МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯРОССИЙСКОЙ ФЕДЕРАЦИИПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №8

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Обход графа в ширину"

Выполнили:

Студенты группы 24ВВВ3:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Изучение 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 C++.

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

Задание 2*

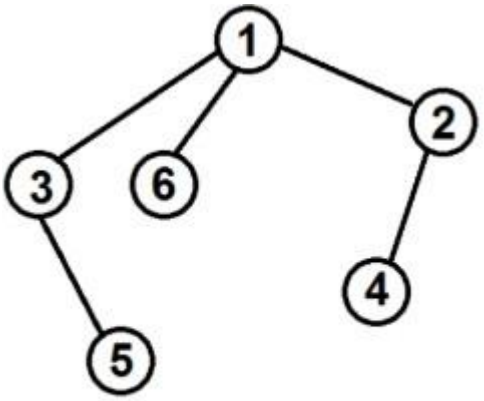
1 Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3

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 программам

Обход графа в ширину – еще один распространенный способ обхода графов.

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. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.



$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Рисунок 1 – Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности (с номерами 2, 3 и 6), и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещенных соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещенные соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

Псевдокод:

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОШ

1.1. для всех i положим NUM[i] = False пометим как "не посещенную";

1.2. **ПОКА** существует "новая" вершина v

1.3. **ВЫПОЛНЯТЬ** BFS (v).

Алгоритм BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. пометить v как "посещенную" NUM[v] = True;

2.4. **ПОКА** Q != ⊘ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО** size_G **ВЫПОЛНЯТЬ**

2.9. **ЕСЛИ** G(v,i) = = 1**И** NUM[i] = = False

2.10. **ТО**

2.11. Поместить i в очередь Q.push(i);

2.12. пометить v как "посещенную" NUM[v] = True;

Реализация состоит из подготовительной части, в которой все вершины помечаются как не посещенные (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3).

В самой процедуре обхода сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается v-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с v не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и помечается как посещенная (п.2.12). После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Так, если для графа на рисунке 1, мы начнем обход из первой вершины, то на шаге 2.2 она будет помещена в очередь и помечена как посещенная (NUM[1] = True). Условие цикла 2.4 будет выполнено, вершина 1 будет установлена в качестве текущей (п.2.5) и удалена из очереди (п.2.6). На экран будет выведена единица.

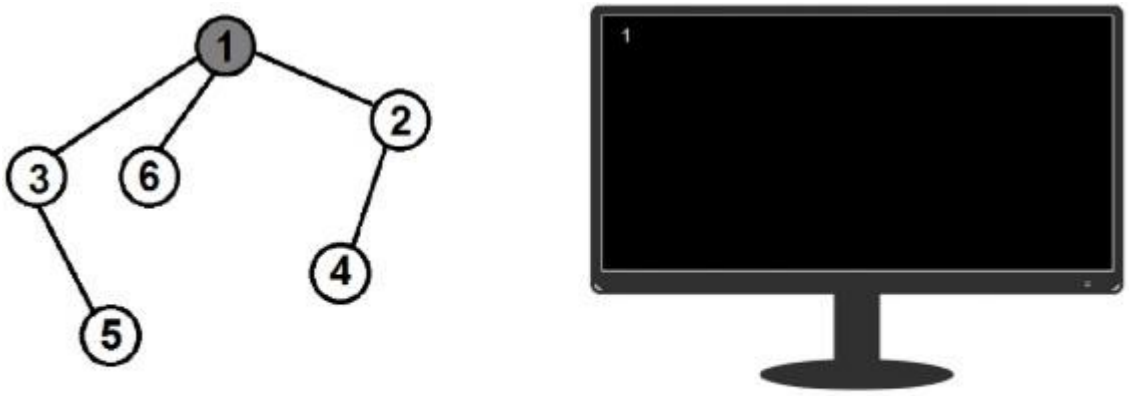


Рисунок 2 – Итерация 1

При просмотре 1-й строки матрицы смежности

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

будет найдена смежная вершина с индексом 2 (G(1,2) = = 1), которая не посещена (NUM[2] = = False) и она будет помещена в очередь и помечена (NUM[2] = = True). Просмотр строки продолжится, и в очередь будут также помещены и помечены как посещенные вершины с индексами 3 и 6.

К концу первой итерации очередь будет содержать Q = {2, 3, 6}.

