МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №7  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Обход графа в глубину»

Выполнил:

Студент группы 23ВВВ2

Чупраков С. В.

Пичаев И. А.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Приобрести навыки программирования и работы алгоритмов обхода графа в глубину различными способами.

**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 глубину, реализованную в соответствии с приведенным выше описанием.

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

### **Задание 2**

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

**Ход работы**

### **Задание 1.1-2**

Матрица смежности генерируется пи помощи функции rand(). Затем вызывается функция обхода в глубину. Вершина отмечается, как посещённая, с помощью массива NUM, размером с количество вершин. Изначально все вершины в массиве отмечены 0, как не посещённые. Если перешли, то меняем значение на 1 и выводим номер вершины в консоли. В цикле ищется следующая вершина для перехода. Для этого должно быть соблюдено два условия: наличие ребра между вершинами и новая вершина не должна быть посещена ранее. Если условия соблюдены рекурсивно вызываем функцию.

### **Задание 1.3**

Генерируется граф сразу записывается в виде списка смежности. Если между вершинами есть ребро, то для обоих добавляется вершина в список. Далее список смежных вершин сортируется с помощью функции qsort() для более понятного вывода списка смежности и корректной работы алгоритма сортировки. Функция обхода аналогичная как для матрицы смежности. Было изменено только условие перехода к новой вершине. Так как в списке смежности для каждой вершины записаны только те вершины с которыми у нее есть ребро, необходима только проверка на наличие в списке посещённых вершин. Если условие выполнено, функция рекурсивно вызывается заново.

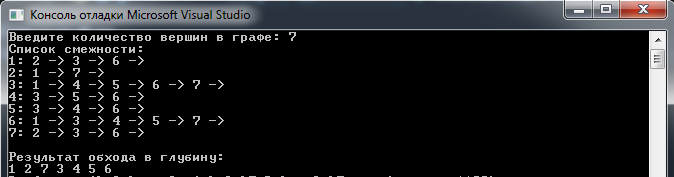
### **Задание 2**

Для реализации обхода в глубину без использования рекурсии воспользуемся стеком. В функции обхода инициализируем пустой стек. Затем записывается вершина, с которой начинается обход. Далее в цикле мы извлекаем из стека верхнюю вершину, и проверяем посещена ли она, если нет, то выводим в консоли и отмечаем, как посещенную. Далее в стек записываются все смежные вершины. Запись идет от большего к меньшему. Далее цикл повторяется пока стек не опустеет.

### **Результаты работы программы**

### 

Рисунок 1 — Результаты работы программы 1.2

 Рисунок 2 — Результаты работы программы 1.3

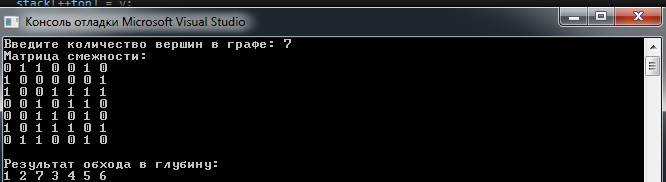


Рисунок 1 — Результаты работы программы 2

**Вывод**

Приобретены навыки программирования и работы алгоритмов обхода графа в глубину различными способами.

**Листинг**

### **Задание 1.(1-2)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include <locale.h>

void createG(int\*\* G, int N) {

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

for (int j = i + 1; j < N; j++) {

int edge = rand() % 2;

G[i][j] = edge;

G[j][i] = edge;

}

}

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

G[i][i] = 0;

}

}

void printG(int\*\* G, int N) {

printf("Матрица смежности:\n");

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

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

printf("%d ", G[i][j]);

}

printf("\n");

}

}

void DFS(int\*\* G, int N, int\* NUM, int v) {

NUM[v] = 1;

printf("%d ", v+1);

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

if (G[v][i] == 1 && NUM[i] == 0) {

DFS(G, N, NUM, i);

}

}

}

int main() {

setlocale(LC\_ALL, "");

int N;

printf("Введите количество вершин в графе: ");

scanf("%d", &N);

int\*\* G = (int\*\*)malloc(N \* sizeof(int\*));

