Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили:

студент группы 23ВВВ2

Кокарев Д.С.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Название**

Обход графа в ширину.

**Цель работы**

Научиться выполнять обход 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.

**Описание метода решения задачи**

*Задание 1*

1. Сгенерировал (используя генератор случайных чисел) матрицу смежности для неориентированного графа G.

2. Для сгенерированного графа осуществил процедуру обхода в ширину, используя функцию void BFS1(int s). При реализации алгоритма в качестве очереди использовал класс **queue** из стандартной библиотеки С++.

3. По аналогичному алгоритму осуществил процедуру обхода в ширину для графа, представленного списками смежности (функция first\_list ).

*Задание 2*

1. Для матричной формы представления графов реализовал алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», которую создал самостоятельно в лабораторной работе № 3. Для этого были использованы функции: add\_to\_list\_queue(s) и del\_from\_list().

2. Оценил время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков. Для этого использовал следующие функции: add\_to\_list\_queue(s) и del\_from\_list().

**Листинг**

**lab8.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <ctime>

#include <locale>

#include <iostream>

#include <stack>

using namespace std;

stack <int> S;

int k;

int size1, h;

int\*\* arr1;

int\* vis;

struct Node {

int n;

Node\* next;

};

Node\* LA;

Node\* make\_LA() {

Node\* la = new Node[size1];

Node\* p;

for (int i = 0; i < size1; i++) {

la[i].n = i;

la[i].next = nullptr;

p = &la[i];

for (int j = 0; j < size1; j++) {

if (arr1[i][j] == 1) {

p->next = new Node;

p = p->next;

p->n = j;

p->next = nullptr;

}

}

}

return la;

}

void print\_LA() {

Node\* p;

for (int i = 0; i < size1; i++) {

p = &LA[i];

while (p != nullptr) {

cout << p->n << " > ";

p = p->next;

}

cout << endl;

}

}

void first\_list(int l) {

Node\* p;

p = &LA[l];

vis[l] = 1;

cout << l << ' ';

while (p->next != nullptr) {

if (vis[p->next->n] == 0) {

vis[p->next->n] = 1;

l = p->next->n;

first\_list(l);

}

p = p->next;

}

}

void create\_matrix() {

for (int i = 0; i < size1; i++) {

for (int j = i; j < size1; j++) {

if (i == j) {

arr1[i][j] = 0;

}

else {

arr1[i][j] = rand() % 2;

arr1[j][i] = arr1[i][j];

}

}

}

}

void print\_matrix() {

cout << " " << endl;

for (int i = 0; i < size1; i++) {

cout << i << ' ';

}

cout << endl;

cout << "-----------------------------------------------------------\n";

for (int i = 0; i < size1; i++) {

for (int j = 0; j < size1; j++) {

cout << arr1[i][j] << ' ';

}

cout << endl;;

}

cout << endl;;

}

void print\_vis() {

cout << "\nМассив vis:\n";

for (int i = 0; i < size1; i++) {

cout << vis[i];

}

cout << "\n\n";

}

void DFS1(int s) {

vis[s] = 1;

cout << s << ' ';

for (int i = 0; i < size1; i++) {

if ((arr1[s][i] == 1) && (vis[i] == 0)) {

DFS1(i);

}

}

}

void DES2(int s) {

vis[s] = 1;

cout << s << ' ';

S.push(s);

while (!S.empty()) {

s = S.top();

S.pop();

for (int i = size1 - 1; i >= 0; i--) {

if ((arr1[s][i] == 1) && (vis[i] == 0)) {

cout << i << ' ';

vis[i] = 1;

S.push(i);

}

}

}

}

void clean\_vis() {

for (int i = 0; i < size1; i++) {

vis[i] = 0;

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "");

cout << "Задание 1\n";

cout << "Введите размер матрицы смежности M1: ";

cin >> size1;

cout << endl;

vis = new int[size1];

arr1 = new int\* [size1];

for (int i = 0; i < size1; i++) {

arr1[i] = new int[size1];

