Министерство высшего образования и науки Российской Федерации

Пензенский Государственный Университет

Кафедра «Вычислительная техника»

Выполнили:

студенты группы 20ВВ2

Аверочкин П.С.

Зиновьев Я.М.

Принял:

к. т. н. доцент Юрова О.В.

д. т. н., профессор Митрохин М.А.

Пенза 2020

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Л и ОА в ИС»

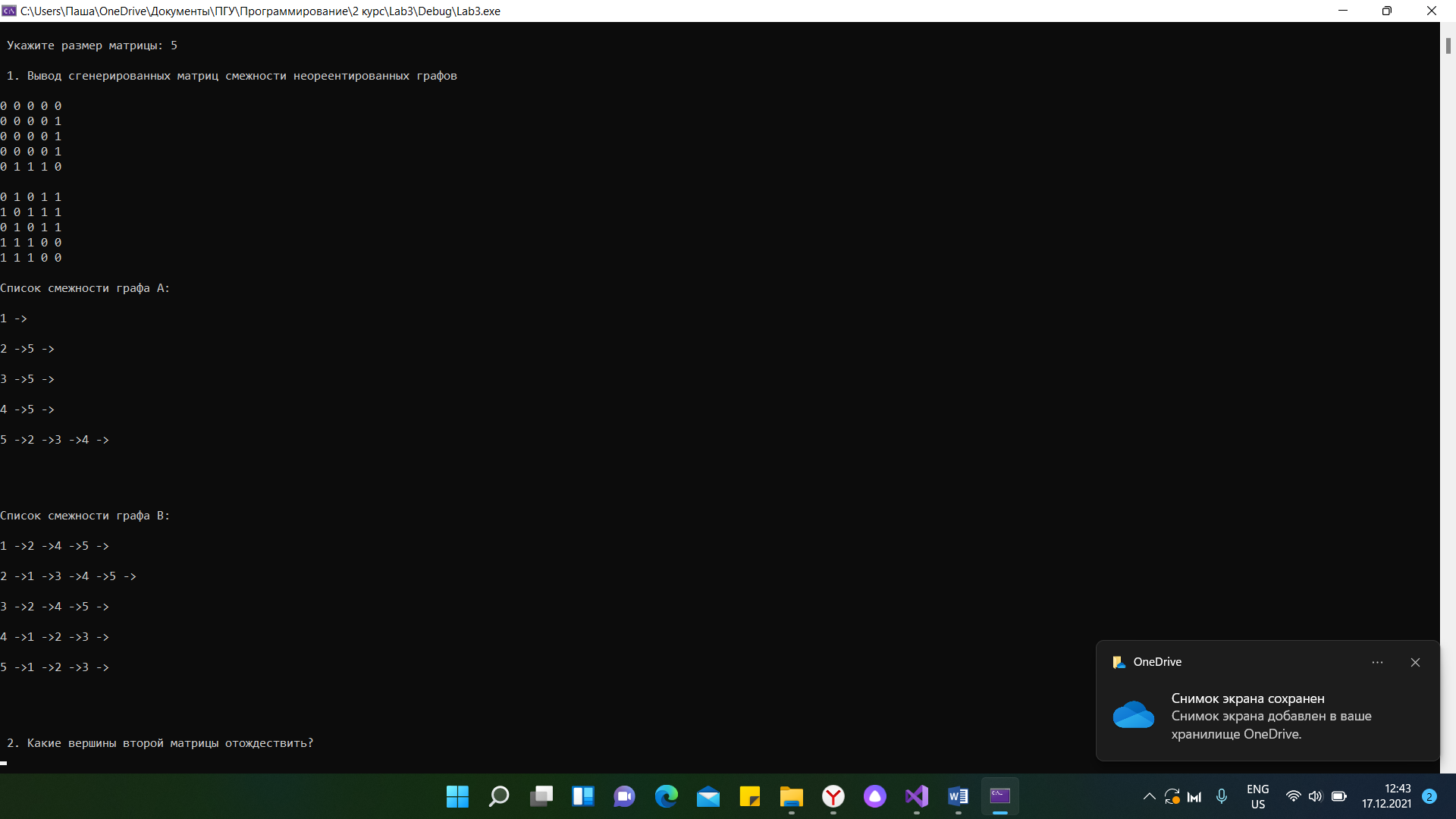
на тему «Унарные и бинарные операции над графами»

