Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Выполнил:**

Студенты группы 19ВВ2:

Юдин Д.

Анохин А.

Вожжов И.

**Приняли:**

Митрохин М.А.

Юрова О.В.

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

Пенза 2020

### Название:

### Унарные и бинарные операции над графами.

### Задание:

### Задание 1

1. Сгенерируйте (используя генератор случайных чисел) две матрицу смежности для неориентированного графа *G*. Выведите сгенерированные матрицы 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.

### Цель работы:

Разработать программу по данным заданиям.

**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.

**Вход**: G – матрица смежности графа.

**Выход**: номера вершин в порядке их прохождения на экране.

**Алгоритм ПОГ**

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

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

1.3. **ВЫПОЛНЯТЬ** DFS (v).

**Алгоритм** DFS(v):

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

2.2. вывести на экран v;

2.3. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

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

2.5. **ТО**

2.6. {

2.7. DFS(i);

2.8. }

Реализация состоит из подготовительной части, в которой все вершины помечаются как не помеченные (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3). И непосредственно процедуры обхода, которая помечает текущую (т.е. ту, в которой на текущей итерации находится алгоритм) вершину как посещенную (п. 2.1). Затем выводит номер текущей вершины на экран (п.2.2) и в цикле просматривает **v**-ю строку матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.4), то для этой вершины вызывается процедура обхода (п.2.7).

Например, пусть дан граф (рисунок 1), заданный в виде матрицы смежности:



Рисунок 1 – Граф

Тогда, если мы начнем обход из первой вершины, то на шаге 2.1 она будет помечена как посещенная (NUM[1] = True), на экран будет выведена единица.

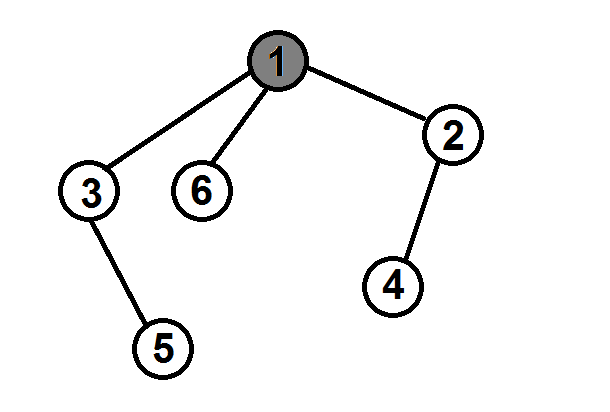
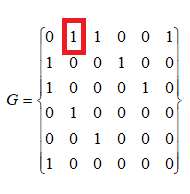
 



Рисунок 2 – Вызов DFS(1)

При просмотре 1-й строки матрицы смежности



будет найдена смежная вершина с индексом 2 (G(1,2) = =1), которая не посещена (NUM[2] = = False) и будет вызвана процедура обхода уже для нее - DFS(2).

На следующем вызове на шаге 2.1 вершина 2 будет помечена как посещенная (NUM[2] = True), на экран будет выведена двойка.

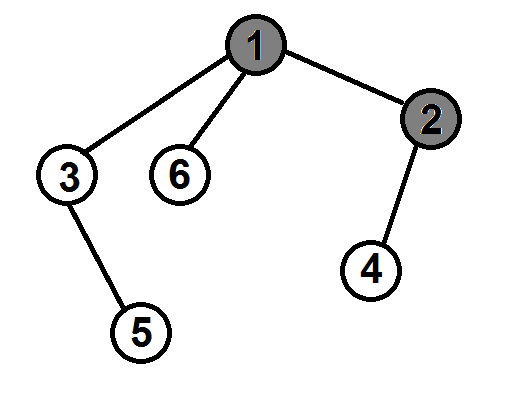
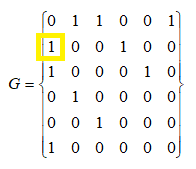




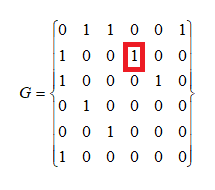
Рисунок 3 – Вызов DFS(2)

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



которая к настоящему моменту уже посещена (NUM[1] = = True) и процедура обхода для нее вызвана не будет. Цикл 2.3 продолжит просмотр матрицы смежности.

Следующая найденная вершина, смежная со второй, будет иметь индекс 4 (G(2,4) = =1), она не посещена (NUM[4] = = False) и для нее будет вызвана процедура обхода - DFS(4).



Вершина 4 будет помечена как посещенная (NUM[4] = True), на экран будет выведена четверка.

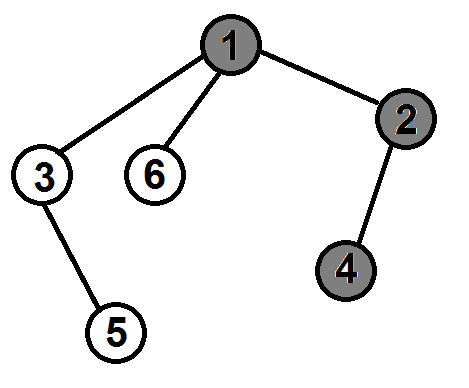
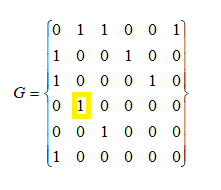




Рисунок 4 – Вызов DFS(4)

При просмотре 4-й строки матрицы будет найдена вершина 2, но она уже посещена (NUM[2] = True), поэтому процедура обхода вызвана не будет.