}

cout << "Матрица M1:\n";

create\_matrix();

print\_matrix();

clean\_vis();

cout << "=======================================================================\n\n";

cout << "Задание 2\n";

cout << "Введите вершину от (0 до " << size1 - 1 << ") для начала обхода: ";

cin >> h;

int l = h;

k = h;

cout << endl;

cout << "был в вершинах: ";

DFS1(h);

cout << endl;

clean\_vis();

cout << endl;

cout << "=======================================================================\n\n";

cout << "Задание 3\n";

cout << "Список смежности S1:\n";

LA = make\_LA();

print\_LA();

cout << endl;

cout << "был в вершинах: ";

first\_list(l);

cout << endl;

cout << "=======================================================================\n\n";

cout << "Задание 4\n";

clean\_vis();

cout << "был в вершинах: ";

DES2(k);

for (int i = 0; i < size1; ++i) delete[] arr1[i];

delete[] arr1;

delete[] vis;

delete[] LA;

return 0;

}

**Результаты работы программы**

Результаты работы программы представлены на рисунках:

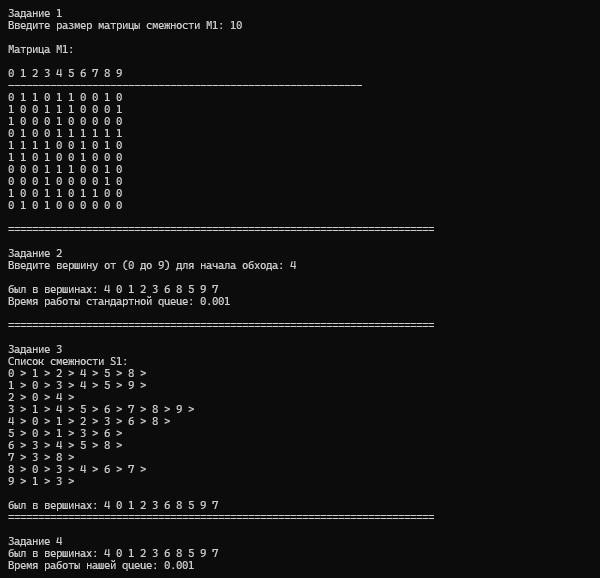
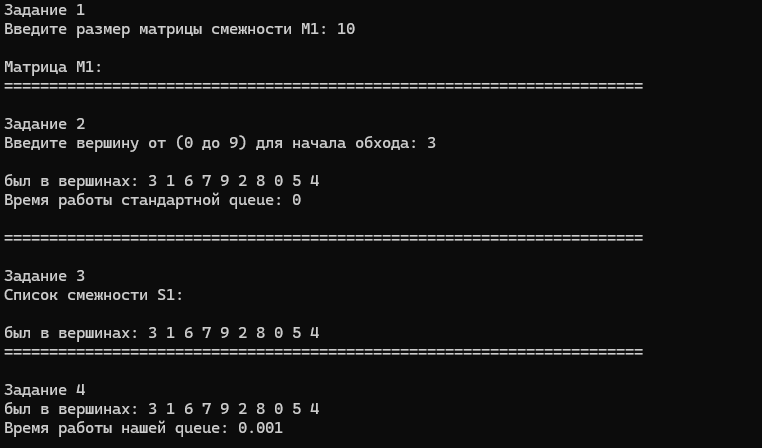
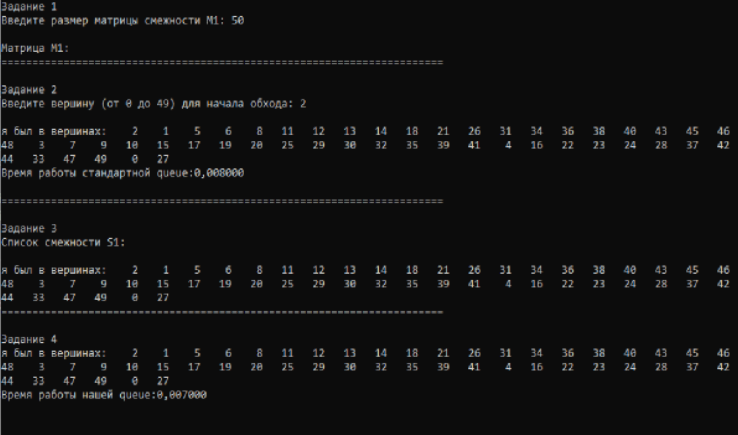


