Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Выполнил:**

Студенты группы 19ВВ2:

Ильин С.

Пронин В.

Дерябина А.

**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 программы 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.

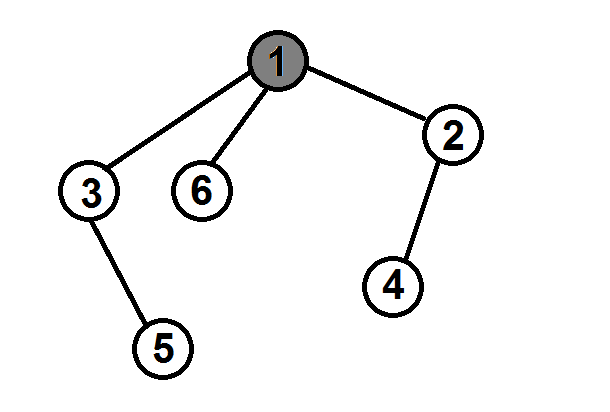
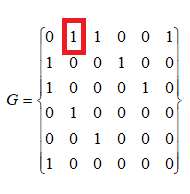
 



Рисунок 2 – Вызов DFS(1)

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.

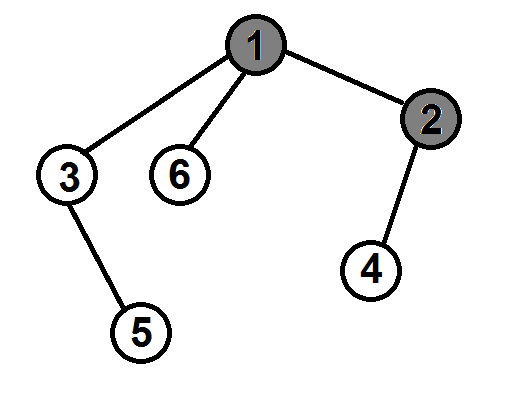
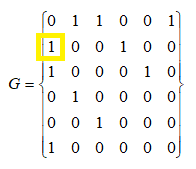




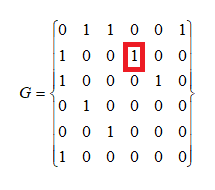
Рисунок 3 – Вызов DFS(2)

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.

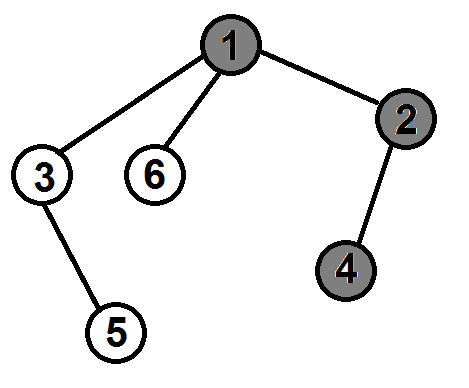
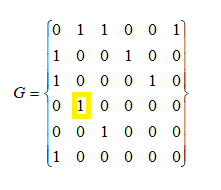




Рисунок 4 – Вызов DFS(4)

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.

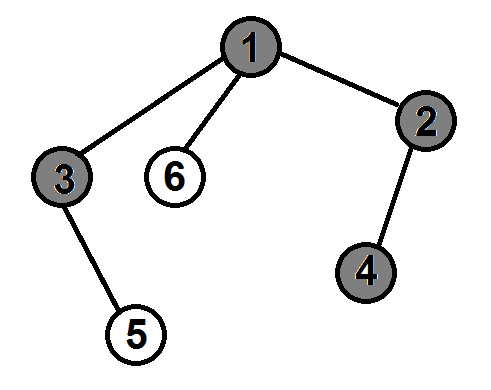




Рисунок 4 – Вызов DFS(3)

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

### Описание метода решения задачи:

### Задание 1

Сгенерировали (используя генератор случайных чисел) две матрицу смежности для неориентированного графа *G*. Вывели сгенерированные матрицы на экран. Для сгенерированного графа осуществили процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием. Реализовали процедуру обхода в глубину для графа, представленного списками смежности.

