Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Определение характеристик графика»

Выполнили

студент группы 21ВВ3:

Савицкий Макар

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Цель работы**

Реализация матрицы смежности и инцидентности графов и определение характеристик графов по реализованным матрицам.

**Лабораторное задание**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Определите размер графа *G*, используя матрицу смежности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

**Задание 2**

1. Постройте для графа G матрицу инцидентности.
2. Определите размер графа *G*, используя матрицу инцидентности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

**Теоретический материал**

*В теории графов доминирующее множество для графа G = (V, E) — это подмножество D множества вершин V, такое, что любая вершина не из D смежна хотя бы одному элементу из D. Число доминирования γ(G) — это число вершин в наименьшем доминирующем множестве G.*

*Степенью вершины графа G называется число инцидентных ей ребер.*

*Вершина со степенью 0 называется изолированной, со степенью 1 – концевой.*

*Вершина графа, смежная с каждой другой его вершиной, называется доминирующей.*

**Описание метода решения задачи**

**Задание 1**

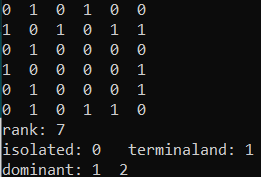
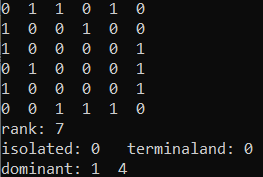
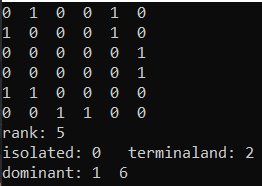
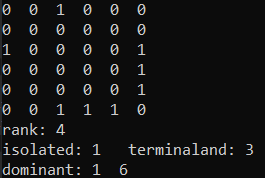
1. *Генерируем граф определенного размера и выводим его на экран*
2. *Проходимся по графу, считая количество изолированных, концевых и доминирующих вершин*
3. *Вводим матрицу на экран*
4. *Вводим найденные значения количество изолированных, концевых и доминирующих вершин*

**Задание 2**

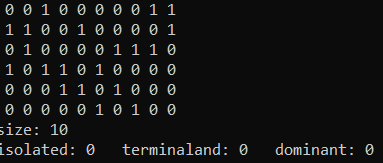
1. *Генерируем случайное число ребер относительно количества вершин*
2. *Выделяем память под матрицы инцидентности*
3. *Заполняем матрицу, исключая повторения ребра*
4. *Считаем количество изолированных, концевых и доминирующих вершин*
5. *Вводим матрицу на экран*
6. *Вводим найденные значения количество изолированных, концевых и доминирующих вершин*

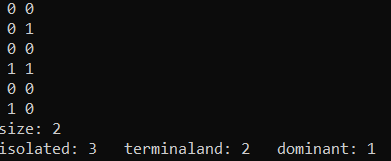
**Результаты работ программы**

**Задание 1**

****

**Задание 2**

****

****

**Листинг 1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //указание прототипов

#include <stdlib.h> //библиотечных функций

#include <windows.h>

#include <malloc.h>

#include <string.h>

#include <time.h>

struct node{

int inf; // приоритет

struct node\* next; // ссылка на следующий элемент

};

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

struct node\* get\_struct(int t) {

struct node\* p = NULL;

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

{

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

exit(1);

}

p->inf = t;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

void spstore(int t) {

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

p = get\_struct(t);

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

head = p;

last = p;

}

else if (head != NULL && p != NULL) { // список уже есть, то вставляем в конец

f = head;

if (f->inf > p->inf) {

p->next = head;

head = p;

}

if (last->inf <= p->inf) {

last->next = p;

last = p;

}

if (last->inf > p->inf && f->inf <= p->inf) {

while (f->next->inf <= p->inf) {

f = f->next;

}

p->next = f->next;

f->next = p;

}

}

return;

}

void review(void) {

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("%d ", struc->inf);

struc = struc->next;

}

printf("\n");

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

struct node\* p = NULL;

int k = 6;

int d[6][6], f[6], t = 0, ii = 0, x = 0, m = -1, u = 0, isolated = 0, terminaland = 0;