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

G[i] = (int\*)malloc(N \* sizeof(int));

}

createG(G, N);

printG(G, N);

int\* NUM = (int\*)malloc(N \* sizeof(int));

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

NUM[i] = 0;

}

printf("\nРезультат обхода в глубину:\n");

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

if (NUM[i] == 0) {

DFS(G, N, NUM, i);

}

}

}

**Задание 1.3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

void createAdjacencyList(Node\*\* adjList, int N) {

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

for (int j = i + 1; j < N; j++) {

int edge = rand() % 2;

if (edge == 1) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->vertex = j;

newNode->next = adjList[i];

adjList[i] = newNode;

newNode = (Node\*)malloc(sizeof(Node));

newNode->vertex = i;

newNode->next = adjList[j];

adjList[j] = newNode;

}

}

}

}

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

void sortAdjacencyList(Node\*\* adjList, int N) {

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

int count = 0;

Node\* temp = adjList[i];

while (temp != NULL) {

count++;

temp = temp->next;

}

int\* vertices = (int\*)malloc(count \* sizeof(int));

temp = adjList[i];

int index = 0;

while (temp != NULL) {

vertices[index++] = temp->vertex;

temp = temp->next;

}

qsort(vertices, count, sizeof(int), compare);

temp = adjList[i];

index = 0;

while (temp != NULL) {

temp->vertex = vertices[index++];

temp = temp->next;

}

free(vertices);

}

}

void printAdjacencyList(Node\*\* adjList, int N) {

printf("Список смежности:\n");

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

printf("%d: ", i + 1);

Node\* temp = adjList[i];

while (temp != NULL) {

printf("%d -> ", temp->vertex + 1);

temp = temp->next;

}

printf("\n");

}

}

void DFS(Node\*\* adjList, int N, int\* NUM, int v) {

NUM[v] = 1;

printf("%d ", v + 1);

Node\* temp = adjList[v];

while (temp != NULL) {

if (NUM[temp->vertex] == 0) {

DFS(adjList, N, NUM, temp->vertex);

}

temp = temp->next;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int N;

printf("Введите количество вершин в графе: ");

scanf("%d", &N);

Node\*\* adjList = (Node\*\*)malloc(N \* sizeof(Node\*));

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

adjList[i] = NULL;

}

createAdjacencyList(adjList, N);

sortAdjacencyList(adjList, N);

printAdjacencyList(adjList, N);

int\* NUM = (int\*)malloc(N \* sizeof(int));

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

NUM[i] = 0;

}

printf("\nРезультат обхода в глубину:\n");

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

if (NUM[i] == 0) {

DFS(adjList, N, NUM, i);

}

}

}

**Задание 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include <locale.h>

void createG(int\*\* G, int N) {

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

for (int j = i + 1; j < N; j++) {

int edge = rand() % 2;

G[i][j] = edge;

G[j][i] = edge;

}

}

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

G[i][i] = 0;

}

}

void printG(int\*\* G, int N) {

printf("Матрица смежности:\n");

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

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

printf("%d ", G[i][j]);

}

printf("\n");

}

}

void DFS(int\*\* G, int N, int\* NUM, int v) {

int\* stack = (int\*)malloc(N \* sizeof(int));

int top = -1;

stack[++top] = v;

while (top != -1) {

int vec = stack[top--];

if (NUM[vec] == 0) {

NUM[vec] = 1;

printf("%d ", vec + 1);

}

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

if (G[vec][i] == 1 && NUM[i] == 0) {

stack[++top] = i;

}

}

}

}

int main() {

setlocale(LC\_ALL, "");

int N;

printf("Введите количество вершин в графе: ");

scanf("%d", &N);

int\*\* G = (int\*\*)malloc(N \* sizeof(int\*));

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

G[i] = (int\*)malloc(N \* sizeof(int));

}

createG(G, N);

printG(G, N);

int\* NUM = (int\*)malloc(N \* sizeof(int));

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

NUM[i] = 0;

}

printf("\nРезультат обхода в глубину:\n");

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

if (NUM[i] == 0) {

DFS(G, N, NUM, i);

}

}

}