### Задание 2

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

### Листинг:

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <stack>

#include <iostream>

#include <iomanip>

using namespace std;

typedef struct Node {

int ver;

struct Node\* next;

} Node;

void push(Node\*\* head, char data)

{

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

tmp->ver = data;

tmp->next = (\*head);

(\*head) = tmp;

}

Node\* getLast(Node\* head) {

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void pushBack(Node\* head, char value) {

Node\* last = getLast(head);

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

tmp->ver = value;

tmp->next = NULL;

last->next = tmp;

}

void initializinglist(int\*\* a, Node\*\* head, int n)

{

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

{

push(&head[i], i);

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

{

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

{

pushBack(head[i], j);

}

}

}

}

void printLinkedList(const Node\* head) {

if (head) {

cout<< head->ver + 1;

head = head->next;

}

while (head) {

cout << "--->" << head->ver + 1;

head = head->next;

}

printf("\n");

}

void DFS(int\*\* a, bool\* visited, int n, int v)

{

cout << v + 1;

visited[v] = true;

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

{

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

{

DFS(a, visited, n, i);

}

}

}

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

{

cout << v + 1;

visited[v] = true;

Node\* tmp = head[v];

while (tmp)

{

if (visited[tmp->ver] == false)

DFSlist(head, visited, tmp->ver);

tmp = tmp->next;

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

stack <int> steck;

int\*\* a, n = 7, num, v;

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++)

{

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

{

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

}

}

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

{

cout << "\n\n";

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

{

cout << a[i][j]<<setw(5);

}

}

cout << "\n\nВыберите вершину для начала обхода в глубину: ";

cin >> num;

if (num > n || num < 1)

{

cout << "\aОшибка ввода, попробуйте снова\n\nВыберите вершину для начала обхода в глубину: ";

cin >> num;

}

num--;

cout << "Проход в глубину с помощью рекурсивной функции для графа представленного матрицей смежности:\n\n";

DFS(a, visited, n, num);

cout << "\n\n";

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

visited[i] = false;

cout << "Проход в глубину с помощью рекурсивной функции для графа представленного списками смежности:\n\n";

initializinglist(a, head, n);

cout << "Список смежности для матрицы А:\n\n";

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

printLinkedList(head[i]);

cout << "\n\n";

DFSlist(head, visited, num);

cout << "\n\n";

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

visited[i] = false;

cout << "Проход в глубину с помощью не рекурсивной функции для графа представленного матрицей смежности:\n\n";

steck.push(num);

while (!steck.empty())

{

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

{

visited[steck.top()] = true;

cout << steck.top() + 1;

v = steck.top();

steck.pop();

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

{

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

{

steck.push(i);

}

}

}

else

{

steck.pop();

}

}

cout << "\n\n";

system("pause");

}

### Результаты работы программы:

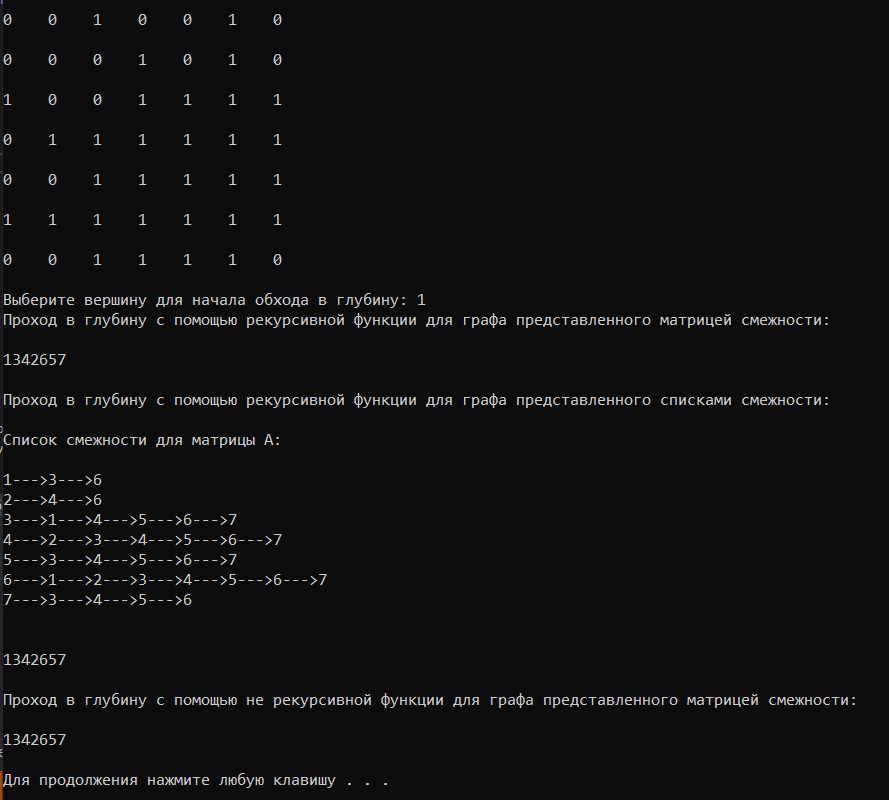


Рисунок Результат работы программы

### Вывод:

В ходе выполнения лабораторной работы мы научились реализовывать обход в глубину с помощью рекурсивных и не рекурсивных функций. Были разработаны программы, соответствующие данным заданиям.