МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Вычислительная техника»

ОТЧЁТ

По лабораторной работе №7

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Обход графа в глубину»

Выполнили
студенты
группы
23ВВВ4:
Святов И.Ю.
Епинин Д.В.

Приняли:
Юрова О.В.
Деев М.В.

Пенза 2024

Общие сведения:
Обход графа – одна из наиболее распространенных операций с графами. Задачей обхода является прохождение всех вершин в графе. Обходы применяются для поиска информации, хранящейся в узлах графа, нахождения связей между вершинами или группами вершин и т.д.
Одним из способов обхода графов является поиск в глубину. Идея такого обхода состоит в том, чтобы начав обход из какой-либо вершины всегда переходить по первой встречающейся в процессе обхода связи в следующую вершину, пока существует такая возможность. Как только в процессе обхода исчерпаются возможности прохода, необходимо вернуться на один шаг назад и найти следующий вариант продвижения. Таким образом, итерационно выполняя описанные операции, будут пройдены все доступные для прохождения вершины. Чтобы не заходить повторно в уже пройденные вершины, необходимо их пометить как пройденные.
Таким образом, можно предложить следующую рекурсивную реализацию алгоритма обхода в глубину.

Задание:

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    const int G = 5;  // Размер графа
    int M[G][G];      // Матрица смежности

    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет ребер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
```

```
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

2. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>
const int G = 5;
int M[G][G];
int visited[G];
// Функция для выполнения обхода в глубину
void DFS(int v) {
    visited[v] = 1;  // Помечаем текущую вершину как посещённую
    printf("%d ", v);

    // Обходим всех соседей текущей вершины
    for (int i = 0; i < G; i++) {
        // Если есть ребро и вершина не посещена
        if (M[v][i] == 1 && !visited[i]) {
            DFS(i); // Рекурсивно вызываем DFS для соседней вершины
        }
    }
}

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

    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }

    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }

    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
```

```
            for (int j = 0; j < G; j++) {
                printf("%d ", M[i][j]);
            }
            printf("\n");
        }

        // Инициализация массива посещённых вершин
        for (int i = 0; i < G; i++) {
            visited[i] = 0;
        }

        printf("Обход в глубину (DFS):\n");
        DFS(0); // Начинаем с вершины 0

        return 0;
    }
```

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

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>
const int G = 5;
typedef struct Node {
    int vertex;
    struct Node* next;
} Node;
Node* adjList[G]; // Массив указателей для хранения списков смежности
int visited[G]; // Массив для отслеживания посещённых вершин

// Функция для добавления ребра в список смежности
void addEdge(int src, int dest) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->vertex = dest;
    newNode->next = adjList[src]; // Добавляем узел в начало списка
    adjList[src] = newNode;

    // Поскольку граф неориентированный, добавляем рёбра в обе стороны
    newNode = (Node*)malloc(sizeof(Node));
    newNode->vertex = src; // добавляем обратное ребро
    newNode->next = adjList[dest]; // Добавляем узел в начало списка
    adjList[dest] = newNode;
}

// Функция для выполнения обхода в глубину
void DFS(int v) {
    visited[v] = 1;  // Помечаем текущую вершину как посещённую
    printf("%d ", v); // Выводим текущую вершину

    // Обходим всех соседей текущей вершины
    Node* temp = adjList[v];
```

```
    while (temp) {
        if (!visited[temp->vertex]) {
            DFS(temp->vertex); // Рекурсивно вызываем DFS для соседней вершины
        }
        temp = temp->next;
    }
}
// Функция для освобождения памяти, выделенной под список смежности
void freeGraph() {
    for (int i = 0; i < G; i++) {
        Node* temp = adjList[i];
        while (temp) {
            Node* toFree = temp;
            temp = temp->next;
            free(toFree); // Освобождаем память для узла
        }
    }
}
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));

    // Инициализация списков смежности
    for (int i = 0; i < G; i++) {
        adjList[i] = NULL; // Изначально пусто
        visited[i] = 0; // Все вершины не посещены
    }

    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            if (rand() % 2) { // С вероятностью 0.5 добавляем ребро
                addEdge(i, j); // Добавляем ребро в граф
            }
        }
    }
    // Вывод списков смежности на экран
    printf("Списки смежности:\n");
    for (int i = 0; i < G; i++) {
        printf("%d:", i);
        Node* temp = adjList[i];
        while (temp) {
            printf(" %d", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }

    // Инициализация массива посещённых вершин
    for (int i = 0; i < G; i++) {
        visited[i] = 0;
    }
    // Запуск обхода в глубину начиная с первой вершины (0)
    printf("Обход в глубину (DFS):\n");
    DFS(0); // Начинаем с вершины 0
    // Освобождаем выделенную память
    freeGraph();
```

```
    return 0;
}
```