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

f[i] = 0;

for (int l = i; l < k; l++) {

d[i][l] = (rand() % 5 + 7) / 10;

d[l][i] = d[i][l];

if (l == i) d[i][l] = 0;

}

}

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

for (int l = i; l < k; l++) {

if (d[l][i] == 1) t += 1;

}

}

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

x = 0;

for (int l = 0; l < k; l++) {

if (d[l][i] == 1) x += 1;

}

if (x == 0) {

isolated += 1;

f[i] = -1;

}

if (x == 1) terminaland += 1;

if (x > m) {

m = x;

ii = i;

}

}

f[ii] = 1;

spstore(ii + 1);

for (int l = 0; l < k; l++) {

if (f[l] == 0) f[l] = d[ii][l];

if (f[l] != 0) u += 1;

}

while (u != k){

m = 0;

u = 0;

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

x = 0;

for (int l = 0; l < k; l++) {

if (d[l][i] == 1 && f[l] == 0) x += 1;

if (l == i && f[l] == 0) x += 1;

}

if (x > m) {

m = x;

ii = i;

}

}

for (int l = 0; l < k; l++) {

if (f[l] != -1) f[l] += 1;

if (f[l] != 0) u += 1;

}

f[ii] = 1;

spstore(ii + 1);

}

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

for (int l = 0; l < k; l++) {

printf("%-3d", d[i][l]);

}

printf("\n");

}

printf("size: %d\nisolated: %d terminaland: %d\ndominant: ", t, isolated, terminaland);

review();

}

**Листинг 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //указание прототипов

#include <stdlib.h> //библиотечных функций

#include <windows.h>

#include <malloc.h>

#include <string.h>

#include <time.h>

struct node {

int inf; // приоритет

struct node\* next; // ссылка на следующий элемент

};

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

struct node\* get\_struct(int t) {

struct node\* p = NULL;

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

{

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

exit(1);

}

p->inf = t;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

void spstore(int t) {

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

p = get\_struct(t);

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

head = p;

last = p;

}

else if (head != NULL && p != NULL) { // список уже есть, то вставляем в конец

f = head;

if (f->inf > p->inf) {

p->next = head;

head = p;

}

if (last->inf <= p->inf) {

last->next = p;

last = p;

}

if (last->inf > p->inf && f->inf <= p->inf) {

while (f->next->inf <= p->inf) {

f = f->next;

}

p->next = f->next;

f->next = p;

}

}

return;

}

void review(void) {

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("%d ", struc->inf);

struc = struc->next;

}

printf("\n");

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

struct node\* p = NULL;

int k = 6;

int d[6][6], f[6], x = 0, y = 0, ii = 0, m = -1, u = 0, isolated = 0, terminaland = 0, dominant = 0;

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

ii += i;

}

ii = rand() % (ii - 1) + 1;

int \*a = (int\*)malloc(6 \* ii \* sizeof(int));

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

for (int l = 0; l < ii; l++) {

a[i \* ii + l] = 0;

}

}

int ppp = 0;

for (int l = 0; l < ii; l++) {

do {

do {

x = rand() % 6;

y = rand() % 6;

} while (y == x);

ppp = 0;

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

if (a[x \* ii + i] == 1 && a[y \* ii + i] == 1)

ppp = 1;

}

} while (ppp == 1);

a[x \* ii + l] = 1;

a[y \* ii + l] = 1;

}

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

x = 0;

for (int l = 0; l < ii; l++) {

if (a[i \* ii + l] != 0) x += 1;

}

if (x == 0) isolated += 1;

if (x == 1) terminaland += 1;

if (x == ii) dominant += 1;

}

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

for (int l = 0; l < ii; l++) {

printf("%2d", a[i \* ii + l]);

}

printf("\n");

}

printf("size: %d\nisolated: %d terminaland: %d dominant: %d\n", ii, isolated, terminaland, dominant);

}

**Вывод**

В ходе выполнения данной лабораторной работы мы научились реализовывать матрицы смежности и инцидентности графов и определение характеристик графов по реализованным матрицам.