Условие цикла while будет выполнено и на следующей итерации вершина 2 будет установлена как текущая и извлечена из очереди, на экран будет выведена двойка.

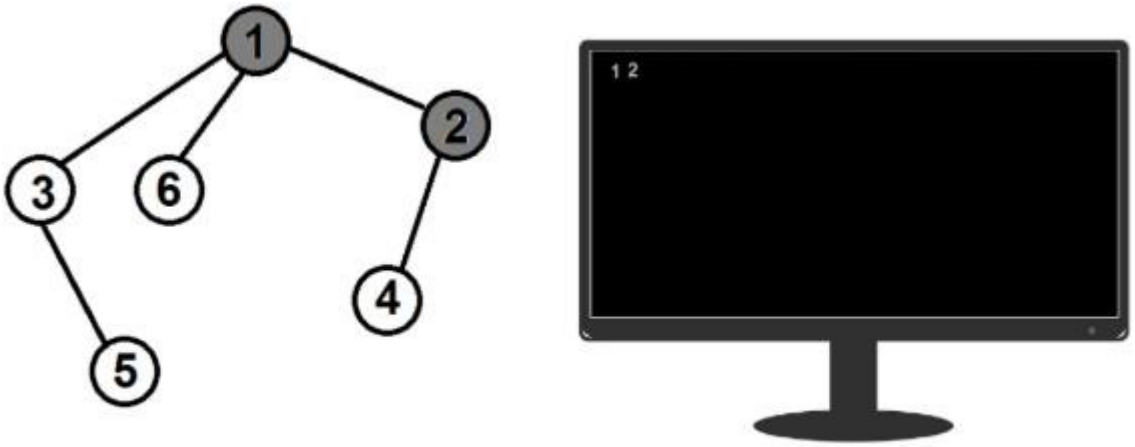


Рисунок 3 – Итерация 2

И алгоритм перейдет к просмотру второй строки матрицы смежности. Первая смежная с вершиной 2 - вершина с индексом 1(G(2,1) = =1),

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

которая к настоящему моменту уже посещена (NUM[1] = = True), она не будет помещена в очередь. Цикл 2.9 продолжит просмотр матрицы смежности.

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Следующая найденная вершина, смежная со второй, будет иметь индекс 4 (G(2,4) = =1), она не посещена (NUM[4] = = False) и она будет помещена в очередь и помечена.

К концу второй итерации очередь будет содержать Q = {3, 6, 4}.

Условие цикла while будет выполнено и на следующей итерации вершина 3 будет установлена как текущая и извлечена из очереди, на экран будет выведена тройка.

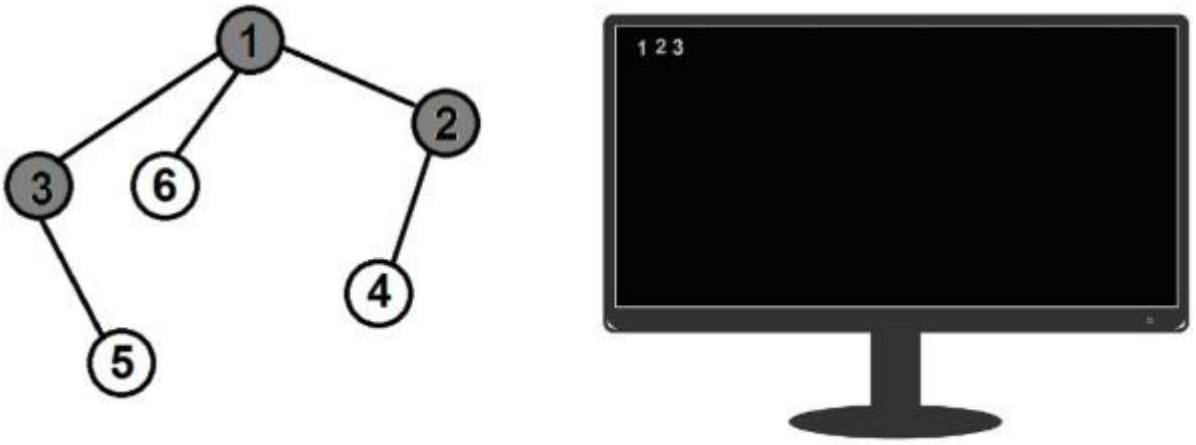


Рисунок 4 – Итерация 3

При просмотре 3-й строки матрицы будет найдена вершина 1, но она уже посещена (NUM[1] = True), поэтому в очередь помещена не будет.

