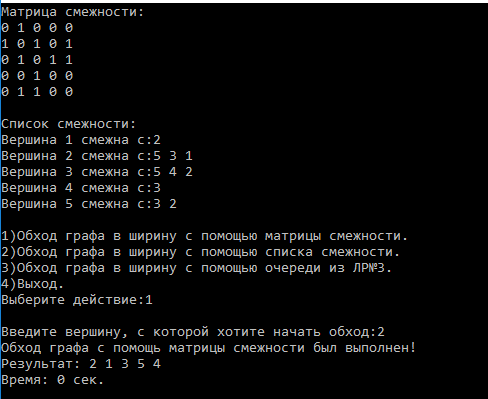
- Затем проходит проверка всех смежных вершин вершины v.

- Если вершина j не была посещена и между вершинами v и j есть ребро, то вершина добавляется в очередь.

-После флаг устанавливается в true.

Оба подхода дают одинаковый результат и реализуют обход графа в ширину.

**Результаты работы программы:**



Задание 2:

Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Оценка:**

Временная сложность обхода в ширину для обеих реализаций очереди будет одинакова - O(V+E), где V - количество вершин, E - количество ребер.

Однако константы времени выполнения будут различаться из-за разной производительности стандартной очереди std::queue и очереди на связном списке.

При одинаковом количестве вершин и ребер алгоритм на std::queue будет работать быстрее, так как:

Добавление и удаление элементов из std::queue происходит за константное время O(1) за счет внутренней реализации на основе двухсвязного списка.

В очереди на односвязном списке добавление происходит быстро O(1), а удаление - O(n), так как нужно найти голову списка.

Поэтому можно сделать вывод, что при одинаковом количестве операций обход на std::queue будет работать быстрее примерно в разы, чем на связном списке.

Это различие в константах времени работы будет тем более заметно, чем больше вершин и ребер в графе.

Таким образом, для повышения производительности алгоритма лучше использовать std::queue.

**Вывод**

В ходе лабораторной работы научились работать с обходом графа в ширину для матрицы смежности и списка смежности, также научились реализовывать алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3. Оценили время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Листинг**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cstdio>

#include <queue>

using namespace std;

int right = 0;

typedef struct Spis

{

int inf;

struct Spis\* next;

}SP;

typedef struct Queue

{

SP\* front;

SP\* rear;

}Queue;

int i, j, ver;

bool isol(int\*\* smej, int ver, int curr)

{

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

{

if (smej[curr][i] == 1 || smej[i][curr] == 1)

{

return false;

}

}

return true;

}

void BypassMatrix(int\*\* smej, int\* visit, int ver, int ver1)

{

queue<int> q; // пустая очередь

visit[ver1 - 1] = 1;

q.push(ver1);

while (!q.empty())

{

int current = q.front();

q.pop();

if (isol(smej, ver, current - 1)) cout << "вершина " << current << " является изолированной";

else cout << current << " ";

{

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

{

if (smej[current - 1][i] == 1 && !visit[i] && !isol(smej, ver, current - 1))

{

visit[i] = 1;

q.push(i + 1);

}

}

}

}

}

bool isolSpis(SP\*\* MasSpis, int curr)

{

if (MasSpis[curr] == NULL)

{

return true;

}

return false;

}

void BypassSpis(SP\*\* MasSpis, int ver2, int\* visit)

{

queue<int> q; // пустая очередь

visit[ver2 - 1] = 1;

q.push(ver2);

while (!q.empty())

{

int currentVer = q.front();

q.pop();

if (isolSpis(MasSpis, currentVer - 1)) cout << "вершина " << currentVer << " является изолированной";

else cout << currentVer << " ";

SP\* current = MasSpis[currentVer - 1];

while (current != NULL)

{

int temp = current->inf;

if (!visit[temp - 1] && !isolSpis(MasSpis, temp - 1))

{

visit[temp - 1] = 1;

q.push(temp);

}

current = current->next;

}

}

}

Queue\* initializeQueue()

{

Queue\* q = new Queue();

if (q == NULL)

{

cout << "Ошибка выделения памяти для очереди!" << endl;

exit(1);

}

q->front = q->rear = NULL;

return q;

}

void enqueue(Queue\* q, int ver)

{

SP\* newSP = new SP();

if (newSP == NULL)

{

cout << "Ошибка выделения памяти для нового элемента!" << endl;

exit(1);

}

newSP->inf = ver;

newSP->next = NULL;

if (q->rear == NULL) q->front = q->rear = newSP;

q->rear->next = newSP;

q->rear = newSP;

}

int dequeue(Queue\* q)

{

if (q->front == NULL) {

cout << "Очередь пуста!" << endl;

exit(1);

}

SP\* temp = q->front;

int item = temp->inf;

q->front = temp->next;

if (q->front == NULL) q->rear = NULL;

return item;

}

void BypassOwnQueue(int\*\* smej, int ver, int ver3, int\* visit)

