Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

Выполнили

студент группы 21ВВ3:

Савицкий Макар

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Цель работы**

Реализация рекурсивного и не рекурсивного обход графа в глубину для списков и матриц списков смежности.

**Лабораторное 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\***

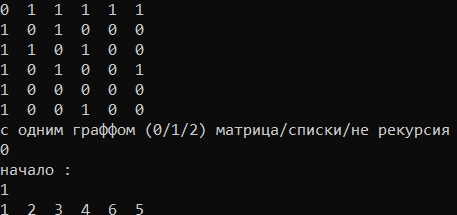
1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Описание метода решения задачи**

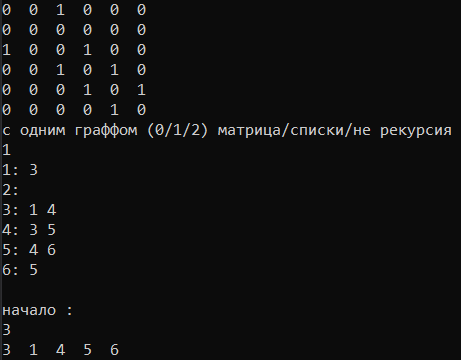
1. *Генерируем матрицу смежности*
2. *В зависимости от введенного значения выполняем соответствующую операцию*
3. *Выбираем с какой вершины начать обход*
4. *В зависимости от введённого значения выполняем ту или иную команду*
5. *Выводим очередность вершин, в порядке обхода*

**Результаты работ программы**

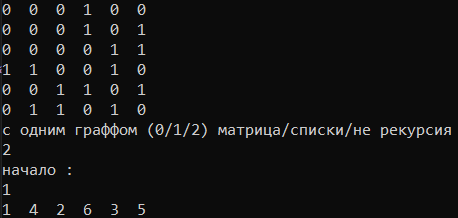
**Рекурсивный обход графа в глубину для матрицы**

****

**Рекурсивный обход графа в глубину для списка**

****

**Не рекурсивный обход графа в глубину для матрицы**

****

**Листинг**#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //указание прототипов

#include <stdlib.h> //библиотечных функций

#include <windows.h>

#include <malloc.h>

#include <time.h>

struct node{

int inf;

struct node\* next;

};

struct node\* head = NULL, \* last = NULL;

void push(int t) {

struct node\* p = NULL, \* f = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL){

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->inf = t;

p->next = NULL;

if (p != NULL) {

if (head == NULL) {

head = p;

last = p;

}

else{

p->next = head;

head = p;

}

}

return;

}

int pop(){

struct node\* struc = head;

int t;

if (head == NULL)

return -1;

t = struc->inf;

head = struc->next;

free(struc);

return t;

}

struct list {

int i;

struct list\* next;

};

struct branch {

int t;

struct list\* head;

struct list\* last;

struct branch\* next;

};

struct branch\* head\_b = NULL, \* last1 = NULL;

void spstore\_branch(int t) {

struct branch\* p = NULL;

if ((p = (branch\*)malloc(sizeof(struct branch))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->t = t;

p->head = NULL;

p->last = NULL;

p->next = NULL;

if (p != NULL) {

if (head\_b == NULL) {

head\_b = p;

last1 = p;

}

else {

last1->next = p;

last1 = p;

}

}

}

void spstore\_list(int i) {

struct branch\* struc1 = last1; // указатель, проходящий по списку установлен на начало списка

struct branch\* prev1 = NULL;

if (head\_b == NULL) {

return;

}

struct list\* p = NULL, \* f = NULL;

if ((p = (list\*)malloc(sizeof(struct list))) == NULL){ // выделяем память под новый элемент списка

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->i = i;

p->next = NULL;

if (struc1->head == NULL && p != NULL) { // если списка нет, то устанавливаем голову списка

struc1->head = p;

struc1->last = p;

}

else if (struc1->head != NULL && p != NULL) {

f = struc1->head;

int ii = 0;

if (f->i > p->i) { //если в начало ставим

p->next = struc1->head;

struc1->head = p;

ii = 1;

}

if (struc1->last->i < p->i) { //если в конец ставим

struc1->last->next = p;

struc1->last = p;

ii = 1;

}

if (struc1->last->i > p->i && f->i < p->i) {

while (f->next->i < p->i) {

f = f->next;

}

p->next = f->next;

f->next = p;

ii = 1;

}

if (ii == 0) {

free(p);

}

}

}

void review\_uz() {

struct branch\* struc = head\_b;

struct list\* struc1;

while (struc) {

printf("%d: ", struc->t + 1);

struc1 = struc->head;

while (struc1) {

printf("%-2d", struc1->i);

struc1 = struc1->next;

}

printf("\n");

struc = struc->next;

}

}

void matrixs(int \*G ,int \*NUM, int size\_G, int v){

\*(NUM + v) = 1;

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

for (int i = 0; i < size\_G; i++) {

if (\*(G + i \* size\_G + v) == 1 && \*(NUM + i) == 0)

matrixs(G, NUM, size\_G, i);

}

}

void lists(int\* NUM, int size\_G, int v) {

\*(NUM + v - 1) = 1;

printf("%d ", v);

struct branch\* b = head\_b;

for(int i = 0; i < v-1; i++)

b = b->next;

struct list\* l = b->head;

while(l){

if (\*(NUM + l->i-1) == 0)

lists(NUM, size\_G, l->i);

l = l->next;

}

}

void without\_recursion(int\* G, int\* NUM, int size\_G, int v) {

push(v);

while (head) {

int node = pop();

if (\*(NUM + node) == 1) continue;

\*(NUM + node) = 1;

for (int i = size\_G - 1; i >= 0; i--) {

if (\*(G + node \* size\_G + i) == 0 || \*(NUM + i) == 1) continue;

push(i);

}

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

}

printf("\n");

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int size\_G = 6, t1;

int\* G = (int\*)malloc(size\_G \* size\_G \* sizeof(int));

int\* NUM = (int\*)malloc(size\_G \* sizeof(int));

for (int i = 0; i < size\_G; i++) {

for (int l = i; l < size\_G; l++) {

\*(G +i \* size\_G + l) = (rand() % 5 + 7) / 10;

\*(G + l \* size\_G + i) = \*(G + i \* size\_G + l);

if (l == i) \*(G + i \* size\_G + l) = 0;

\*(NUM + i) = 0;

}

}

for (int i = 0; i < size\_G; i++) {

for (int l = 0; l < size\_G; l++) {

printf("%d ", \*(G + i \* size\_G + l));

}

printf("\n");

}

printf("с одним граффом (0/1/2) матрица/списки/не рекурсия\n");

scanf("%d", &t1);

switch (t1) {

case 0:

printf("начало : \n");

scanf("%d", &t1);

matrixs(G, NUM, size\_G, t1-1);

break;

case 1:

for (int i = 0; i < size\_G; i++) {

spstore\_branch(i);

for (int l = 0; l < size\_G; l++)

if (\*(G + i \* size\_G + l) == 1)

spstore\_list(l+1);

}

review\_uz();

printf("\n");

printf("начало : \n");

scanf("%d", &t1);

lists(NUM, size\_G, t1);

break;

case 2:

printf("начало : \n");

scanf("%d", &t1);

without\_recursion(G, NUM, size\_G, t1-1);

break;

}

}

**Вывод**

В ходе выполнения данной лабораторной работы мы научились реализовывать рекурсивный и не рекурсивный обход графа в глубину для списков и матриц списков смежности.