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Следующая найденная вершина, смежная с третьей – вершина с номером 5 (G(3,5) = =1), она не посещена (NUM[5] = = False) и будет помещена в очередь.

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

К концу третьей итерации очередь будет содержать Q = {6, 4, 5}.

Условие цикла while будет выполнено. На следующей итерации вершина 6 будет установлена как текущая и извлечена из очереди, на экран будет выведена цифра шесть.

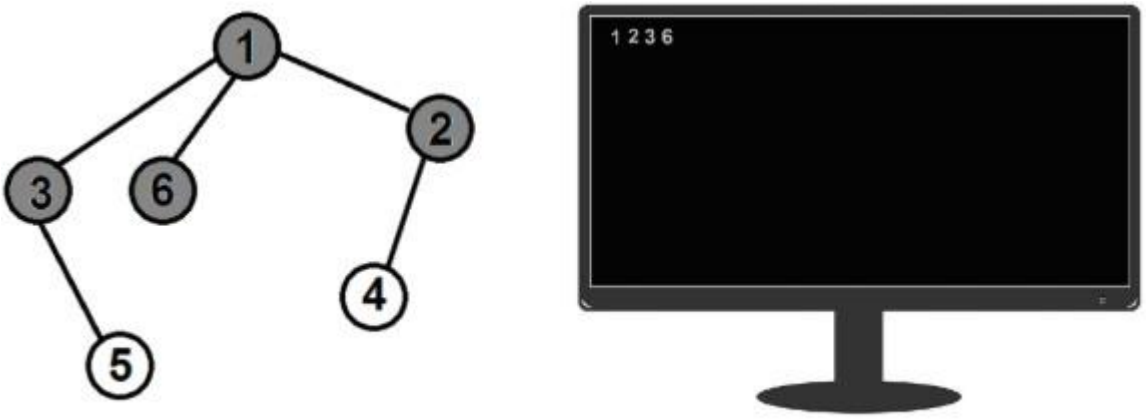


Рисунок 5 – Итерация 4

При просмотре 6-й строки матрицы будет найдена вершина 1, но она уже посещена (NUM[1] = True), поэтому в очередь помещена не будет.

К концу четвертой итерации очередь будет содержать Q = {4, 5}.

Далее из очереди будет извлечена вершина 4, установлена как текущая и выведена на экран.

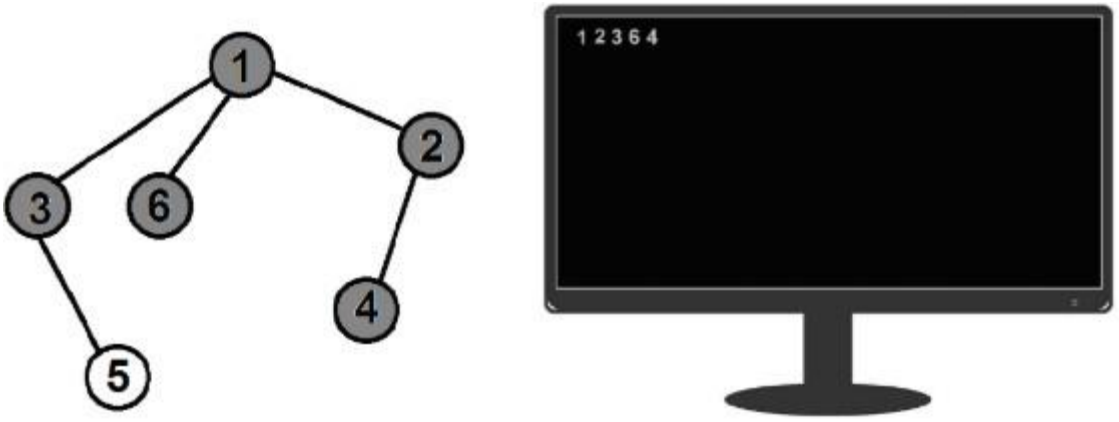


Рисунок 6 – Итерация 5

Так как все вершины кроме пятой уже посещены, то при просмотре 4-й строки в матрице смежности в очередь не будут добавлены вершины.

К концу пятой итерации очередь будет содержать только одну вершину Q = {5}.