{

Queue\* q = initializeQueue();

visit[ver3 - 1] = 1;

enqueue(q, ver3);

while (q->front != NULL)

{

int current = dequeue(q);

if (isol(smej, ver, current - 1)) cout << "вершина " << current << " является изолированной";

else { if (current != ver3) { cout << current << " "; } }

for (i = 0; i < ver; i++)

{

if (!visit[i] && smej[current - 1][i] == 1 && !isol(smej, ver, current - 1))

{

visit[i] = 1;

enqueue(q, i + 1);

}

}

}

delete q;

delete[] visit;

}

void main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* smej = 0; //Матрица смежности

int choice, ver1, ver2, ver3;

SP\*\* MassSpis = NULL;//Массив списков смежности

int\* visit = 0;

bool prov1 = true, prov2 = true, prov3 = true;

clock\_t start, end;

double TT = 0.0;

cout << "Введите количество вершин графа: ";

for (;;) {

cin >> ver;

if (cin.fail() || ver < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер количество вершин графа! Повторите попытку: ";

}

else { break; }

}

cout << endl;

//Выделение памяти под массив смежности

smej = new int\* [ver];

if (smej == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

smej[i] = new int[ver];

}

//Выделение памяти под массив списков

MassSpis = new SP \* [ver];

if (MassSpis == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

MassSpis[i] = NULL;

}

//Генерация массива смежности

for (i = 0; i < ver; i++)

{

for (j = i; j < ver; j++)

{

if (i == j)

{

smej[i][j] = 0; // на главной диагонали нули

}

else

{

smej[i][j] = rand() % 2; // случайные значения 0 или 1

smej[j][i] = smej[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

//Генерация списка смежности

for (i = 0; i < ver; i++)

{

for (j = 0; j < ver; j++)

{

if (smej[i][j] == 1)

{

SP\* newSP = new SP;

newSP->inf = j + 1;

newSP->next = MassSpis[i];

MassSpis[i] = newSP;

}

}

}

//Вывод матрицы смежности

cout << "Матрица смежности:" << endl;

for (i = 0; i < ver; i++)

{

for (j = 0; j < ver; j++)

{

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

}

cout << endl;

}

cout << endl;

/\*Вывод списка смежности

cout << "Список смежности:" << endl;

for (i = 0; i < ver; i++)

{

if (MassSpis[i] == NULL)

{

cout << "Вершина " << i + 1 << " не имеет смежных";

}

else

{

cout << "Вершина " << i + 1 << " смежна с:";

SP\* current = MassSpis[i];

while (current != NULL)

{

cout << current->inf << " ";

current = current->next;

}

}

cout << endl;

}

\*/

do

{

cout << endl;

cout << "1)Обход графа в ширину с помощью матрицы смежности." << endl;

cout << "2)Обход графа в ширину с помощью списка смежности." << endl;

cout << "3)Обход графа в ширину с помощью очереди из ЛР№3." << endl;

cout << "4)Выход." << endl;

cout << "Выберите действие:";

cin >> choice;

switch (choice)

{

case 1:

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

visit[i] = 0;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать обход:";

for (;;) {

cin >> ver1;

if (cin.fail() || (ver1 < 1 || ver1 > ver)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

}

else { prov1 = false; break; }

}

} while (prov1);

cout << "Обход графа с помощь матрицы смежности был выполнен!" << endl;

cout << "Результат: ";

start = clock();

BypassMatrix(smej, visit, ver, ver1);

end = clock();

cout << endl;

TT = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 2:

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

visit[i] = 0;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать:";

for (;;) {

cin >> ver2;

if (cin.fail() || (ver2 < 1 || ver2 > ver)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

}

else { prov2 = false; break; }

}

}

while (prov2);

cout << "Обход графа с помощь списка смежности был выполнен!" << endl;

cout << "Результат:";

start = clock();

BypassSpis(MassSpis, ver2, visit);

end = clock();

cout << endl;

TT = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "Время: " << TT << " сек.";

cout << endl;

delete[] visit;

break;

case 3:

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

visit[i] = 0;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать:";

for (;;) {

cin >> ver3;

if (cin.fail() || (ver3 < 1 || ver3 > ver)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер вершину, с которой хотите начать! Повторите попытку: ";

}

else { prov3 = false; break; }

}

} while (prov3);

cout << "Обход графа выполнен!" << endl;

cout << "Результат: " << ver3 << " ";

start = clock();

BypassOwnQueue(smej, ver, ver3, visit);

end = clock();

cout << endl;

TT = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "Время: " << TT << " сек.";

cout << endl;

break;

case 4:

system("cls");

cout << "До свидания!";

break;

default:

cout << endl;

cout << "Неверный выбор! Повторите попытку.";

cout << endl;

break;

}

} while (choice != 4);

delete[] MassSpis;

for (i = 0; i < ver; i++)

{

delete[] smej[i];

}

delete[] smej;

}