Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 6

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

## на тему: "Унарные и бинарные операции над графами"

Выполнили студенты гр. 22ВВП2:

Корнилов В.М.

Самофалова А.В.

Горбунов Д.А.

Приняли:

Акифьев И.В.

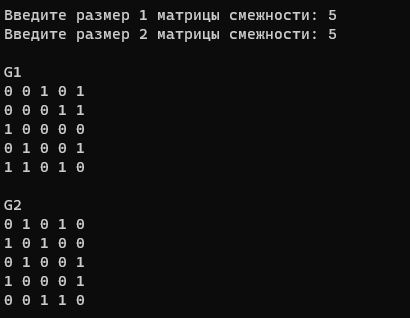
Юрова О.В.

Пенза, 2023

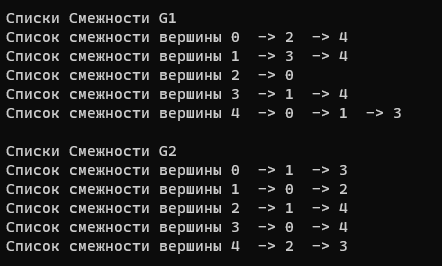
**Цель: Научиться выполнять унарные и бинарные операции над графами.**

**Задание 1:**

1. Сгенерировать (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Выведите сгенерированные матрицы на экран.



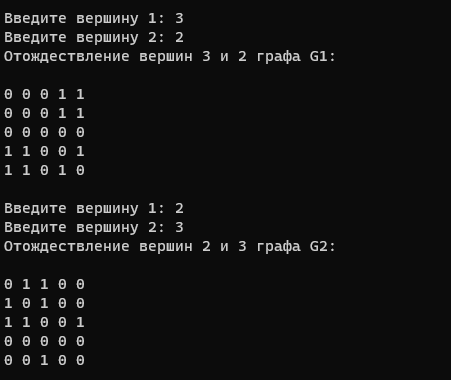
1. \* Для указанных графов преобразовать представление матриц смежности в списки смежности. Выведите полученные списки на экран.



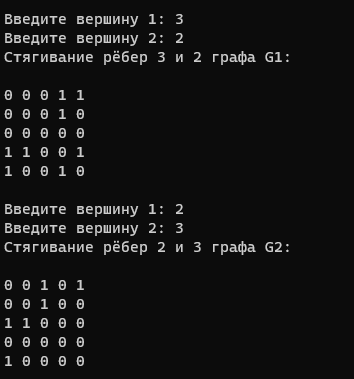
**Задание 2:**

1. Для матричной формы представления графов выполните операцию:

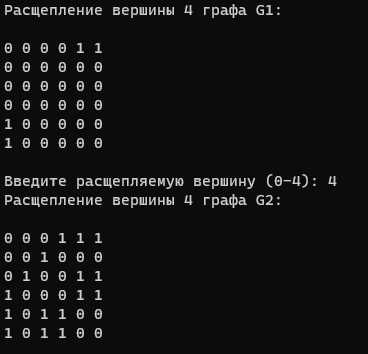
а) отождествления вершин



б) стягивания ребра

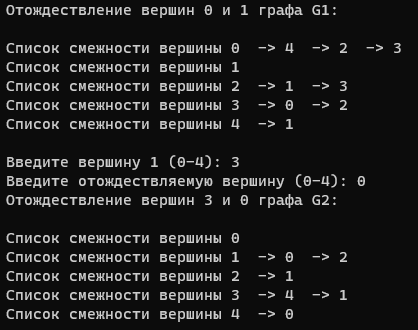


в) расщепления вершины

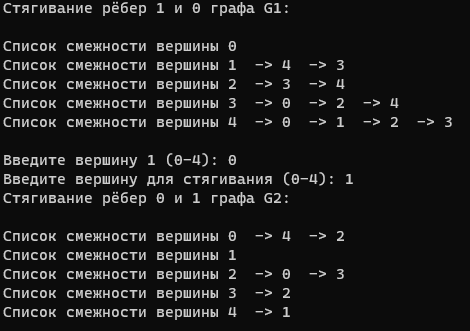


1. \* Для представления графов в виде списков смежности выполните операцию:

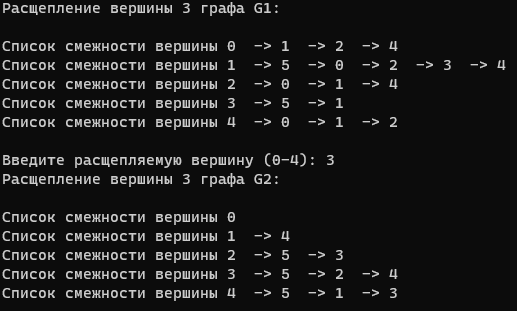
а) отождествления вершин



б) стягивания ребра



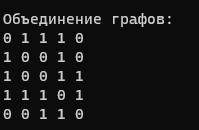
в) расщепления вершины



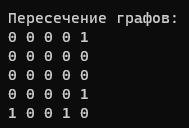
**Задание 3:**

1. Для матричной формы представления графов выполните операцию:

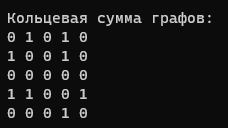
а) объединения *G* = *G*1  *G*2



б) пересечения *G* = *G*1  *G*2

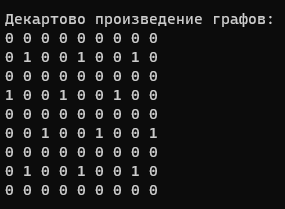


в) кольцевой суммы *G* = *G*1  *G*2

****

**Задание 4:**

1. \* Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.



**Вывод:** научились выполнять унарные и бинарные операции над графами.

**Приложение А (Листинг)**

**Файл spasite.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <locale.h>

#include <time.h>

#include <malloc.h>

int\*\* A;

int n, m;

int\*\* B;

int\*\* res;

int cc;

int ver1, ver2;

void zad1();

void zad2();

void zad3();

void zad4();

void zad21();

void zad22();

void zad211();

void zad212();

void zad213();

void zad221();

void zad222();

void zad223();

void zad31();

void zad32();

void zad33();

void menu();

typedef struct Node {

int ver;

struct Node\* next;

} Node;

void addver(Node\*\* List, int x, int y) {

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

newNode->ver = y;

newNode->next = NULL;

if (List[x] == NULL) {

List[x] = newNode;

}

else {

Node\* temp = List[x];

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

return;

}

void addEdge(Node\*\* List, int src, int dest) {

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

newNode->ver = dest;

newNode->next = List[src];

List[src] = newNode;

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

newNode->ver = src;

newNode->next = List[dest];

List[dest] = newNode;

}

void vivod(Node\*\* List, int n) {

int i;

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

Node\* temp = List[i];

printf("Список смежности вершины %d ", i);

while (temp) {

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

}

void ListidenVer(Node\*\* List, int ver1, int ver2) {

while (List[ver2] != NULL) {

Node\* temp = List[ver2];

if (List[ver2]->ver == ver1) {

List[ver2] = List[ver2]->next;

continue;

}

List[ver2] = List[ver2]->next;

temp->next = List[ver1];

List[ver1] = temp;

}

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

Node\* curr = List[i];

while (curr) {

if (curr->ver == ver2) {

curr->ver = ver1;

break;

}

curr = curr->next;

}

}

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

Node\* curr = List[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

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

Node\* temp = List[i];

if (i == ver2) continue;

printf("Список смежности вершины %d ", i);

while (temp) {

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

}

void ListcontEdge(Node\*\* List, int ver1, int ver2) {

Node\* prev = NULL;

Node\* curr = List[ver1];

int count=0;

while (curr) {

if (curr->ver == ver2) {

count = 1;

break;

}

curr = curr->next;

}

if (count == 0) {

printf("Ребра нет\n");

return;

}

Node\* current = List[ver1];

while (current != NULL) {

if (current->ver == ver2) {

if (prev == NULL) {

List[ver1] = current->next;

}

else {

prev->next = current->next;

}

free(current);

break;

}

prev = current;

current = current->next;

}

current = List[ver2];

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

Node\* curr = List[i];

while (curr) {

if (curr->ver == ver2) {

curr->ver = ver1;

break;

}

curr = curr->next;

}

}

while (current != NULL) {

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

newNode->ver = current->ver;

newNode->next = List[ver1];

if (newNode->ver == ver1) {

current = current->next;

continue;

}

List[ver1] = newNode;

current = current->next;

}

List[ver2] = NULL;

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

Node\* curr = List[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

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

Node\* temp = List[i];

if (i == ver2) continue;

printf("Список смежности вершины %d ", i);

while (temp) {

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

}

void splitVer(int\*\* A, int ver1, int n) {

if (ver1 >= n) {

printf("Неверная вершина\n");

return;