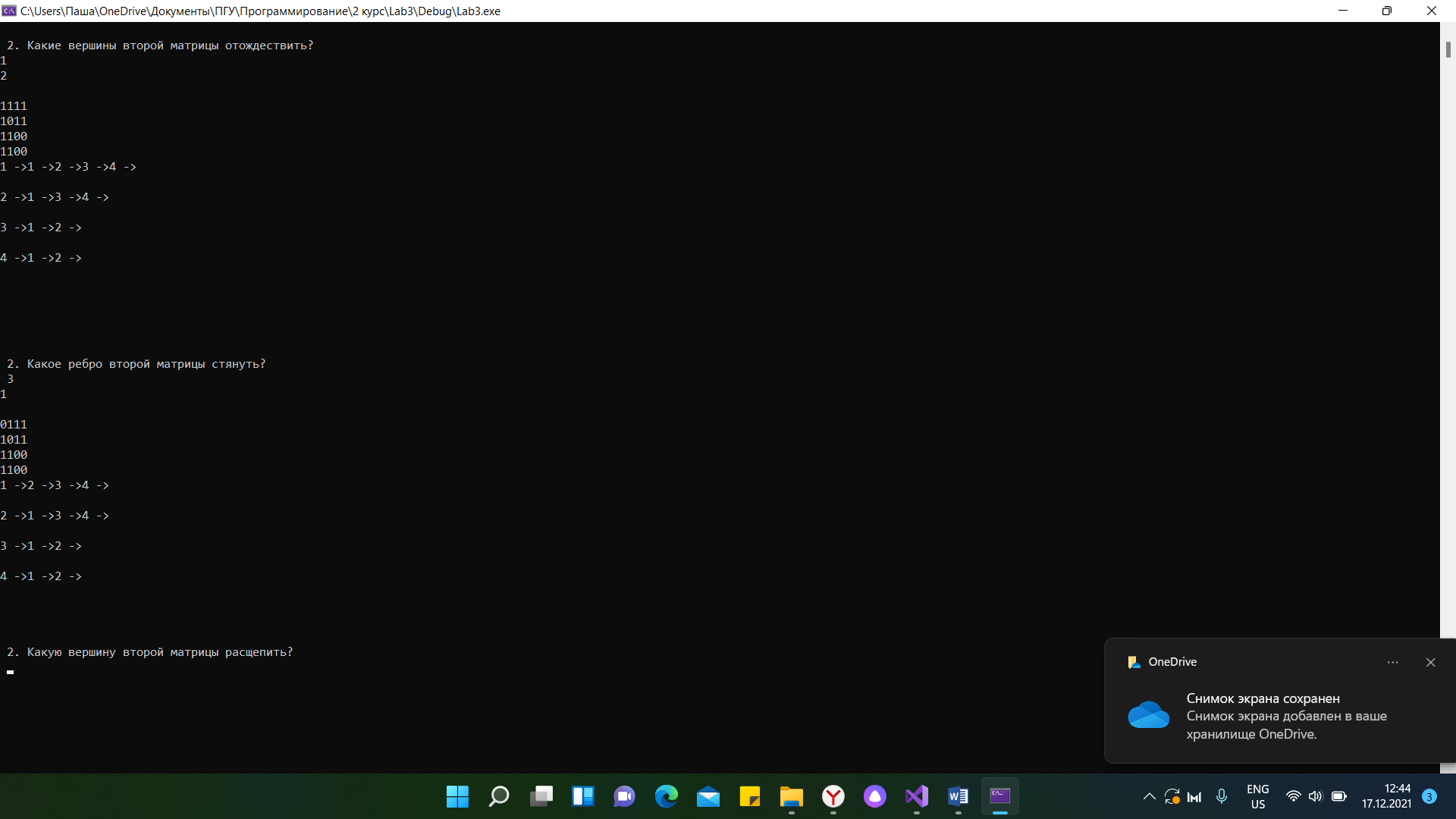
Ход работы:

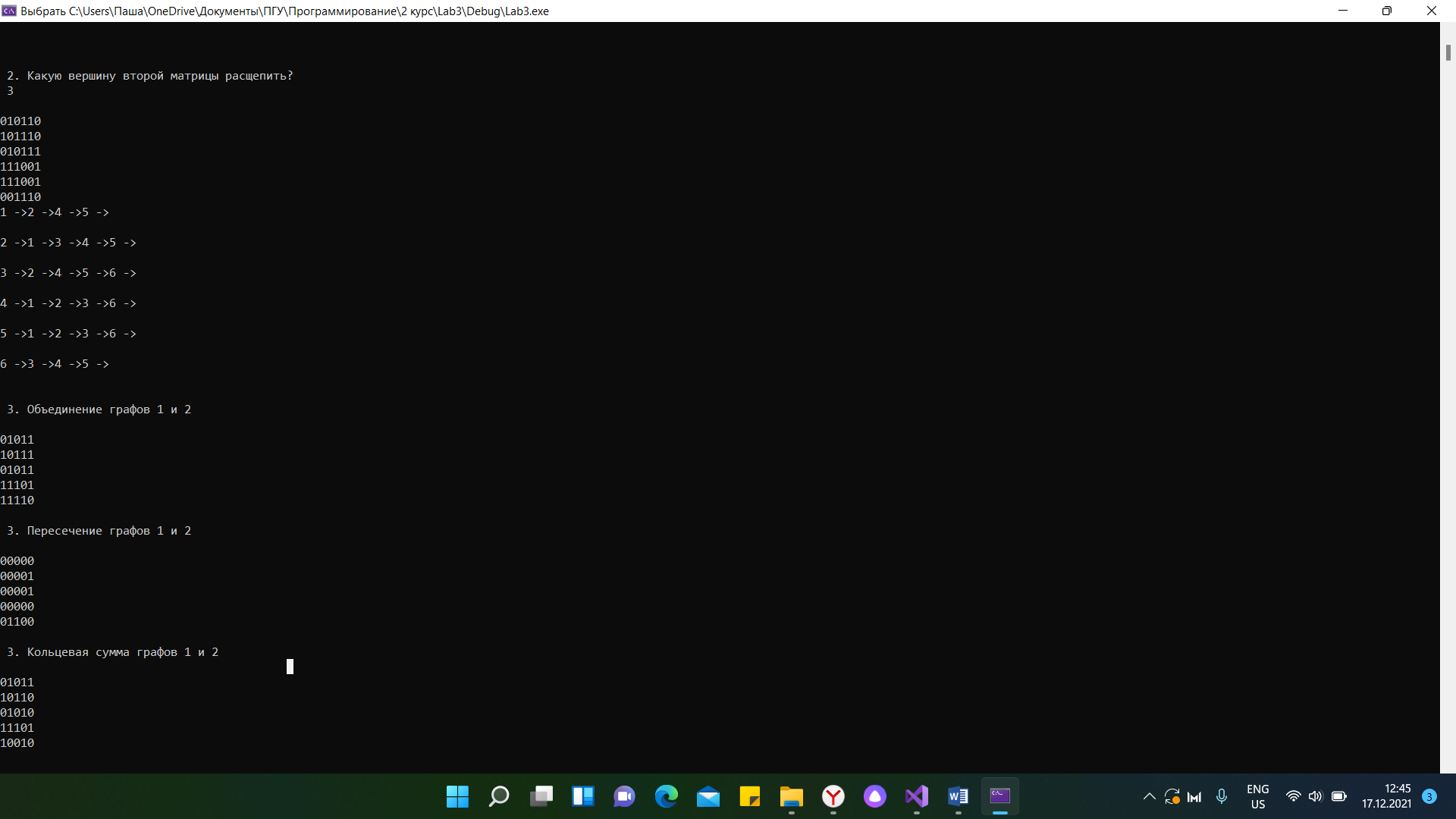
Программа производит унарные и бинарные операции над графами в представлении матрицы смежности и построенными на ней списком смежности. С помощью динамического массива задаются две матрицы смежности равного размера, затем по ним благодаря структурам строятся списки смежности (все вершины, записанные в структуру являются смежными). Далее юзеру предлагается отождествить вершины, стянуть ребра и расщепить вершину. Данная операция производится с помощью добавления/удаления членов структуры. Над матрицами смежности все намного проще, создается массив, в которой переписывается новый граф.