Задание 2*

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

const int G = 5; // Размер графа
int M[G][G]; // Матрица смежности
int visited[G]; // Массив для отслеживания посещённых вершин
// Функция для выхода из DFS (не рекурсивная реализация)
void DFS_Iterative(int start) {
    int stack[G]; // Стек для итеративного DFS
    int top = -1; // Указатель на верхушку стека

    // Добавляем начальную вершину в стек и отмечаем её как посещённую
    stack[++top] = start;
    visited[start] = 1;
    printf("%d ", start); // Выводим начальную вершину

    while (top != -1) { // Пока стек не пуст
        int v = stack[top]; // Берём вершину с вершины стека
        int found = 0; // Флаг для проверки не посещённых соседей

        for (int i = 0; i < G; i++) {
            // Ищем непосещённого соседа
            if (M[v][i] == 1 && !visited[i]) {
                stack[++top] = i; // Добавляем соседа в стек
                visited[i] = 1; // Отмечаем соседа как посещённого
                printf("%d ", i); // Выводим соседа
                found = 1; // Отмечаем, что нашли непосещённого
                break; // Прерываем цикл, чтобы начать обход с нового узла
            }
        }

        if (!found) {
            top--; // Если не нашли соседей, убираем вершину из стека
        }
    }
}
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0; // Нет петли
```

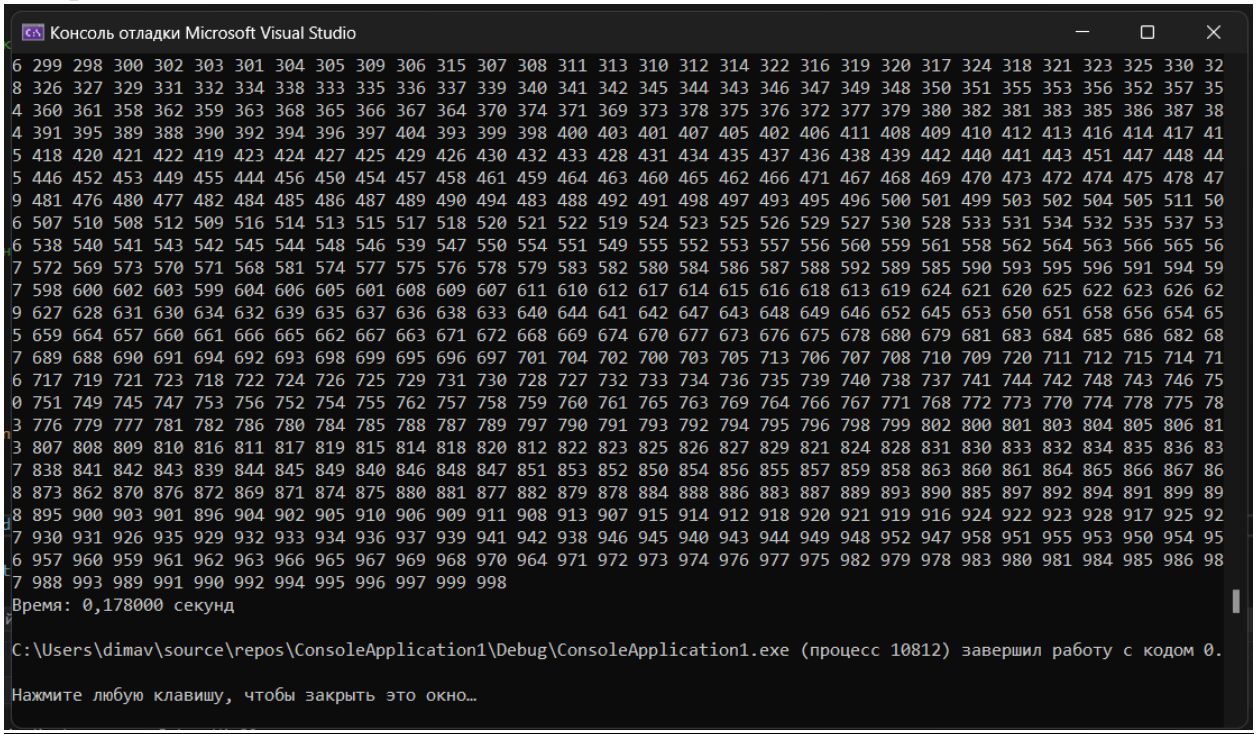
```
            }
            else {
                M[i][j] = 0; // Изначально нет рёбер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2; // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    // Инициализация массива посещённых вершин
    for (int i = 0; i < G; i++) {
        visited[i] = 0;
    }
    // Запуск итеративного обхода в глубину начиная с первой вершины (0)
    printf("Итеративный обход в глубину (DFS):\n");
    DFS_Iterative(0); // Начинаем с вершины 0