}

A = (int\*\*)realloc(A, (n + 1) \* sizeof(int\*));

A[n] = (int\*)calloc(n + 1, sizeof(int));

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

A[i] = (int\*)realloc(A[i], (n + 1) \* sizeof(int));

for (int j = n; j > ver1; j--) {

A[i][j] = A[i][j - 1];

}

}

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

if (i == ver1) {

A[i][ver1] = 0;

}

else {

A[i][ver1] = A[ver1][i] / 2;

A[ver1][i] /= 2;

}

}

n++;

}

void ListsplitVer(Node\*\* List, int ver1, int n) {

addEdge(List, n, ver1);

Node\* current = List[ver1];

while (current != NULL) {

addEdge(List, n, current->ver);

current = current->next;

}

n++;

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

Node\* curr = List[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

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

Node\* temp = List[i];

printf("Список смежности вершины %d ", i);

while (temp) {

if ((i == n - 1) && (temp->ver == n - 1)) {

temp = temp->next;

continue;

}

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

}

void removeDuplicates(Node\*\* List, int n) {

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

Node\* curr = List[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

}

void end() {

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

free(A[i]);

}

free(A);

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

free(B[i]);

}

free(B);

while (1) {

printf("\n");

printf("Введите любое число, чтобы вернуться в меню.\n");

scanf("%d", &cc);

system("cls");

menu();

}

}

void zad1() {

printf("Введите размер 1 матрицы смежности: ");

scanf("%d", &n);

printf("Введите размер 2 матрицы смежности: ");

scanf("%d", &m);

printf("\n");

srand(time(NULL));

A = (int\*\*)malloc(sizeof(int\*) \* n);

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

A[i] = (int\*)malloc(sizeof(int) \* n);

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

for (int j = i; j < n; j++) {

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

A[j][i] = A[i][j];

if (i == j) A[i][j] = 0;

}

printf("G1\n");

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

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

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

}

printf("\n");

}

printf("\n");

B = (int\*\*)malloc(sizeof(int\*) \* m);

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

B[i] = (int\*)malloc(sizeof(int) \* m);

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

for (int j = i; j < m; j++) {

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

B[j][i] = B[i][j];

if (i == j) B[i][j] = 0;

}

printf("G2\n");

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

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

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

}

printf("\n");

}

Node\*\* list1 = (Node\*\*)malloc(n \* sizeof(Node\*));

int i, j;

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

list1[i] = NULL;

}

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

for (j = 0; j < n; ++j) {

if (A[i][j] == 1) {

addver(list1, i, j);

}

}

}

printf("\n");

printf("Списки Смежности G1\n");

vivod(list1, n);

printf("\n");

printf("Списки Смежности G2\n");

Node\*\* list2 = (Node\*\*)malloc(m \* sizeof(Node\*));

int z, x;

for (z = 0; z < m; ++z) {

list2[z] = NULL;

}

for (z = 0; z < m; ++z) {

for (x = 0; x < m; ++x) {

if (B[z][x] == 1) {

addver(list2, z, x);

}

}

}

vivod(list2, m);

}

void menu() {

while (1) {

printf("Задание 1 выводится по умолчанию\n");

printf("1. Задание 2\n");

printf("2. Задание 3\n");

printf("3. Задание 4\n");

printf("4. Выход\n");

scanf("%d", &cc);

switch (cc) {

case 1:

system("cls");

zad2();

break;

case 2:

system("cls");

zad3();

break;

case 3:

system("cls");

zad4();

break;

case 4:

exit(1);

break;

default:

printf("Неверное число");

exit(1);

break;

}

}

}

void zad2() {

while (1) {

printf("1. Граф в матричной форме\n");

printf("2. Граф в виде списков смежности\n");

printf("Введите другое число, чтобы вернуться назад.\n");

scanf("%d", &cc);

switch (cc) {

case 1:

system("cls");

zad21();

break;

case 2:

system("cls");

zad22();

break;

default:

system("cls");

menu();

}

}

}

void zad21() {

while (1) {

printf("1. Отождествление вершин\n");

printf("2. Стягивание рёбер\n");

printf("3. Расщепление вершин\n");

printf("Введите другое число, чтобы вернуться назад.\n");

scanf("%d", &cc);

switch (cc) {

case 1:

system("cls");

zad211();

break;

case 2:

system("cls");

zad212();

break;

case 3:

system("cls");

zad213();

break;

default:

system("cls");

zad2();