Рисунок 1 - Обход графа в ширину

Сравнение времени работы алгоритмов обхода в ширину, реализованных с помощью стандартной очереди и нашей очереди для графов разных порядков:

Рисунок 2 - Граф из 10 вершин

Рисунок 3 - Граф из 50 вершин

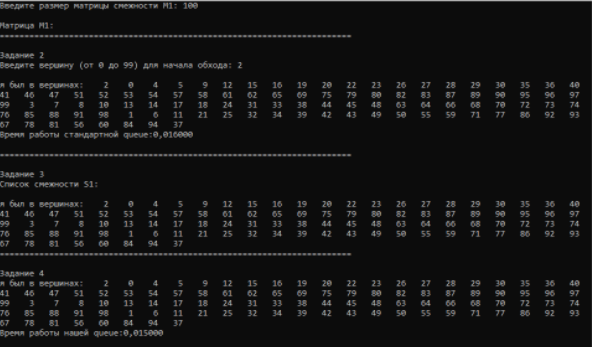
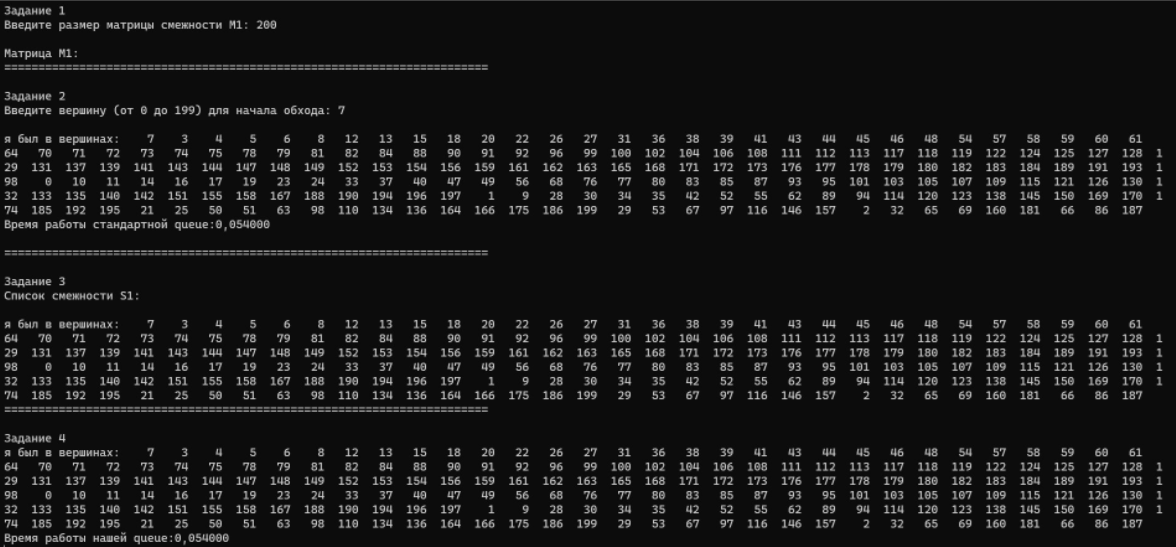
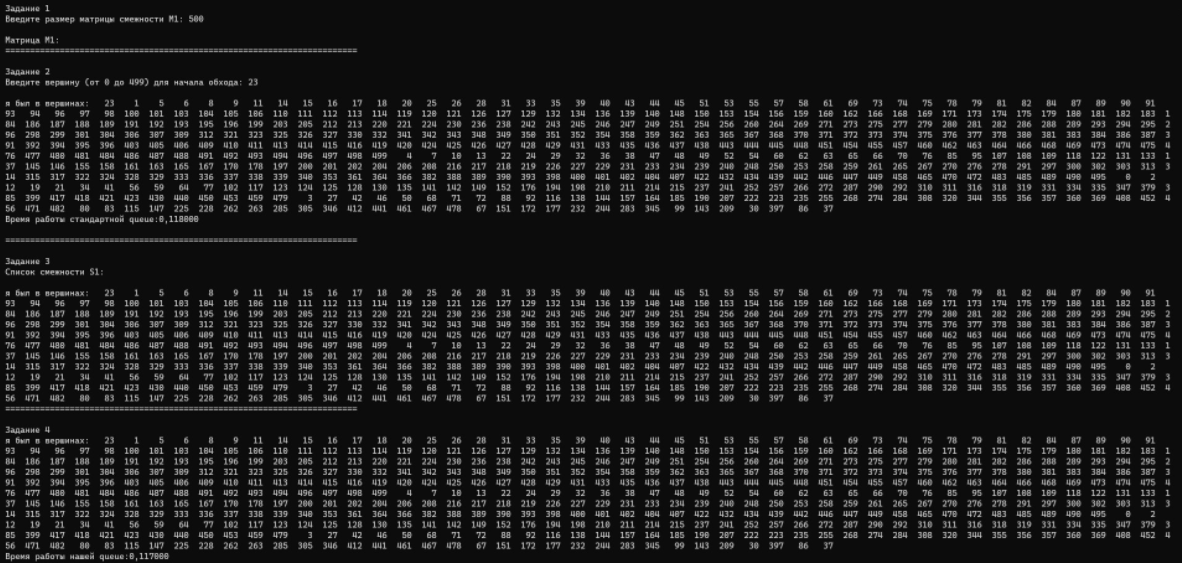
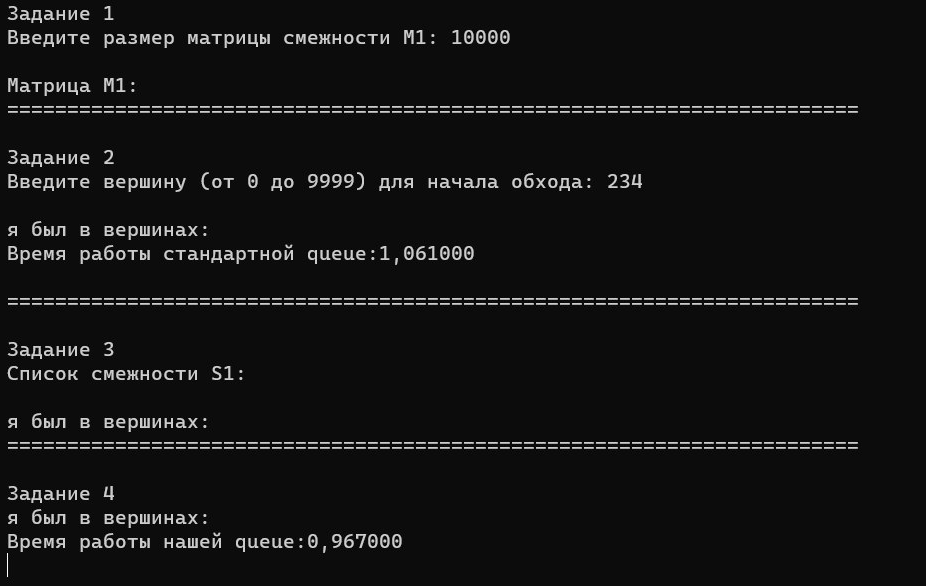


Рисунок 4 - Граф из 100 вершин

Рисунок 5 - Граф из 200 вершин

Рисунок 6 - Граф из 500 вершин

Рисунок 7 - Граф из 10.000 вершин

|  |  |  |
| --- | --- | --- |
| **Количество вершин графа** | **Стандартная queue** | **Наша queue** |
| 10 | 0,002 | 0,001 |
| 50 | 0,008 | 0,007 |
| 100 | 0,016 | 0,015 |
| 200 | 0,054 | 0,054 |
| 500 | 0,118 | 0,117 |
| 10000 | 1,061 | 0,967 |

Таким образом, функция, использующая очередь, которая реализована самостоятельно, работает быстрее, чем функция, использующая стандартный класс queue. Это связано с тем, что стандартная очередь queue работает с различными типами данных, а наша queue работает только с типом int.

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, выполняющая обход графа в ширину. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок. Также был получен опыт в создании проектов в среде Microsoft Visual Studio и приобретен навык программирования алгоритмов.