На шестой итерации вершина с номером пять будет установлена как текущая и извлечена из очереди, на экран будет выведена цифра пять.

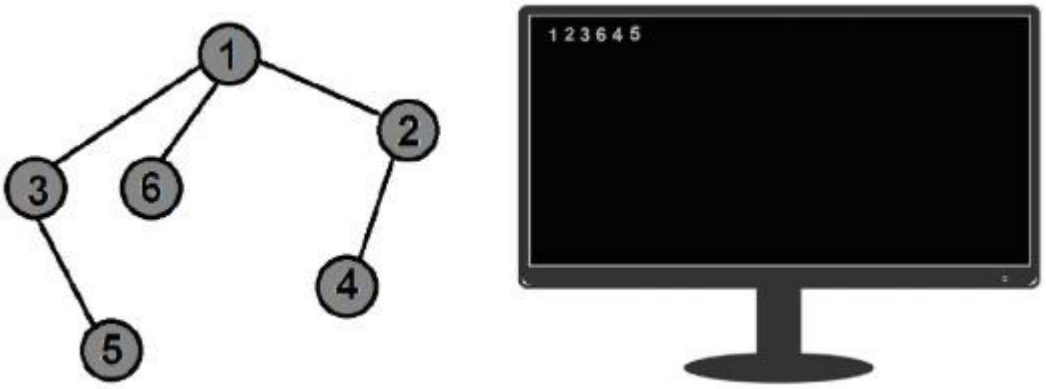


Рисунок 7 – Результат работы обхода

Просмотр 5-й строки матрицы смежности не добавит в очередь новых вершин, так как к этому моменту они уже все помечены как посещенные.

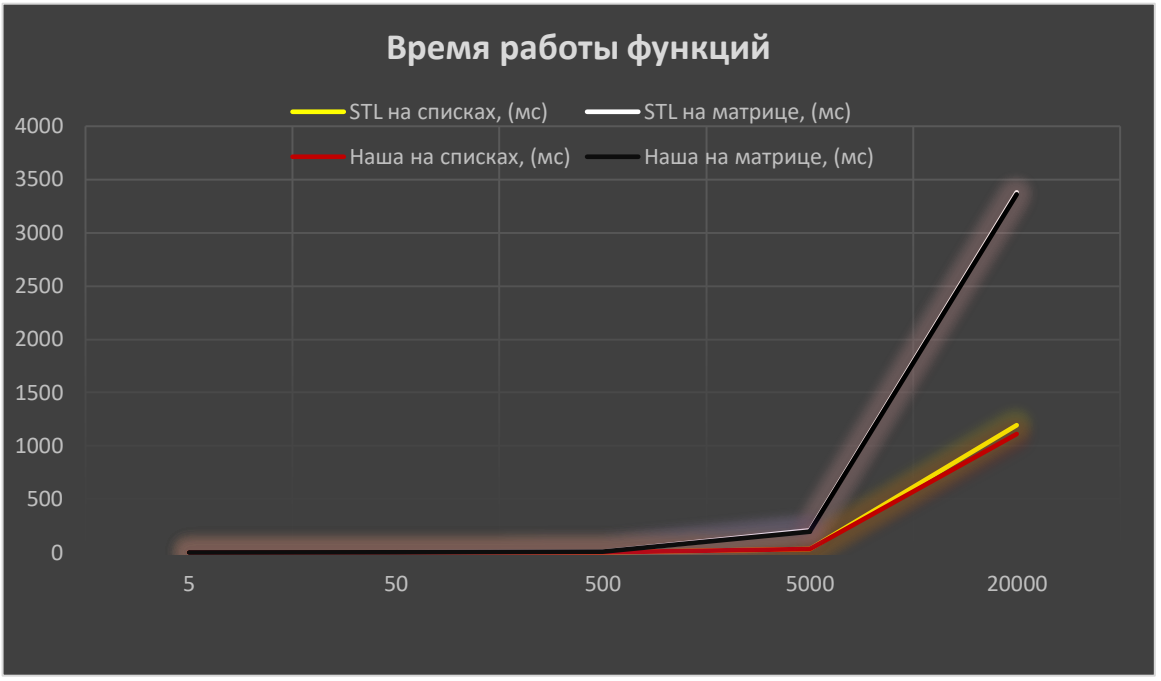
После этой итерации очередь окажется пуста Q = {}и алгоритм завершит свою работу.