Так же в конце программа просчитывает объединение графов ( совокупность всех вершин графов), пересечение графов ( совокупность одинаковых вершин графов), кольцевую сумму графов, декартово произведение.

Результат работы программы:







((())))

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

#include <iostream>

#include <locale.h>

#include <malloc.h>

using namespace std;

int\*\* mas1, \*\* mas2, \*\* m\_longed, \*\* mas3;

int N;

typedef struct Node {

char vertex;

struct Node\* next;

} Node;

void add\_Node\_to\_beginning(Node\*\* head, int data) //функция добавления узла в начало

{

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = (\*head); //присваиваем указателю tmp адрес след. узла

(\*head) = tmp\_node; //Присваиваем указателю head адрес tmp

} //после выхода из функции tmp уничтожается

int delete\_node(Node\*\* head) { //удаляем узел на который указывает адрес

Node\* prev = NULL;

int val;

prev = (\*head); //передаем во временную переменную prev адрес

val = prev->vertex;

(\*head) = (\*head)->next; //в head кладем адрес следующего узла

free(prev); //очищаем временную переменную prev

return val;

}

Node\* findLastHead(Node\* head) { //поиск адреса последнего элемента

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void add\_Node\_to\_end(Node\* head, int data) { //функция добавления нового узла в конец

Node\* last = findLastHead(head); //получаем указатель на последний элемент списка

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = NULL;

last->next = tmp\_node; //записываем в последний элемент списка указатель на новый узел

}

void createLinkedList(int\*\* mas, Node\*\* head, int N) //создаем связный список

{

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

{

add\_Node\_to\_beginning(&head[i], i); //передаем адрес вершины и ее номер

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

{

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

{

add\_Node\_to\_end(head[i], j); //добавляем к вершине связные с ней вершины

}

}

}

}

void cleanlist(Node\*\* head, int n)

{

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

head[i] = NULL;

}

void cleanlistfield(Node\*\* head, int del, int n)

{

Node\* tmp;

int back;

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

{

tmp = head[i];

while (tmp)

{

if (tmp->next)

if (tmp->next->vertex == del)

{

tmp->next = tmp->next->next;

}

if (tmp->vertex >= del)

{

back = tmp->vertex;

back--;

tmp->vertex = back;

}

tmp = tmp->next;

}

}

for (int i = del; i < n - 1; i++)

{

head[i] = head[i + 1];

}

head[n - 1] = NULL;

}

void paste(Node\* head, int m)

{

Node\* tmp;

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

tmp->vertex = m;

tmp->next = head->next;

head->next = tmp;

}

void addver(Node\*\* head, int field, int n)

{

Node\* ptr, \* tmp;

int base = head[field]->vertex, search;

tmp = head[field];

tmp = tmp->next;

while (tmp)

{

search = tmp->vertex;

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

{

if (head[i]->vertex == search)

{

if (!head[i]->next)

{

add\_Node\_to\_beginning(&head[i], base);

}

ptr = head[i];

while (ptr->next)

{

if (ptr->next->vertex == base)

{

break;

}

if (ptr->next->vertex > base)

{

paste(ptr, base);

}

}

break;

}

}

tmp = tmp->next;

}

}

void printLinkedList(Node\* head) {

cout << endl;

while (head) {

printf("%d —>", head->vertex + 1);

head = head->next;

}

cout << endl;

}

void identificationlist(int v1, int v2, Node\*\* head, int n)

{

int del, field;

Node\* ptr = NULL, \* tmp;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

tmp = head[del]->next;

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

ptr->vertex = head[field]->vertex;

ptr->next = NULL;

head[field] =

head[field]->next;

while (head[field] || tmp)

{

if (head[field] && tmp)

{

if (head[field]->vertex > tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

continue;

}

if (head[field]->vertex < tmp->vertex)

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

continue;

}

if (head[field]->vertex == tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

head[field] = head[field]->next;

continue;

}

}

if (head[field])

{

while (head[field])

