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.

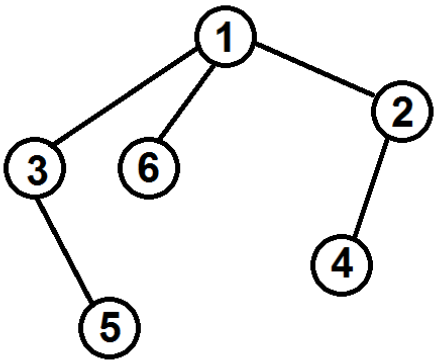
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

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.

Результат программы

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

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

```
// обход в ширину (поиск расстояний)
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <locale>
#include <limits>
#include <iomanip>
#include <queue>

using namespace std;

void clearScreen();
int isInteger(const string& message);
void bfsd(int** G, int numG, int* distance, int s);

int main() {
    setlocale(LC_ALL, "Rus");
    clearScreen();
    srand(time(NULL));
    int** G;
    int numG, current;
    int* distance;

    numG = isInteger("Введите количество вершин графа: ");
    while (numG <= 0) {
```

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

        G = (int**)malloc(sizeof(int*) * numG);
        distance = (int*)malloc(numG * sizeof(int));

        for (int i = 0; i < numG; i++) {
            G[i] = (int*)malloc(numG * sizeof(int));
        }

        for (int i = 0; i < numG; i++) {
            distance[i] = -1;
            for (int j = i; j < numG; j++) {
                G[i][j] = G[j][i] = (i == j ? 0 : rand() % 2);
            }
        }

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

        current = isInteger("Введите вершину, с которой хотите начать обход
графа: ");
        while (current < 0) {
            cout << "Ошибка! Вершина не может быть отрицательной\n";
```

```
            current = isInteger("Введите вершину, с которой хотите начать
обход графа: ");
        }

        cout << "\nПуть: \n";

        bfsd(G, numG, distance, current);

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

        return 0;
    }

    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 bfsd(int** G, int numG, int* distance, int s) {
    queue<int> q;
    int v;

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

    while(!q.empty()) {
        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;
       cout << " : изолированная вершина\n";
    }
       else {
           cout << setw(3) << i << " : " << distance[i] << "\n";
       }
    }

}
```