    return 0;
}
```

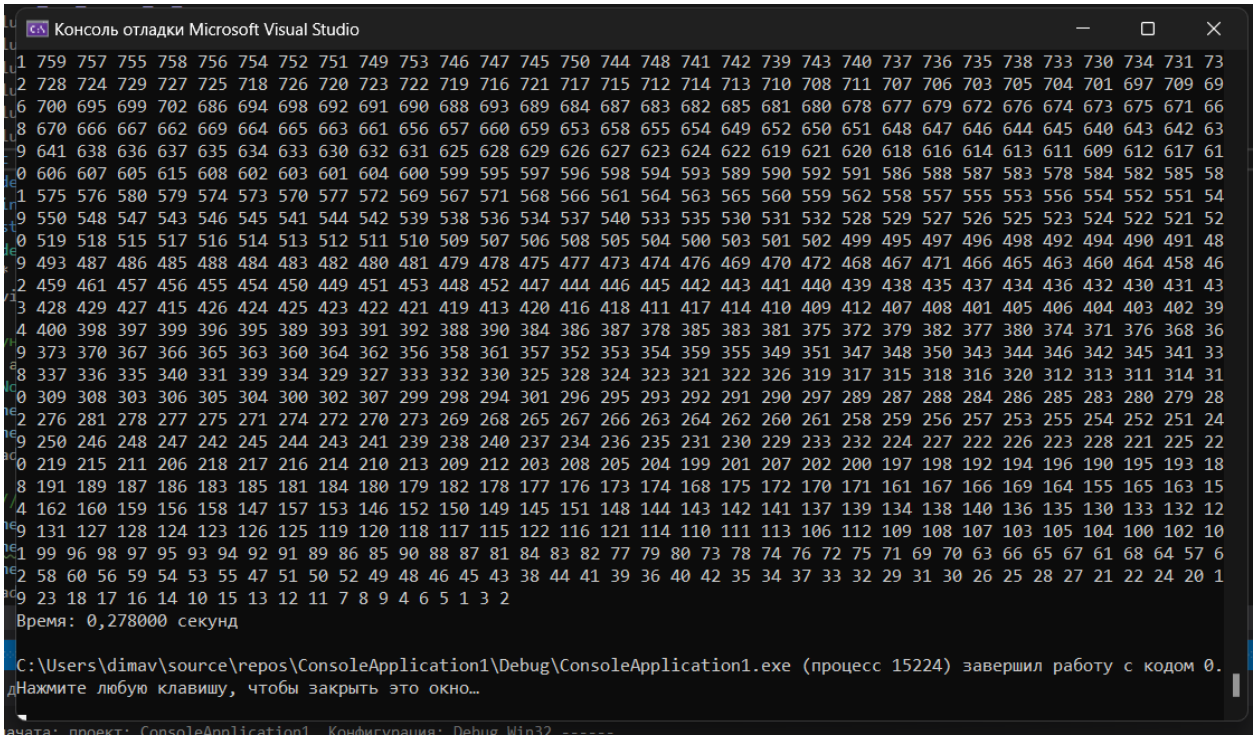
Задание: проверить за сколько выполняется код

Ответ:

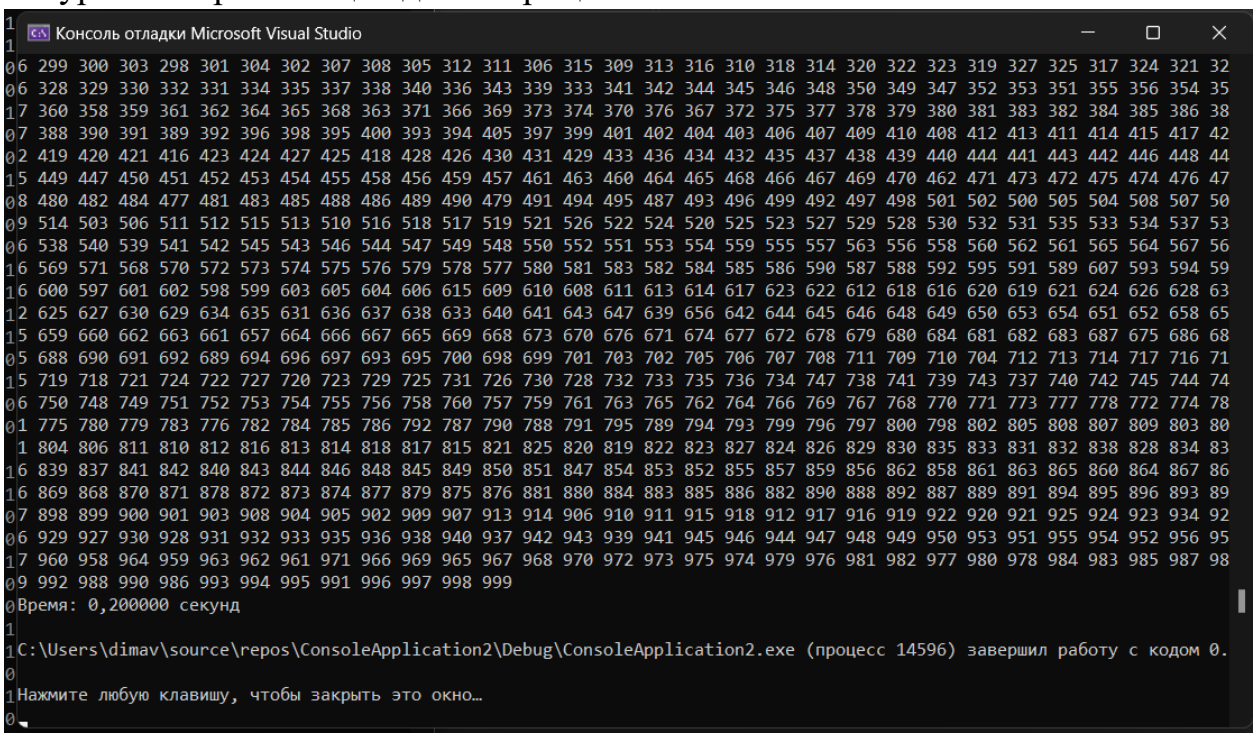
Матрица смежности:



Списки смежности:



Рекурсивная реализация для матрицы смежности:



Вывод: в ходе выполнения лабораторной работы были успешно сгенерированы случайные графы в матричной и списочной форме. Также были реализованы процедуры обхода графа в глубину как в рекурсивной, так и в не рекурсивной формах. Работы демонстрируют различные подходы к представлению графов и алгоритмам обхода.