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

}

}

else

{

while (tmp)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

}

}

}

head[field] = ptr;

fflush(stdin);

cleanlistfield(head, del, n);

addver(head, field, n);

}

void identificationlist\_2(int v1, int v2, Node\*\* head, int n)

{

int del, field;

Node\* ptr = NULL, \* tmp;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

tmp = head[del]->next;

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

ptr->vertex = head[field]->vertex;

ptr->next = NULL;

head[field] = head[field]->next;

while (head[field] || tmp)

{

if (head[field] && tmp)

{

if (head[field]->vertex > tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

continue;

}

if (head[field]->vertex < tmp->vertex)

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

continue;

}

if (head[field]->vertex == tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

head[field] = head[field]->next;

continue;

}

}

if (head[field])

{

while (head[field])

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

}

}

else

{

while (tmp)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

}

}

}

head[field] = ptr;

fflush(stdin);

cleanlistfield(head, del, n);

}

void splitinglist(Node\*\* head, int v, int n)

{

add\_Node\_to\_beginning(&head[n - 1], n - 1);

Node\* ptr = head[v];

if (ptr->next)

{

if (ptr->vertex < ptr->next->vertex)

{

add\_Node\_to\_end(head[n - 1], ptr->vertex);

add\_Node\_to\_end(head[n - 1], ptr->next->vertex);

ptr->next = ptr->next->next;

}

else

{

add\_Node\_to\_end(head[n - 1], ptr->next->vertex);

add\_Node\_to\_end(head[n - 1], ptr->vertex);

ptr->next = ptr->next->next;

}

}

else

{

add\_Node\_to\_end(head[n], ptr->vertex);

}

}

void main() {

setlocale(LC\_ALL, "russian");

cout << "\n Укажите размер матрицы: ";

cin >> N;

Node\*\* head = (Node\*\*)malloc(N \* sizeof(Node\*));

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

head[i] = NULL;

mas1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mas1[i] = (int\*)malloc(N \* sizeof(int));

}

mas2 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mas2[i] = (int\*)malloc(N \* sizeof(int));

}

srand(time(NULL));

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

{

mas1[i][i] = 0;

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

{

mas1[i][j] = rand() % 2;

mas1[j][i] = mas1[i][j];

}

}

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

{

mas2[i][i] = 0;

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

{

mas2[i][j] = rand() % 2;

mas2[j][i] = mas2[i][j];

}

}

printf("\n 1. Вывод сгенерированных матриц смежности неореентированных графов \n");

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

cout << endl;

for (int j = 0; j < N; j++) {

cout << mas1[i][j];

}

}

cout << endl;

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

cout << endl;

for (int j = 0; j < N; j++) {

cout << mas2[i][j];

}

}

createLinkedList(mas1, head, N);

cout << "\n\nСписок смежности графа А:" << endl;;

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

printLinkedList(head[i]);

cleanlist(head, N);

cout << endl << endl;

createLinkedList(mas2, head, N);

cout << "\n\nСписок смежности графа B:\n";

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

printLinkedList(head[i]);

cleanlist(head, N);

cout << endl << endl;

m\_longed = (int\*\*)malloc((N + 1) \* sizeof(int\*)); printf("\t");

for (int i = 0; i < (N + 1); i++)

{

m\_longed[i] = (int\*)malloc((N + 1) \* sizeof(int));

}

int v1, v2;

createLinkedList(mas2, head, N);

cout << "\n\n 2. Какие вершины второй матрицы отождествить?\n" << " ";

cin >> v1 >> v2;

int del, not\_del;

int i1 = 0, j = 0, j1 = 1, sch = 0;

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

if (v1 > v2)

{

del = v1 - 1;

not\_del = v2 - 1;

}

else

{

del = v2 - 1;

not\_del = v1 - 1;

}

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

{

if (m\_longed[del][i] == 1)

m\_longed[not\_del][i] = 1;

if (m\_longed[i][del] == 1)

m\_longed[i][not\_del] = 1;

}

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

{

for (int j = del; j < N - 1; j++)

{

m\_longed[i][j] = m\_longed[i][j + 1];

}

}

for (int i = del; i < N - 1; i++)

{

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

{

m\_longed[i][j] = m\_longed[i + 1][j];

