|  |
| --- |
| Министерство образования Российской Федерации  Пензенский государственный университет  Кафедра «Вычислительная техника» |
| Отчет  по лабораторной работе №3  по курсу «Логика и основы алгоритмизации в инженерных задачах»  на тему «Унарные и бинарные операции над графами» |
|  |
|  |
| Выполнил студент группы 19ВВ3:  Трубаненко А.Г.  Принял:  Митрохин М. А. |
| Пенза  2020 |

**Цель работы**

Реализовать алгоритм реализации бинарных и унарных операций.

### **Лабораторное задание**

Создать алгоритм реализации отождествления вершин, стягивания, расщепления вершины, объединения, пересечения, кольцевой суммы.

**Листинг программы**

**Source.cpp**

#include "time.h"

#include "windows.h"

#include "operations.h"

int main()

{

int count = 6, count\_m1 = 6;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

Graph\* graph1 = init\_graph(count);

Graph\* graph2 = init\_graph(count);

generate\_matrix(graph1);

generate\_matrix(graph2);

printf(" Граф G1:\n\n");

print\_graph(graph1);

printf("\n Граф G2:\n\n");

print\_graph(graph2);

////////

Graph\* next\_graph = NULL;

printf("\n Выберите граф для дальнейших операций (1 или 2)");

char graph\_selector = \_getch();

if (graph\_selector=='1'){

next\_graph = graph1;}

else {

next\_graph = graph2;}

Graph\* buffer;

//A

int vertex;

printf("\n Введите номер вершины, которую вы хотете расщепить: ");

scanf("%d", &vertex);

vertex--;

buffer = next\_graph;

next\_graph = rasshep(next\_graph, vertex);

free\_graph(buffer);

printf("\n Результат расщепления вершины:\n\n");

print\_graph(next\_graph);

int point\_1, point\_2;

printf("\n Введите номера вершин, которые вы хотели бы отождествить (в порядке возрастания) : ");

scanf("%d%d", &point\_1, &point\_2);

point\_1--;

point\_2--;

otojd(next\_graph, point\_1, point\_2);

printf("\n Граф 1 после операции отождествления:\n\n");

print\_graph(next\_graph);

//A

//B

printf("\n Введите номера вершин, ребро между которыми вы хотели бы стянуть (в порядке возрастания) : ");

scanf("%d%d", &point\_1, &point\_2);

point\_1--;

point\_2--;

stag(next\_graph, point\_1, point\_2);

printf("\n Результат стягивания вершин:\n\n");

print\_graph(next\_graph);

//B

if (graph\_selector == '1') {

graph1 = next\_graph;

}

else {

graph2 = next\_graph;

}

//A2

\_getch();

printf("\n Результат после операции обЪединения Mass1 и Mass2:\n\n");

Graph\* graph3 = unionn(graph1, graph2);

print\_graph(graph3);

//A2

//B2

\_getch();

printf("\n Результат после операции пересечения Mass1 и Mass2:\n\n");

graph3 = cross(graph1, graph2);

print\_graph(graph3);

//B2

//C3

\_getch();

printf("\n Результат после операции кольццевой суммы Mass1 и Mass2:\n\n");

graph3 = circle\_sum(graph1, graph2);

print\_graph(graph3);

//C3

\_getch();

}

**operations.cpp**

#include "operations.h"

#define max(a,b) (a > b ? a : b)

#define min(a,b) (a < b ? a : b)

Graph\* cross(Graph\* graph1, Graph\* graph2) {

int max\_size = max(graph1->size, graph2->size);

int min\_size = min(graph1->size, graph2->size);

Graph\* graph\_max;

(max\_size == graph1->size ? graph\_max = graph1 : graph\_max = graph2);

Graph\* graph\_res = init\_graph(max\_size);

for (int i = 0; i < min\_size; i++) {

for (int j = 0; j < min\_size; j++) {

graph\_res->matrix[i][j] = graph1->matrix[i][j] & graph2->matrix[i][j];

}

}

if (max\_size != min\_size) {

for (int i = min\_size; i < max\_size; i++) {

for (int j = 0; j < max\_size; j++) {

graph\_res->matrix[i][j] = graph\_res->matrix[j][i] = graph\_max->matrix[i][j];

}

}

}

return graph\_res;

}

Graph\* unionn(Graph\* graph1, Graph\* graph2) {

int max\_size = max(graph1->size, graph2->size);

int min\_size = min(graph1->size, graph2->size);

Graph\* graph\_max;

(max\_size == graph1->size ? graph\_max = graph1 : graph\_max = graph2);