}

}

}

void zad22() {

while (1) {

printf("1. Отождествление вершин\n");

printf("2. Стягивание рёбер\n");

printf("3. Расщепление вершин\n");

printf("Введите другое число, чтобы вернуться назад.\n");

scanf("%d", &cc);

switch (cc) {

case 1:

system("cls");

zad221();

break;

case 2:

system("cls");

zad222();

break;

case 3:

system("cls");

zad223();

break;

default:

system("cls");

zad2();

}

}

}

void zad211() {

zad1();

printf("\n");

printf("Введите вершину 1 (0-%d): ", n - 1);

scanf("%d", &ver1);

printf("Введите отождествляемую вершину (0-%d): ", n - 1);

scanf("%d", &ver2);

if (ver1 >= n || ver2 >= n) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Отождествление вершин %d и %d графа G1:\n\n", ver1, ver2);

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

if (A[ver2][i] == 1) {

A[ver1][i] = 1;

A[i][ver1] = 1;

}

A[ver2][i] = 0;

A[i][ver2] = 0;

}

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

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

if (i == ver2 || j == ver2) {

if (j == n - 1) printf("\n");

continue;

}

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

if (j == n - 1) printf("\n");

}

}

printf("\n");

printf("Введите вершину 1 (0-%d): ", m - 1);

scanf("%d", &ver1);

printf("Введите отождествляемую вершину (0-%d): ", m - 1);

scanf("%d", &ver2);

if (ver1 >= m || ver2 >= m) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Отождествление вершин %d и %d графа G2:\n\n", ver1, ver2);

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

if (B[ver2][i] == 1) {

B[ver1][i] = 1;

B[i][ver1] = 1;

}

B[ver2][i] = 0;

B[i][ver2] = 0;

}

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

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

if (i == ver2 || j == ver2) {

if (j == m - 1) printf("\n");

continue;

}

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

if (j == m - 1) printf("\n");

}

}

end();

}

void zad212() {

zad1();

printf("\n");

printf("Введите вершину 1 (0-%d): ", n - 1);

scanf("%d", &ver1);

printf("Введите вершину для стягивания (0-%d): ", n - 1);

scanf("%d", &ver2);

if (ver1 >= n || ver2 >= n) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Стягивание рёбер %d и %d графа G1:\n\n", ver1, ver2);

if (A[ver1][ver2] == 0) {

printf("Стягивание невозможно");

goto smert;

}

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

if (A[ver2][i] == 1 && A[ver1][ver2] != 0) {

A[ver1][i] = 1;

A[i][ver1] = 1;

}

A[ver2][i] = 0;

A[i][ver2] = 0;

A[i][i] = 0;

}

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

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

if (i == ver2 || j == ver2) {

if (j == n - 1) printf("\n");

continue;

}

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

if (j == n - 1) printf("\n");

}

}

smert:

printf("\n");

printf("Введите вершину 1 (0-%d): ", m - 1);

scanf("%d", &ver1);

printf("Введите вершину для стягивания (0-%d): ", m - 1);

scanf("%d", &ver2);

if (ver1 >= m || ver2 >= m) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Стягивание рёбер %d и %d графа G2:\n\n", ver1, ver2);

if (B[ver1][ver2] == 0) {

printf("Стягивание невозможно");

goto smert1;

}

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

if (B[ver2][i] == 1) {

B[ver1][i] = 1;

B[i][ver1] = 1;

}

B[ver2][i] = 0;

B[i][ver2] = 0;

B[i][i] = 0;

}

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

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

if (i == ver2 || j == ver2) {

if (j == m - 1) printf("\n");

continue;

}

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

if (j == m - 1) printf("\n");

}

}

smert1:

end();

}

void zad213() {

zad1();

printf("\n");

printf("Введите расщепляемую вершину (0-%d): ", n - 1);

scanf("%d", &ver1);

if (ver1 >= n) {

printf("Неверная вершина\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Расщепление вершины %d графа G1:\n\n", ver1);

A = (int\*\*)realloc(A, (n + 1) \* sizeof(int\*));

A[n] = (int\*)calloc(n + 1, sizeof(int));

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

A[i] = (int\*)realloc(A[i], (n + 1) \* sizeof(int));

}

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

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

A[i][n] = 0;