}

}

if (mas2[del][not\_del]) {

m\_longed[not\_del][not\_del] = 1;

}

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

{

cout << endl;

for (int j = 0; j < (N - 1); j++)

{

cout << m\_longed[i][j];

}

}

identificationlist(v1, v2, head, N);

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

printLinkedList(head[i]);

cleanlist(head, N);

cout << endl << endl;

createLinkedList(mas2, head, N);

cout << "\n\n 2. Какое ребро второй матрицы стянуть?\n" << " ";

cin >> v1 >> v2;

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

if (v1 > v2)

{

del = v1 - 1;

not\_del = v2 - 1;

}

else

{

del = v2 - 1;

not\_del = v1 - 1;

}

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

{

if (m\_longed[del][i] == 1)

m\_longed[not\_del][i] = 1;

if (m\_longed[i][del] == 1)

m\_longed[i][not\_del] = 1;

}

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

{

for (int j = del; j < N - 1; j++)

{

m\_longed[i][j] = m\_longed[i][j + 1];

}

}

for (int i = del; i < N - 1; i++)

{

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

{

m\_longed[i][j] = m\_longed[i + 1][j];

}

}

m\_longed[not\_del][not\_del] = 0;

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

{

cout << endl;

for (int j = 0; j < (N - 1); j++)

{

cout << m\_longed[i][j];

}

}

int N2 = N + 1;

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

for (int i = 0; i < N2; i++)

{

for (int j = 0; j < N2; j++)

{

if (m\_longed[i][j] < 0 || m\_longed[i][j] > 1)

m\_longed[i][j] = 0;

}

}

identificationlist\_2(v1, v2, head, N);

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

printLinkedList(head[i]);

cleanlist(head, N);

cout << "\n\n 2. Какую вершину второй матрицы расщепить?\n" << " ";

cin >> v1;

v1 = v1 - 1;

int i = v1;

for (; i < N2; i++)

{

if (m\_longed[v1][i] == 1)

{

m\_longed[N2 - 1][i] = m\_longed[i][N2 - 1] = 1;

}

else

m\_longed[N2 - 1][i] = m\_longed[i][N2 - 1] = 0;

}

m\_longed[N2 - 1][v1] = m\_longed[v1][N2 - 1] = 1;

for (int i = 0; i < N2; i++)

{

cout << endl;

for (int j = 0; j < N2; j++)

{

cout << m\_longed[i][j];

}

}

cleanlist(head, N + 1);

createLinkedList(m\_longed, head, N + 1);

for (int i = 0; i < N + 1; i++)

printLinkedList(head[i]);

cleanlist(head, N);

cout << "\n\n 3. Объединение графов 1 и 2\n";

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

{

cout << endl;

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

{

m\_longed[i][j] = mas1[i][j] || mas2[i][j];

cout << m\_longed[i][j];

}

}

cout << "\n\n 3. Пересечение графов 1 и 2\n";

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

{

cout << endl;

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

{

m\_longed[i][j] = mas1[i][j] & mas2[i][j];

cout << m\_longed[i][j];

}

}

cout << "\n\n 3. Кольцевая сумма графов 1 и 2\n";

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

{

cout << endl;

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

{

m\_longed[i][j] = mas1[i][j] ^ mas2[i][j];

cout << m\_longed[i][j];

}

}

cout << "\n\nДекартово произведение A и B";

mas3 = (int\*\*)malloc(N \* sizeof(int\*));

for (int i = 0; i < (N \* N); i++)

{

mas3[i] = (int\*)malloc((N \* N) \* sizeof(int));

}

int a1, a2;

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

{

for (int m = 0; m < N; m++)

{

cout << endl << endl;

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

{

for (int k = 0; k < N; k++)

{

if (i == j) {

mas3[][] = mas2[m][k];

}

if (m == k)

{

mas3[][] = mas1[i][j];

}

if ((i != j) && (m != k))

{

mas3[i][j] = 0;

}

}

}

}

}

cout << endl;

for (i = 0; i < N \* N; i++)

{

for (j = 0; j < N \* N; j++)

{

cout << mas3[i][j] << " ";

}

cout << endl;

}

cout << endl << endl;

\_getch();

}

Вывод: Мы научились производить унарные и бинарные операции над графами.