МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №9

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

на тему: "Поиск расстояния в графе"

Выполнили:

Студенты группы 24ВВВ3:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Изучение алгоритма поиска расстояний в графе.

Лабораторное задание

Задание

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 C++.

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.

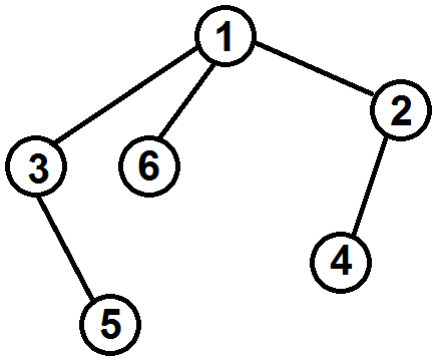
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

Пояснительный текст к программам

Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа. Для этого при каждом переходе в новую вершину необходимо запоминать, сколько шагов до нее мы сделали. При этом вектор, который хранил информацию о посещении вершин становится вектором расстояний. Довольно просто модернизировать для поиска расстояний в графе алгоритм обхода в ширину, т.к. этот алгоритм проходит вершины по уровням удаленности, то для

не ориентированного графа для вершин каждого следующего уровня глубины расстояние от исходной вершины увеличивается на 1. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины.



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

Рисунок 1 – Граф

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

Вход: G – матрица смежности графа, v – исходная вершина.

Выход: DIST – вектор расстояний до всех вершин от исходной.

Алгоритм ПОШ

1.1. для всех i положим DIST [i] = -1 пометим как "не посещенную";

1.2. **ВЫПОЛНЯТЬ** BFSD (v).

1.3 для всех i вывести DIST [i] на экран;

Алгоритм BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [x] = 0;

2.4. **ПОКА** Q != Æ 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** DIST = = -1

2.10. **ТО**

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

2.12. Обновить вектор расстояний DIST [i] = DIST [v] + 1;

Реализация состоит из подготовительной части, в которой все вершины помечаются как не посещенные (п.1.1). В отличие от алгоритма BFS не посещенные вершины помечаем -1, т.к. значение 0 и 1 могут быть расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре как и в алгоритме BFS сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0 (расстояние до самой себя).

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 для нее обновляется вектор расстояния (п.2.12). Расстояние до новой **i**-й вершины вычисляется как расстояние до текущей **v**-й вершины плюс 1 (так как ребра нашего графа не взвешенные).

После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Таким образом, если вершина помещается в очередь при просмотре сроки матрицы смежности на 1-й итерации, то они находятся на 1 уровне удаленности и расстояние до этих вершин будет равным 1.

DIST [i] = DIST [v] + 1, где DIST [v] = 0 – расстояние от исходной вершины до самой себя.

Далее, начинают просматриваться вершины первого уровня и соответствующие им строки матрицы смежности. При добавлении смежных с вершинами первого уровня вершин, расстояния до них будут равны 2.

DIST [i] = DIST [v] + 1, где DIST [v] = 1 – расстояние от исходной вершины до вершин 1 уровня.

После того, как все вершины первого уровня будут просмотрены и извлечены из очереди, начнется просмотр вершин 2 уровня и соответствующих им строк матрицы смежности. При добавлении смежных с вершинами второго уровня вершин, расстояния до них будут равны 3.

DIST [i] = DIST [v] + 1, где DIST [v] = 2 – расстояние от исходной вершины до вершин 2 уровня.

И так далее, алгоритм проходит вершины по уровням, пока очередь не опустеет.