A[n][i] = 0;

}

}

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

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

if (A[i][ver1] == 1) {

A[i][n] = 1;

A[n][i] = 1;

}

A[ver1][n] = 1;

A[n][ver1] = 1;

A[i][i] = 0;

}

}

n++;

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

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

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

}

printf("\n");

}

printf("\n");

printf("Введите расщепляемую вершину (0-%d): ", m - 1);

scanf("%d", &ver1);

if (ver1 >= m) {

printf("Неверная вершина\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Расщепление вершины %d графа G2:\n\n", ver1);

B = (int\*\*)realloc(B, (m + 1) \* sizeof(int\*));

B[m] = (int\*)calloc(m + 1, sizeof(int));

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

B[i] = (int\*)realloc(B[i], (m + 1) \* sizeof(int));

}

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

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

B[i][m] = 0;

B[m][i] = 0;

}

}

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

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

if (B[i][ver1] == 1) {

B[i][m] = 1;

B[m][i] = 1;

}

B[ver1][m] = 1;

B[m][ver1] = 1;

B[i][i] = 0;

}

}

m++;

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

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

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

}

printf("\n");

}

while (1) {

printf("\n");

printf("Введите любое число, чтобы вернуться в меню.\n");

scanf("%d", &cc);

system("cls");

menu();

}

}

void zad221() {

zad1();

Node\*\* list1 = (Node\*\*)malloc(n \* sizeof(Node\*));

int i, j;

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

list1[i] = NULL;

}

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

for (j = 0; j < n; ++j) {

if (A[i][j] == 1) {

addver(list1, i, j);

}

}

}

printf("\n");

printf("Введите вершину 1 (0-%d): ", n - 1);

scanf("%d", &ver1);

printf("Введите отождествляемую вершину (0-%d): ", n - 1);

scanf("%d", &ver2);

if (ver1 >= n || ver2 >= n) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Отождествление вершин %d и %d графа G1:\n\n", ver1, ver2);

ListidenVer(list1, ver1, ver2);

Node\*\* list2 = (Node\*\*)malloc(m \* sizeof(Node\*));

int z, x;

for (z = 0; z < m; ++z) {

list2[z] = NULL;

}

for (z = 0; z < m; ++z) {

for (x = 0; x < m; ++x) {

if (B[z][x] == 1) {

addver(list2, z, x);

}

}

}

printf("\n");

printf("Введите вершину 1 (0-%d): ", m - 1);

scanf("%d", &ver1);

printf("Введите отождествляемую вершину (0-%d): ", m - 1);

scanf("%d", &ver2);

if (ver1 >= m || ver2 >= m) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Отождествление вершин %d и %d графа G2:\n\n", ver1, ver2);

ListidenVer(list2, ver1, ver2);

free(list1);

free(list2);

end();

}

void zad222() {

zad1();

Node\*\* list1 = (Node\*\*)malloc(n \* sizeof(Node\*));

int i, j;

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

list1[i] = NULL;

}

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

for (j = 0; j < n; ++j) {

if (A[i][j] == 1) {

addver(list1, i, j);

}

}

}

printf("\n");

printf("Введите вершину 1 (0-%d): ", n - 1);

scanf("%d", &ver1);

printf("Введите вершину для стягивания (0-%d): ", n - 1);

scanf("%d", &ver2);

if (ver1 >= n || ver2 >= n) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Стягивание рёбер %d и %d графа G1:\n\n", ver1, ver2);

ListcontEdge(list1, ver1, ver2);

Node\*\* list2 = (Node\*\*)malloc(m \* sizeof(Node\*));

int z, x;

for (z = 0; z < m; ++z) {

list2[z] = NULL;

}

for (z = 0; z < m; ++z) {

for (x = 0; x < m; ++x) {

if (B[z][x] == 1) {

addver(list2, z, x);

}

}

}

printf("\n");

printf("Введите вершину 1 (0-%d): ", m - 1);

scanf("%d", &ver1);

printf("Введите вершину для стягивания (0-%d): ", m - 1);

scanf("%d", &ver2);

if (ver1 >= m || ver2 >= m) {

printf("Неверные вершины\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Стягивание рёбер %d и %d графа G2:\n\n", ver1, ver2);

ListcontEdge(list2, ver1, ver2);

free(list1);

free(list2);

end();

}

void zad223() {

zad1();