Цикл 2.3 завершится и для текущего вызова DFS(4) процедура закончит свою работу, вернувшись к точке вызова, т.е. к моменту просмотра циклом 2.3 строки с индексом 2 для вызова DFS(2).

В вызове DFS(2) цикл 2.3 продолжит просмотр строки 2 в матрице смежности, и, пройдя её до конца завершится. Вместе с этим завершится и вызов процедуры DFS(2), вернувшись к точке вызова - просмотру циклом 2.3 строки с индексом 1 для вызова DFS(1).

При просмотре строки 1 циклом 2.3 в матрице смежности будет найдена следующая не посещенная, смежная с 1-й, вершина с индексом 3 (G(1,2) = =1 и NUM[3] = = False) и для нее будет вызвана DFS(3).

Вершина 3 будет помечена как посещенная (NUM[3] = True), на экран будет выведена тройка.

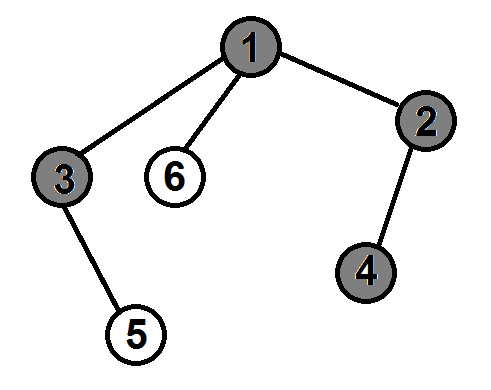




Рисунок 4 – Вызов DFS(3)

Работа алгоритма будет продолжаться до тех пор, пока будут оставаться не посещенные вершины, т.е. для которых NUM[i] = = False.

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





Рисунок 5 – Результат работы обхода

### Листинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

#include <stack>

using namespace std;

typedef struct Node {

int vertex;

struct Node \*next;

} Node;

void add\_Node\_to\_beginning(Node \*\*head, int data) //функция добавления узла в начало

{

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = (\*head); //присваиваем указателю tmp адрес след. узла

(\*head) = tmp\_node; //Присваиваем указателю head адрес tmp

} //после выхода из функции tmp уничтожается

Node\* findLastHead(Node\* head) { //поиск адреса последнего элемента

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void add\_Node\_to\_end(Node\* head, int data) { //функция добавления нового узла в конец

Node\* last = findLastHead(head); //получаем указатель на последний элемент списка

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = NULL;

last->next = tmp\_node; //записываем в последний элемент списка указатель на новый узел

}

void createLinkedList(int\*\* a, Node\*\* head, int n) //создаем связный список

{

for (int i = 0; i < n; i++)

{

add\_Node\_to\_beginning(&head[i], i); //передаем адрес вершины и ее номер

for (int j = 0; j < n; j++)

{

if (a[i][j] == 1)

{

add\_Node\_to\_end(head[i], j); //добавляем к вершине связные с ней вершины

}

}

}

}

void printLinkedList(const Node\* head) {

printf("\n");

while (head) {

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

head = head->next;

}

printf("\n");

}

void DFS(int\*\* a, bool\* visited, int n, int v) //поиск в глубину, v - текущая вершина, n - кол-во вершин, a - массив

{

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

visited[v] = true; //помещаем текущюю вершину

for (int i = 0; i < n; i++)

{

if (a[v][i] == 1 && visited[i] == false) //проверяем смежные с ней вершины

{

DFS(a, visited, n, i); //вызываем рекурсию

}

}

}

void DFSlist(Node\*\* head, bool\* visited, int v)

{

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

visited[v] = true;

Node\* tmp\_node = head[v];

while (tmp\_node) //идем по текущей вершине

{

if (visited[tmp\_node->vertex] == false)

DFSlist(head, visited, tmp\_node->vertex); //если вершина не посещена то передаем ее адрес и номер и идем дальше по ней

tmp\_node = tmp\_node->next;

}

}

void DFS\_no\_rec(int\*\* a, bool\* visited, int n, int v){ //обход графа без рекурсии

stack <int> stack;

stack.push(v);

while (!stack.empty())

{

if (visited[stack.top()] == false)

{

visited[stack.top()] = true;

printf("%d ", stack.top() + 1);

v = stack.top();

stack.pop();

for (int i = n - 1; i > 0; i--)

{

if (a[v][i] == 1 && visited[i] == false)

{

stack.push(i);

}

}

}

else

{

stack.pop();

}

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* a, N, num; //N-кол-во вершин, num-вершина для ввода

printf("\n Укажите размер матрицы N\*N: ");

scanf("%d", &N);