В конце работы алгоритма все вершины будут посещены. А на экран будут выведены номера вершин в порядке их посещения алгоритмом.

Оценка времени работы

Оценили время работы функций

Количество вершин	**STL на списках, (мс)**	**STL на матрице, (мс)**	**Наша на списках, (мс)**	**Наша на матрице, (мс)**
5	0.014	0.096	0.013	0.013
50	0.138	0.245	0.118	0.158
500	2.587	5.562	1.370	4.677
5000	33.253	209.088	32.421	196.985
20000	1194.435	3379.606	1111.703	3361.194



Как мы видим, наша реализация функции queue и обход графа в ширину на списках смежности показал наилучший результат, STL queue на списках смежности показала результат немного хуже, но не критично, что мы не можем сказать о функциях обхода, реализованных на матрице смежности, они показали плохие результаты. Если эти результаты слабо замечались на маленьком количестве вершин, то с увеличением их числа, разница стремительно росла и становилась критичной.

Рисунок 8а - Результаты работы **lab8.cpp** на 5 вершинах

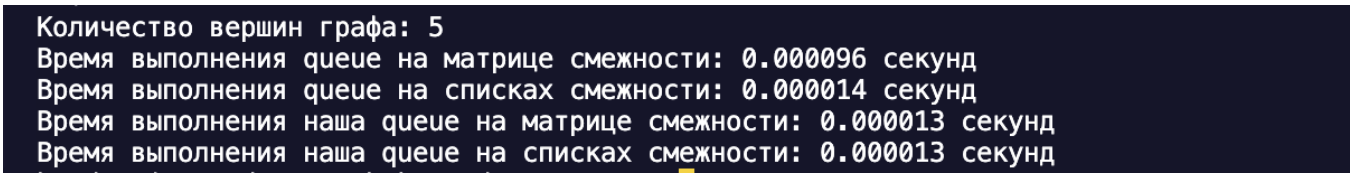


Рисунок 8б - Результаты работы **lab8.cpp** на 50 вершинах

Рисунок 8в - Результаты работы **lab8.cpp** на 500 вершинах

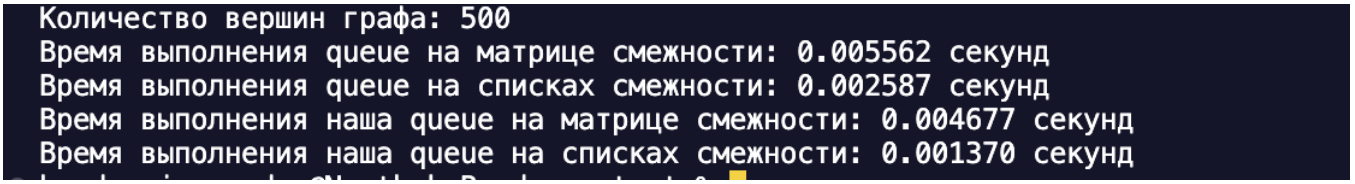


Рисунок 8г - Результаты работы **lab8.cpp** на 5000 вершинах

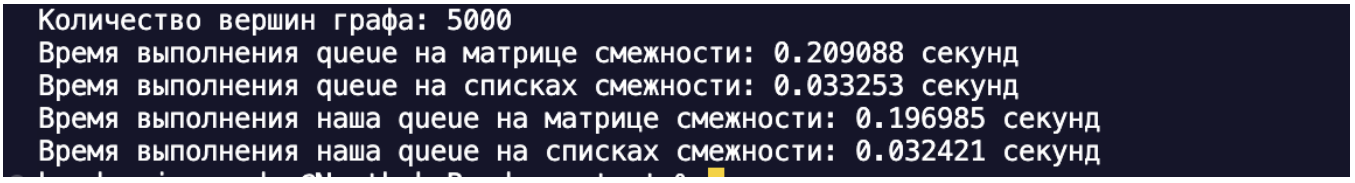
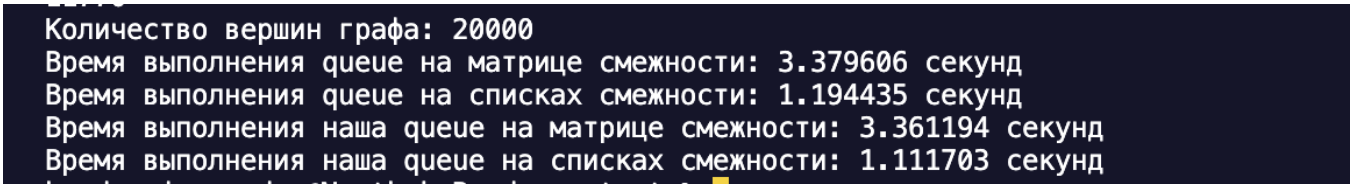
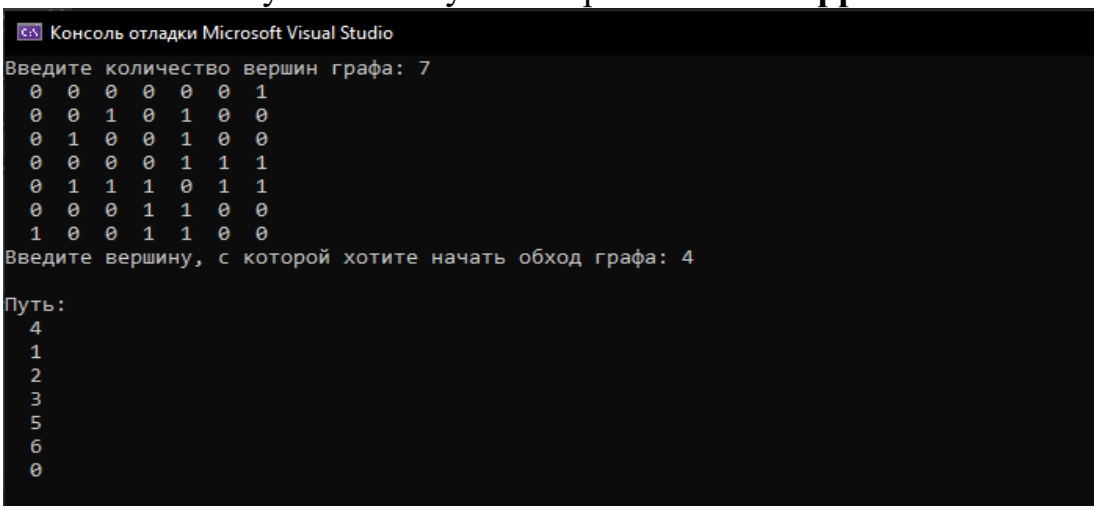


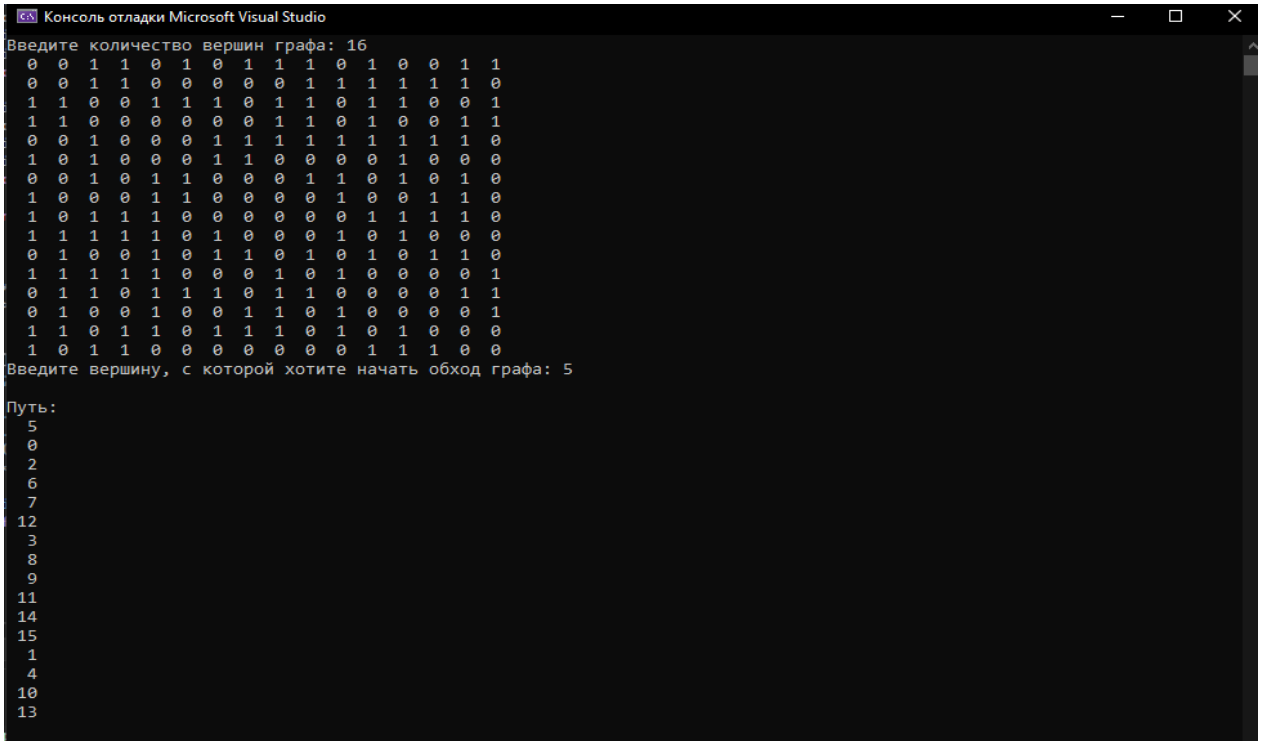
Рисунок 8д - Результаты работы **lab8.cpp** на 20000 вершинах