Результаты программы

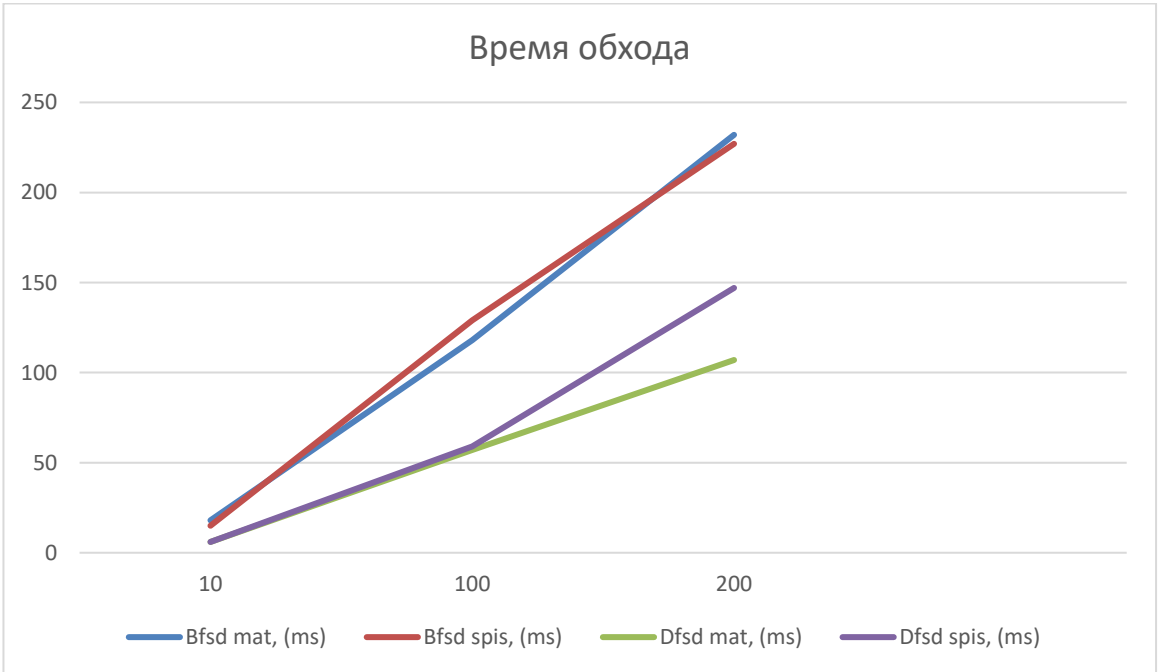
1а Рис. - Результат работы **lab9.cpp**

```
Путь (DFS списки):
  3
  2
  4
  5
  7
  1
  6
  0
  8
  9
Расстояние от вершины 3 до вершин:
  0 : 7
  1 : 5
  2 : 1
  3 : 0
  4 : 2
  5 : 3
  6 : 6
  7 : 4
  8 : 7
  9 : 7

Количество вершин графа: 10
Время выполнения bfsd на матрице смежности: 0.018000 секунд
Время выполнения bfsd на списках смежности: 0.015000 секунд
Время выполнения dfsd на матрице смежности: 0.006000 секунд
Время выполнения dfsd на списках смежности: 0.006000 секунд
```

Таблица 1. – Результаты работы **lab9.cpp**

Количество	Bfsd mat, (ms)	Bfsd spis, (ms)	Dfsd mat, (ms)	Dfsd spis, (ms)
10	**18**	**15**	**6**	**6**
100	**118**	**129**	**57**	**59**
200	**232**	**227**	**107**	**147**



Вывод: В ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №9 – поиск расстояний в графе.