Graph\* graph\_res = init\_graph(max\_size);

for (int i = 0; i < min\_size; i++) {

for (int j = 0; j < min\_size; j++) {

graph\_res->matrix[i][j] = graph1->matrix[i][j] | graph2->matrix[i][j];

}

}

if (max\_size != min\_size) {

for (int i = min\_size; i < max\_size; i++) {

for (int j = 0; j < max\_size; j++) {

graph\_res->matrix[i][j] = graph\_res->matrix[j][i] = graph\_max->matrix[i][j];

}

}

}

return graph\_res;

}

int otojd(Graph\* graph, unsigned int point\_1, unsigned int point\_2) {

if (point\_1 < graph->size && point\_2 < graph->size) {

for (int i = 0; i < graph->size; i++) {

if (graph->matrix[point\_1][i] == 1 || graph->matrix[point\_2][i] == 1) {

graph->matrix[point\_1][i] = 1;

graph->matrix[i][point\_1] = 1;

}

}

for (int i = point\_2; i < graph->size - 1; i++) {

for (int j = 0; j < graph->size; j++) {

graph->matrix[i][j] = graph->matrix[i + 1][j];

}

}

for (int i = point\_2; i < graph->size - 1; i++) {

for (int j = 0; j < graph->size; j++) {

graph->matrix[j][i] = graph->matrix[j][i + 1];

}

}

printf(" (в результирующей матрице номера вершин начинная с %d-ой уменьшаются на единицу)\n", point\_2 + 2);

graph->size--;

return graph->size;

}

}

int stag(Graph\* graph, unsigned int point\_1, unsigned int point\_2) {

if (graph->matrix[point\_1][point\_2] == 0) {

printf("\n Вершины не ицедентны.\n");

return graph->size;

}

else {return otojd(graph, point\_1, point\_2);}

}

Graph\* rasshep(Graph\* graph, unsigned int v) {

if (v < graph->size){

Graph\* new\_graph = init\_graph(graph->size + 1);

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

new\_graph->matrix[i][j] = graph->matrix[i][j];

}

}

for (int i = 0; i < graph->size; i++) {

new\_graph->matrix[graph->size][i] = new\_graph->matrix[i][graph->size] = graph->matrix[v][i];

}

new\_graph->matrix[graph->size][graph->size] = 0;

return new\_graph;

}

return NULL;

}

Graph\* circle\_sum(Graph\* graph1, Graph\* graph2) {

int max\_size = max(graph1->size, graph2->size);

int min\_size = min(graph1->size, graph2->size);

Graph\* graph\_max;

(max\_size == graph1->size ? graph\_max = graph1 : graph\_max = graph2);

Graph\* graph\_res = init\_graph(max\_size);

for (int i = 0; i < min\_size; i++) {

for (int j = 0; j < min\_size; j++) {

graph\_res->matrix[i][j] = graph1->matrix[i][j] ^ graph2->matrix[i][j];

}

}

if (max\_size != min\_size) {

for (int i = min\_size; i < max\_size; i++) {

for (int j = 0; j < max\_size; j++) {

graph\_res->matrix[i][j] = graph\_res->matrix[j][i] = graph\_max->matrix[i][j];

}

}

}

return graph\_res;

}

**Matrix\_Graph.cpp**

#include "Matrix\_Graph.h"

int\*\* matrix(int size)

{

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

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

{

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

}

return matrix;

}

void generate\_matrix(Graph\* graph)

{

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

if (j > i) {

graph->matrix[i][j] = rand() % 2;

graph->matrix[j][i] = graph->matrix[i][j];

}

else if (i == j) graph->matrix[i][j] = 0;

}

}

}

Graph\* init\_graph(int size)

{

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->matrix = matrix(size);

graph->size = size;

return graph;

};

void free\_graph(Graph\* graph)

{

for (int i = 0; i < graph->size; i++)

{

free(graph->matrix[i]);

}

free(graph->matrix);

free(graph);

}

void print\_graph(Graph\* graph)

{

printf(" ");

for (int k = 1; k <= graph->size; k++) printf(" %d", k);

printf("\n -----------\n");

for (int i = 0; i < graph->size; i++) {

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

for (int j = 0; j < graph->size; j++) {

printf("%d ", graph->matrix[i][j]);

}

printf("\n");

}

}

**Matrix\_Graph.h**

#pragma once

#include "stdlib.h"

#include "stdio.h"

typedef struct Graph

{

int\*\* matrix;

int size;

}Graph;

int\*\* matrix(int size);

void generate\_matrix(Graph\* graph);