Результаты программ

Рисунок 9 - Результаты работы **laba8.cpp**





Вывод: в ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №8 – обход графа в глубину.

Приложение А Листинг

Файл laba8.cpp

```
// обход графа в ширину, доп со своей очередью списки смежности в 1 код
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <locale>
#include <limits>
#include <iomanip>
#include <queue>

using namespace std;

struct Node {
    int inf;
    Node* next;
};

void enqueue(Node *& head, Node *& tail, int value);
int dequeue(Node *& head, Node *& tail);
bool isEmpty(Node *head);

void bfsOurQueue(int **G, int numG, int *visited, int start);
void bfsOurQueueSpisok(Node **adj, int v, int *visited);

void bfsSTL(int **G, int numG, int *visited, int s);
void bfsSTLSpisok(Node **adj, int v, int *visited);

void clearScreen();
int isInteger(const string& message);

int main() {
    setlocale(LC_ALL, "Rus");
    clearScreen();
    srand(time(NULL));

    int numG = isInteger("Введите количество вершин графа: ");
    while (numG <= 0) {
        cout << "Ошибка! Количество вершин должно быть положительным\n";
        numG = isInteger("Введите количество вершин графа: ");
    }

    int** G = new int*[numG];
    for (int i = 0; i < numG; i++)
```

```
    G[i] = new int[numG];

int* visited = new int[numG]{0};

for (int i = 0; i < numG; i++) {
    for (int j = i; j < numG; j++) {
        G[i][j] = G[j][i] = (i == j ? 0 : rand() % 2);
    }
}

cout << "\nМатрица смежности:\n";
for (int i = 0; i < numG; i++) {
    for (int j = 0; j < numG; j++)
        cout << setw(3) << G[i][j];
    cout << "\n";
}

Node** adj = new Node*[numG];
for (int i = 0; i < numG; i++)
    adj[i] = nullptr;

for (int i = 0; i < numG; i++) {
    for (int j = numG - 1; j >= 0; j--) {
        if (G[i][j] == 1) {
            Node* p = new Node{ j, adj[i] };
            adj[i] = p;
        }
    }
}

cout << "\nСписки смежности:\n";
for (int i = 0; i < numG; i++) {
    cout << i << ": ";
    Node* cur = adj[i];
    while (cur) {
        cout << cur->inf << " ";
        cur = cur->next;
    }
    cout << "\n";
}

int current = isInteger("\nВведите вершину, с которой хотите начать обход графа: ");
while (current < 0 || current >= numG) {
    cout << "Ошибка! Вершина должна быть в диапазоне [0," << numG-1 << "]\n";
    current = isInteger("Введите вершину: ");
```

```
    }

    clock_t t1 = clock();
    cout << "\nПуть (BFS с queue):\n";
    bfsSTL(G, numG, visited, current);
    clock_t t2 = clock();
    double timeSTL = double(t2 - t1) / CLOCKS_PER_SEC;;

    for (int i = 0; i < numG; i++)
    visited[i] = 0;

    clock_t t3 = clock();
    cout << "\nПуть (BFS с queue списки смежности):\n";
    bfsOurQueueSpisok(adj, current, visited);
    clock_t t4 = clock();
    double timeSTLSpisok = double(t4 - t3) / CLOCKS_PER_SEC;

    for (int i = 0; i < numG; i++)
        visited[i] = 0;

    clock_t t5 = clock();
    cout << "\nПуть (BFS с queue собственная):\n";
    bfsOurQueue(G, numG, visited, current);
    clock_t t6 = clock();
    double timeOurQueue = double(t6 - t5) / CLOCKS_PER_SEC;

    for (int i = 0; i < numG; i++)
        visited[i] = 0;

    clock_t t7 = clock();
    cout << "\nПуть (BFS с queue собственная списки смежности):\n";
    bfsOurQueueSpisok(adj, current, visited);
    clock_t t8 = clock();
    double timeOurQueueSpisok = double(t8 - t7) / CLOCKS_PER_SEC;