Приложение А

Листинг

Файл lab9.cpp

```
// для матрицы сформировать матрицу
расстояний в обоих методах и сравнить
время формирования этих матриц
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <locale>
#include <limits>
#include <iomanip>
#include <queue>

using namespace std;

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

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

void bfsd(int** G, int numG, int* distance, int
s);
void bfsdSpisok(Node** adj, int numG, int*
distance, int s);

void dfsd(int** G, int numG, int* distance, int*
visited, int v);
void dfsdSpisok(Node** adj, int numG, int*
distance, int* visited, int v);

void bfsdD(int** G, int numG, int** GD);
void dfsdHelper(int** G, int numG, int*
```

```
distance, int* visited, int v);
void dfsdD(int** G, int numG, int** GD);

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];

    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** GD_bfs = new int* [numG];
int** GD_dfs = new int* [numG];
for (int i = 0; i < numG; i++) {
    GD_bfs[i] = new int[numG];
    GD_dfs[i] = new int[numG];
}

clock_t t_bfsD1, t_bfsD2, t_dfsD1, t_dfsD2;

t_bfsD1 = clock();
bfsdD(G, numG, GD_bfs);
t_bfsD2 = clock();
```

```
    double timeBfsdD = double(t_bfsD2 -
t_bfsD1) / CLOCKS_PER_SEC;


    t_dfsD1 = clock();
    dfsdD(G, numG, GD_dfs);
    t_dfsD2 = clock();
    double timeDfsdD = double(t_dfsD2 -
t_dfsD1) / CLOCKS_PER_SEC;


    cout << "\nМатрица расстояний (bfsdD —
BFS от всех вершин):\n";
    for (int i = 0; i < numG; i++) {
        for (int j = 0; j < numG; j++) {
            if (GD_bfs[i][j] == -1)
                cout << setw(4) << "INF";
            else
                cout << setw(4) << GD_bfs[i][j];
        }
        cout << "\n";
    }


    cout << "\nМатрица расстояний (dfsdD —
DFS от всех вершин):\n";
    for (int i = 0; i < numG; i++) {
        for (int j = 0; j < numG; j++) {
            if (GD_dfs[i][j] == -1)
                cout << setw(4) << "INF";
            else
                cout << setw(4) << GD_dfs[i][j];
        }
        cout << "\n";
    }


    cout << "\nВремя построения полной
матрицы расстояний:\n";
    cout << "BFS (матрица смежности): " <<
fixed << setprecision(6) << timeBfsdD << "
```

```
сек\n";
    cout << "DFS (матрица смежности): " <<
fixed << setprecision(6) << timeDfsdD << "
сек\n";


    int start = isInteger("\nВведите стартовую
вершину: ");
    while (start < 0 || start >= numG) {
        cout << "Ошибка! Вершина должна
быть в диапазоне [0," << numG - 1 << "]\n";
        start = isInteger("Введите вершину: ");
    }


    int* distance = new int[numG];
    int* visited = new int[numG];


    clock_t t1, t2, t3, t4, t5, t6, t7, t8;


    for (int i = 0; i < numG; i++)
        distance[i] = -1;


    t1 = clock();
    cout << "\nПуть (BFS матрица):\n";
    bfsd(G, numG, distance, start);
    t2 = clock();
    double timeBfsd = double(t2 - t1) /
CLOCKS_PER_SEC;


    for (int i = 0; i < numG; i++)
        distance[i] = -1;
    t3 = clock();
    cout << "\nПуть (BFS списки):\n";
    bfsdSpisok(adj, numG, distance, start);
    t4 = clock();
    double timeBfsdSpisok = double(t4 - t3) /
CLOCKS_PER_SEC;
```

```
    for (int i = 0; i < numG; i++) {
        distance[i] = -1;
        visited[i] = 0;
    }
    distance[start] = 0;
    t5 = clock();
    cout << "\nПуть (DFS матрица):\n";
    cout << setw(3) << start << "\n";
    dfsd(G, numG, distance, visited, start);
    t6 = clock();
    cout << "Расстояние от вершины " << start
<< " до вершин:\n";
    for (int i = 0; i < numG; i++) {
        if (distance[i] == -1) cout << setw(3) << i
<< " : изолированная вершина\n";
        else cout << setw(3) << i << " : " <<
distance[i] << "\n";
    }
    double timeDfsd = double(t6 - t5) /
CLOCKS_PER_SEC;


    for (int i = 0; i < numG; i++) {
        distance[i] = -1;
        visited[i] = 0;
    }
    distance[start] = 0;
    t7 = clock();
    cout << "\nПуть (DFS списки):\n";
    cout << setw(3) << start << "\n";
    dfsdSpisok(adj, numG, distance, visited,
start);
    t8 = clock();


    cout << "Расстояние от вершины " << start
<< " до вершин:\n";
    for (int i = 0; i < numG; i++) {
        if (distance[i] == -1) cout << setw(3) << i
```

```
<< " : изолированная вершина\n";