Node\*\* list1 = (Node\*\*)malloc((n + 1) \* sizeof(Node\*));

int i, j;

for (i = 0; i < n+1; ++i) {

list1[i] = NULL;

}

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

for (j = 0; j < n; ++j) {

if (A[i][j] == 1) {

addver(list1, i, j);

}

}

}

printf("\n");

printf("Введите расщепляемую вершину (0-%d): ", n - 1);

scanf("%d", &ver1);

if (ver1 >= n) {

printf("Неверная вершина\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Расщепление вершины %d графа G1:\n\n", ver1);

addEdge(list1, n, ver1);

Node\* current = list1[ver1];

while (current != NULL) {

addEdge(list1, n, current->ver);

current = current->next;

}

n++;

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

Node\* curr = list1[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

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

Node\* temp = list1[i];

printf("Список смежности вершины %d ", i);

while (temp) {

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

Node\*\* list2 = (Node\*\*)malloc((m+1) \* sizeof(Node\*));

int z, x;

for (z = 0; z < m+1; ++z) {

list2[z] = NULL;

}

for (z = 0; z < m; ++z) {

for (x = 0; x < m; ++x) {

if (B[z][x] == 1) {

addver(list2, z, x);

}

}

}

printf("\n");

printf("Введите расщепляемую вершину (0-%d): ", m - 1);

scanf("%d", &ver1);

if (ver1 >= m) {

printf("Неверная вершина\n");

printf("Возврат в меню...\n");

Sleep(2000);

system("cls");

menu();

}

printf("Расщепление вершины %d графа G2:\n\n", ver1);

addEdge(list2, m, ver1);

current = list2[ver1];

while (current != NULL) {

addEdge(list2, m, current->ver);

current = current->next;

}

m++;

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

Node\* curr = list2[i];

while (curr) {

Node\* temp = curr;

while (temp->next) {

if (temp->next->ver == curr->ver) {

Node\* dupl = temp->next;

temp->next = temp->next->next;

free(dupl);

}

else {

temp = temp->next;

}

}

curr = curr->next;

}

}

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

Node\* temp = list2[i];

printf("Список смежности вершины %d ", i);

while (temp) {

printf(" -> %d ", temp->ver);

temp = temp->next;

}

printf("\n");

}

n--;

m--;

end();

}

void zad3() {

while (1) {

printf("1. Объединение графов\n");

printf("2. Пересечение графов\n");

printf("3. Кольцевая сумма графов\n");

printf("Введите другое число, чтобы вернуться назад.\n");

scanf("%d", &cc);

switch (cc) {

case 1:

system("cls");

zad31();

break;

case 2:

system("cls");

zad32();

break;

case 3:

system("cls");

zad33();

break;

default:

system("cls");

menu();

}

}

}

void zad31() {

zad1();

if (m > n)

n = m;

res = (int\*\*)calloc(n, sizeof(int\*));

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

res[i] = (int\*)calloc(n, sizeof(int));

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

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

res[i][j] = A[i][j] || B[i][j];

}

}

printf("\n");

printf("Объединение графов:\n");

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

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

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

}

printf("\n");

}

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

free(res[i]);

}

free(res);

end();

}

void zad32() {

zad1();

res = (int\*\*)calloc(n, sizeof(int\*));

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

res[i] = (int\*)calloc(n, sizeof(int));

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

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

res[i][j] = A[i][j] && B[i][j];

}

}

printf("\n");

printf("Пересечение графов:\n");

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

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

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

}

printf("\n");

}

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

free(res[i]);

}

free(res);

end();

}

void zad33() {

zad1();

res = (int\*\*)calloc(n, sizeof(int\*));

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

res[i] = (int\*)calloc(n, sizeof(int));

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

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

res[i][j] = A[i][j] ^ B[i][j];

}

}

printf("\n");

printf("Кольцевая сумма графов:\n");

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

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

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

}

printf("\n");

}

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

free(res[i]);

}

free(res);

end();

}

void zad4() {

zad1();

res = (int\*\*)calloc(n \* m, sizeof(int\*));

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

res[i] = (int\*)calloc(n \* m, sizeof(int));

printf("\n");

printf("Декартово произведение графов:\n");

for (int i = 0; i < m \* n; i++) {

for (int j = 0; j < m \* n; j++) {

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

}

printf("\n");

}

for (int i = 0; i < n \* m; i++) {

free(res[i]);

}

free(res);

end();

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

menu();

return 0;

}