Graph\* init\_graph(int size);

void free\_graph(Graph\* graph);

void print\_graph(Graph\* graph);

**operations.h**

#pragma once

#include "conio.h"

#include "Matrix\_Graph.h"

#pragma warning (disable : 4996)

Graph\* cross(Graph\* graph1, Graph\* graph2);

Graph\* unionn(Graph\* graph1, Graph\* graph2);

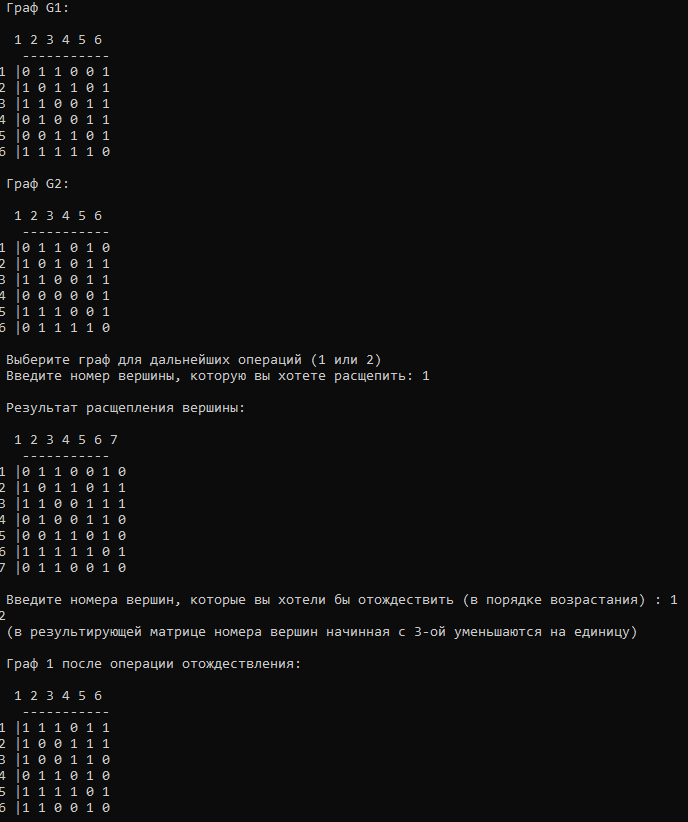
int otojd(Graph\* graph, unsigned int point\_1, unsigned int point\_2);

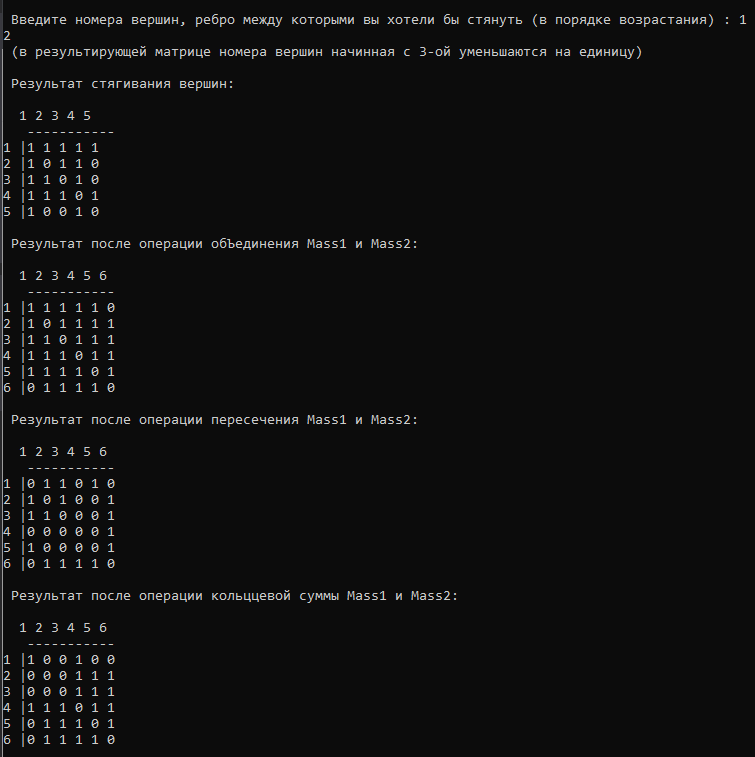
int stag(Graph\* graph, unsigned int point\_1, unsigned int point\_2);

Graph\* rasshep(Graph\* graph, unsigned int v);

Graph\* circle\_sum(Graph\* graph1, Graph\* graph2);

**Результат работы программы:**





**Вывод:** Реализовал бинарные и унарные операции над графами, представленными матрицей смежности.