        else cout << setw(3) << i << " : " <<
distance[i] << "\n";
    }
    double timeDfsdSpisok = double(t8 - t7) /
CLOCKS_PER_SEC;


    cout << "\nКоличество вершин графа: " <<
numG << "\n";
    cout << "Время выполнения bfsd на
матрице смежности: " << fixed <<
setprecision(6) << timeBfsd << " секунд\n";
    cout << "Время выполнения bfsd на
списках смежности: " << fixed <<
setprecision(6) << timeBfsdSpisok << "
секунд\n";
    cout << "Время выполнения dfsd на
матрице смежности: " << fixed <<
setprecision(6) << timeDfsd << " секунд\n";
    cout << "Время выполнения dfsd на
списках смежности: " << fixed <<
setprecision(6) << timeDfsdSpisok << "
секунд\n";


    delete[] distance;
    delete[] visited;


    for (int i = 0; i < numG; i++) {
        Node* cur = adj[i];
        while (cur) {
            Node* tmp = cur;
            cur = cur->next;
            delete tmp;
        }
        delete[] G[i];
        delete[] GD_bfs[i];
        delete[] GD_dfs[i];
    }
```

```
    delete[] G;
    delete[] adj;
    delete[] GD_bfs;
    delete[] GD_dfs;

    return 0;
}

void bfsd(int** G, int numG, int* distance, int
s) {
    queue<int> q;
    distance[s] = 0;
    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 && distance[i] == -1) {
                q.push(i);
                distance[i] = distance[v] + 1;
            }
        }
    }

    cout << "Расстояние от вершины " << s <<
" до вершины:\n";
    for (int i = 0; i < numG; i++) {
        if (distance[i] == -1) cout << setw(3) << i
<< " : изолированная вершина\n";
        else cout << setw(3) << i << " : " <<
distance[i] << "\n";
    }
}
```

```
void bfsdSpisok(Node** adj, int numG, int*
distance, int s) {
    queue<int> q;
    distance[s] = 0;
    q.push(s);

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

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

    cout << "Расстояние от вершины " << s <<
" до вершины:\n";
    for (int i = 0; i < numG; i++) {
        if (distance[i] == -1) cout << setw(3) << i
<< " : изолированная вершина\n";
        else cout << setw(3) << i << " : " <<
distance[i] << "\n";
    }
}

void dfsd(int** G, int numG, int* distance, int*
visited, int v) {
    visited[v] = 1;
    for (int i = 0; i < numG; i++) {
```

```
        if (G[v][i] == 1 && !visited[i]) {
            distance[i] = distance[v] + 1;
            cout << setw(3) << i << "\n";
            dfsd(G, numG, distance, visited, i);
        }
    }
}

void dfsdSpisok(Node** adj, int numG, int*
distance, int* visited, int v) {
    visited[v] = 1;
    Node* cur = adj[v];
    while (cur) {
        int to = cur->inf;
        if (!visited[to]) {
            distance[to] = distance[v] + 1;
            cout << setw(3) << to << "\n";
            dfsdSpisok(adj, numG, distance, visited,
to);
        }
        cur = cur->next;
    }
}

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;
    }
}

void bfsdD(int** G, int numG, int** GD) {
    int* distance = new int[numG];
    queue<int> q;

    for (int s = 0; s < numG; s++) {
        for (int i = 0; i < numG; i++) {
            distance[i] = -1;
        }

        distance[s] = 0;
        q.push(s);

        while (!q.empty()) {
            int v = q.front();
```

```
            q.pop();

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

         for (int i = 0; i < numG; i++) {
            GD[s][i] = distance[i];
         }
      }

      delete[] distance;
}

void dfsdHelper(int** G, int numG, int*
distance, int* visited, int v) {
      for (int i = 0; i < numG; i++) {
         if (G[v][i] == 1 && !visited[i]) {
            visited[i] = 1;
            distance[i] = distance[v] + 1;
            dfsdHelper(G, numG, distance, visited,
i);
         }
      }
}

void dfsdD(int** G, int numG, int** GD) {
      int* distance = new int[numG];
      int* visited = new int[numG];

      for (int s = 0; s < numG; s++) {
```

```
        for (int i = 0; i < numG; i++) {
            distance[i] = -1;
            visited[i] = 0;
        }

        distance[s] = 0;
        visited[s] = 1;
        dfsdHelper(G, numG, distance, visited, s);

        for (int i = 0; i < numG; i++) {
            GD[s][i] = distance[i];
        }
    }

    delete[] distance;
    delete[] visited;
}
```