Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №5  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Определение характеристик графа.»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Определение характеристик графа

**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.

**Ход работы**

* Задание 1:

Сгенерировал (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Вывел матрицу на экран.

void print\_arr(int\*\* Arr, int size\_column, int size\_row)

{

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

{

for (int j = 0; j < size\_row; j++)

{

printf("%3d ", Arr[i][j]);

}

printf("\n");

}

}

void fill\_arr(int\*\* Arr, int size)

{

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

{

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

{

if (i == j)

{

Arr[i][j] = 0;

continue;

}

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

Arr[j][i] = Arr[i][j];

}

}

}

Определил размер графа *G*, используя матрицу смежности графа.

int size\_arr(int\*\* Arr, int size\_column, int size\_row)

{

int count = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

}

return count / 2;

}

Нашел изолированные, концевые и доминирующие вершины.

int\* peaks(int\*\* Arr, int\* Peaks, int Adjacency, int size\_column, int size\_row)

{

int count = 0;

int peaks = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

if (count == 0)

Peaks[0]++;

if (count == 1)

Peaks[1]++;

if (Adjacency == 0)

{

if (count == size\_row - 1)

Peaks[2]++;

}

else

{

if (count == size\_column - 1)

Peaks[2]++;

}

count = 0;

}

return Peaks;

}

* Задание 2:

Построил для графа G матрицу инцидентности.

void fill\_arr\_inccidence(int\*\* Arr, int\*\* Arrey, int size)

{

int c = 0;

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

{

for (int j = 0; j < size\_arr(Arr, size, size); j++)

{

Arrey[i][j] = 0;

}

}

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

{

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

{

if (Arr[i][j]) {

Arrey[i][c] = 1;

Arrey[j][c++] = 1;

}

}

}

}

Определил размер графа *G*, используя матрицу инцидентности графа.

int size\_arr(int\*\* Arr, int size\_column, int size\_row)

{

int count = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

}

return count / 2;

}

Нашел изолированные, концевые и доминирующие вершины.

int\* peaks(int\*\* Arr, int\* Peaks, int Adjacency, int size\_column, int size\_row)

{

int count = 0;

int peaks = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

if (count == 0)

Peaks[0]++;

if (count == 1)

Peaks[1]++;

if (Adjacency == 0)

{

if (count == size\_row - 1)

Peaks[2]++;

}

else

{

if (count == size\_column - 1)

Peaks[2]++;

}

count = 0;

}

return Peaks;

}

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void fill\_arr(int\*\* Arr, int size);

int\* reset\_arr(int\* Peaks, int size);

void print\_arr(int\*\* Arr, int size\_column, int size\_row);

int size\_arr(int\*\* Arr, int size\_column, int size\_row);

int\* peaks(int\*\* Arr, int\* Peaks, int Adjacency, int size\_column, int size\_row);

void fill\_arr\_inccidence(int\*\* Arr, int\*\* Arrey, int size);

int main(void)

{

srand(time(NULL));

int size;

int\*\* Arr;

int\* Peaks = (int\*)malloc(3 \* sizeof(int));

reset\_arr(Peaks, 3);

printf("Enter the size of the adjacency matrix: ");

scanf("%d", &size);

Arr = (int\*\*)malloc(size \* sizeof(int\*));

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

{

Arr[i] = (int\*)malloc(size \* sizeof(int));

}

printf("Adjacency matrix\n");

fill\_arr(Arr, size);

print\_arr(Arr, size, size);

printf("size arr: %d\n", size\_arr(Arr, size, size));

Peaks = peaks(Arr, Peaks, 0, size, size);

printf("isolated peaks: %d\n", Peaks[0]);

printf("terminal peaks: %d\n", Peaks[1]);

printf("dominant peaks: %d\n", Peaks[2]);

reset\_arr(Peaks, 3);

printf("\n\nInccidence matrix\n");

int\*\* Arrey = (int\*\*)malloc(size \* sizeof(int\*));

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

{

Arrey[i] = (int\*)malloc(size\_arr(Arr, size, size) \* sizeof(int));

}

fill\_arr\_inccidence(Arr, Arrey, size);

print\_arr(Arrey, size, size\_arr(Arr, size, size));

printf("size arr: %d\n", size\_arr(Arrey, size, size\_arr(Arr, size, size)));

Peaks = peaks(Arrey, Peaks, 1, size, size\_arr(Arr, size, size));

printf("isolated peaks: %d\n", Peaks[0]);

printf("terminal peaks: %d\n", Peaks[1]);

printf("dominant peaks: %d\n", Peaks[2]);

free(Arr);

free(Arrey);

free(Peaks);

return(0);

}

void print\_arr(int\*\* Arr, int size\_column, int size\_row)

{

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

{

for (int j = 0; j < size\_row; j++)

{

printf("%3d ", Arr[i][j]);

}

printf("\n");

}

}

void fill\_arr(int\*\* Arr, int size)

{

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

{

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

{

if (i == j)

{

Arr[i][j] = 0;

continue;

}

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

Arr[j][i] = Arr[i][j];

}

}

}

void fill\_arr\_inccidence(int\*\* Arr, int\*\* Arrey, int size)

{

int c = 0;

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

{

for (int j = 0; j < size\_arr(Arr, size, size); j++)

{

Arrey[i][j] = 0;

}

}

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

{

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

{

if (Arr[i][j]) {

Arrey[i][c] = 1;

Arrey[j][c++] = 1;

}

}

}

}

int size\_arr(int\*\* Arr, int size\_column, int size\_row)

{

int count = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

}

return count / 2;

}

int\* peaks(int\*\* Arr, int\* Peaks, int Adjacency, int size\_column, int size\_row)

{

int count = 0;

int peaks = 0;

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

{

for (int j = 0; j < size\_row; j++)

{

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

count++;

}

if (count == 0)

Peaks[0]++;

if (count == 1)

Peaks[1]++;

if (Adjacency == 0)

{

if (count == size\_row - 1)

Peaks[2]++;

}

else

{

if (count == size\_column - 1)

Peaks[2]++;

}

count = 0;

}

return Peaks;

}

int\* reset\_arr(int\* Peaks, int size)

{

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

{

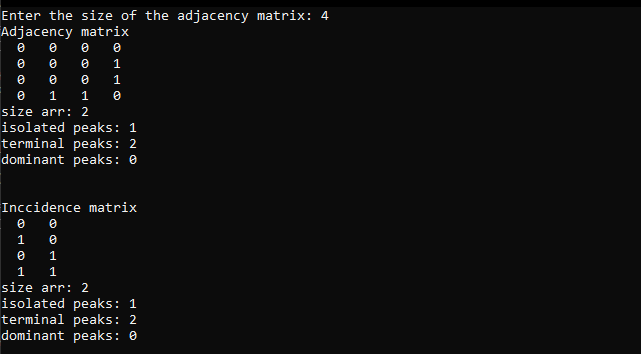
Peaks[i] = 0;

}

return Peaks;

}

**Результат работы программы**

****

**Вывод**

В результате выполнения лабораторной работы я научился определять характеристики графа.