    cout << "Количество вершин графа: " << numG << "\n";
    cout << "Время выполнения queue на матрице смежности: " << fixed << setprecision(6) << timeSTL << " секунд\n";
    cout << "Время выполнения queue на списках смежности: " << fixed << setprecision(6) << timeSTLSpisok << " секунд\n";
    cout << "Время выполнения наша queue на матрице смежности: " << fixed << setprecision(6) << timeOurQueue << "
секунд\n";
    cout << "Время выполнения наша queue на списках смежности: " << fixed << setprecision(6) << timeOurQueueSpisok << "
секунд\n";

    delete[] visited;
    for (int i = 0; i < numG; i++)
```

```
        delete[] G[i];
    delete[] G;

    return 0;
}

void bfsSTL(int **G, int numG, int *visited, int s) {
    queue<int> q;
    visited[s] = 1;
    q.push(s);

    while(!q.empty()) {
        int v = q.front();
        q.pop();
        cout << setw(3) << v << "\n";

        for (int i = 0; i < numG; i++) {
            if (G[v][i] == 1 && visited[i] == 0) {
                q.push(i);
                visited[i] = 1;
            }
        }
    }
}

void bfsSTLSpisok(Node **adj, int start, int *visited) {
    queue<int> q;
    visited[start] = 1;
    q.push(start);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << setw(3) << v << "\n";

        Node* cur = adj[v];
        while (cur) {
            if (!visited[cur->inf]) {
                visited[cur->inf] = 1;
                q.push(cur->inf);
            }
            cur = cur->next;
        }
    }
}
```

```
void bfsOurQueueSpisok(Node **adj, int start, int *visited) {
    Node* head = nullptr;
    Node* tail = nullptr;

    visited[start] = 1;
    enqueue(head, tail, start);

    while (!isEmpty(head)) {
        int curV = dequeue(head, tail);
        cout << setw(3) << curV << "\n";

        Node* cur = adj[curV];
        while (cur) {
            if (!visited[cur->inf]) {
                visited[cur->inf] = 1;
                enqueue(head, tail, cur->inf);
            }
            cur = cur->next;
        }
    }
}

void bfsOurQueue(int **G, int numG, int *visited, int start) {
    Node* head = nullptr;
    Node* tail = nullptr;

    visited[start] = 1;
    enqueue(head, tail, start);

    while (!isEmpty(head)) {
        int v = dequeue(head, tail);
        cout << setw(3) << v << "\n";

        for (int i = 0; i < numG; i++) {
            if (G[v][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                enqueue(head, tail, i);
            }
        }
    }
}

void enqueue(Node *& head, Node *& tail, int value) {
    Node* p = new Node{value, nullptr};
```

```
    if (!head) {
        head = tail = p;
    } else {
        tail->next = p;
        tail = p;
    }
}

int dequeue(Node *& head, Node *& tail) {
    if (!head) return -1;
    Node* temp = head;
    int val = temp->inf;
    head = head->next;
    if (!head) tail = nullptr;
    delete temp;
    return val;
}

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

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int isInteger(const string& message) {
    int value;
    while (true) {
        cout << message;
        if (!(cin >> value)) {
            cout << "Ошибка: введено не число.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
        if (cin.peek() != '\n') {
            cout << "Ошибка: введено не целое число.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
```

```
    }
    return value;
  }
}
```