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

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

head[i] = NULL;

}

bool\* visited = (bool\*)malloc(N \* sizeof(bool));

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

{

visited[i] = false;

}

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

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

{

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

}

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

{

a[i][i] = 0;

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

{

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

}

printf("\n Вывод матрицы смежности\n ");

printf("\n ");

printf("\n ");

printf("\t");

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

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

}

for (int i = 0; i < N; i++) { // вывод матрицы смежности

printf("\n");

printf("%d\t", i+1);

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

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

}

}

printf("\n\nВведите вершину для начала обхода графа в глубину : ");

scanf("%d", &num);

num--;

printf("\n\nРекурсивный проход в глубину по матрице смежности : ");

DFS(a, visited, N, num);

printf("\n\n");

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

visited[i] = false;

createLinkedList(a, head, N);

printf("\n\nСоздадим список смежности: ");

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

printLinkedList(head[i]);

printf("\n\n");

printf("\n\nРекурсивный проход в глубину по списку смежности : ");

DFSlist(head, visited, num);

printf("\n\n");

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

visited[i] = false;

}

printf("\n\nПроход в глубину по матрице смежности без рекурсивной функции: ");

DFS\_no\_rec(a, visited, N, num);

printf("\n\n");

\_getch;

}

### Результаты работы программы:

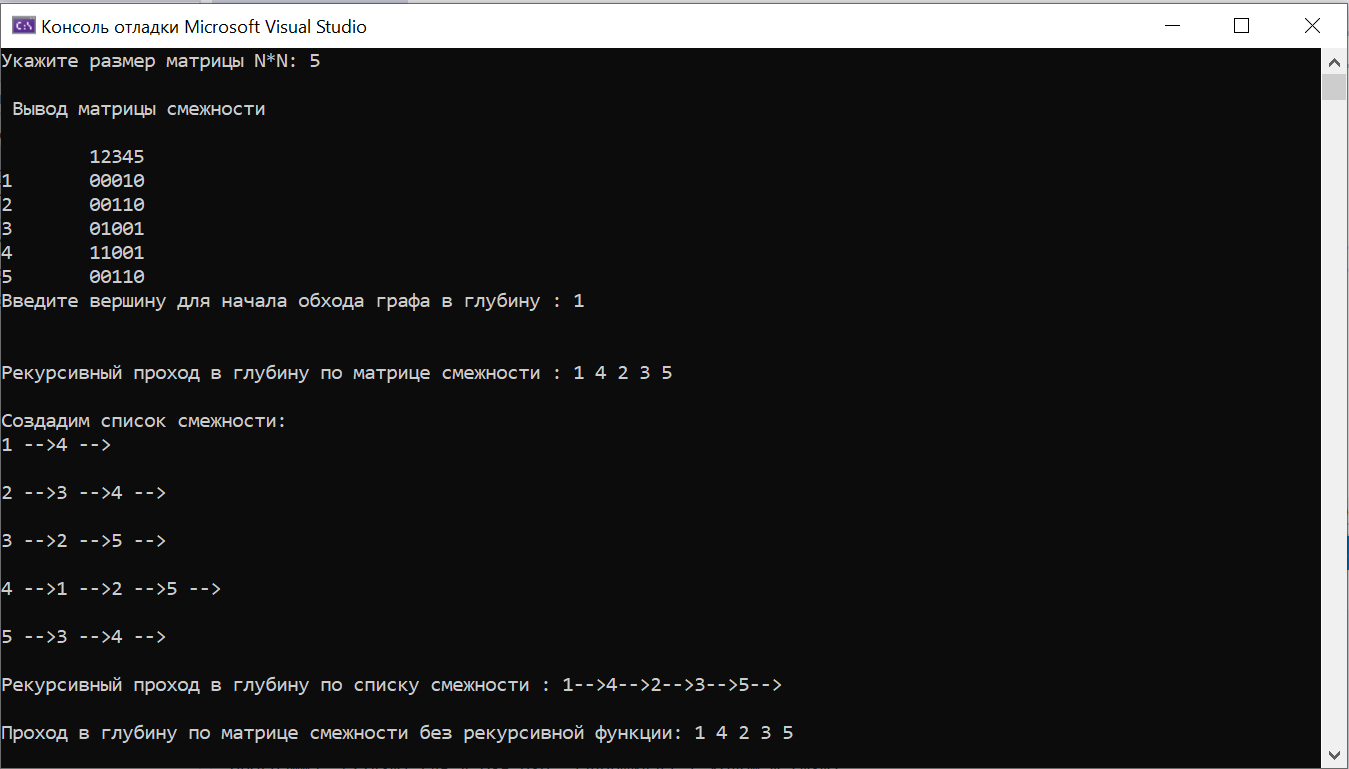


Рисунок 1 Результат работы программы

### Вывод:

В ходе выполнения лабораторной работы мы научились реализовывать обход в глубину с помощью рекурсивной функции для матрицы смежности, списка смежности, а также с помощью библиотеки stack для матрицы смежности. Были написаны функции и разработана программа для соответствующего задания.