Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе № 6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнили ст. гр. 22ВВС1:

Пилюгин А.Э

Приняли:

Акифьев И.В

Юрова О.В

ПЕНЗА 2023

**Цель работы**

Изучить теоретический материал: всевозможные унарные и бинарные операции над графами. Научиться реализовать разные алгоритмы на матрицах смежности, а также на списках смежности.

**Лабораторное задание**

**Задание 1**

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

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

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

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

**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

Результат выполнения операции выведите на экран.

**Задание 4 \***

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

Результат выполнения операции выведите на экран.

**Листинг программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

#include<string>

#include<cstring>

#include<time.h>

#include<Windows.h>

using namespace std;

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <algorithm>

struct List\_body {

int vertex;

struct List\_body\* next;

};

struct List\_head {

int number;

struct List\_body\* head;

struct List\_head\* next;

};

struct List\_head\* head\_list1 = NULL, \* head\_spisok2 = NULL;

string\* name;

bool name\_number = TRUE;

struct List\_body\* creation\_element(int number) {

struct List\_body\* p = NULL;

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

{

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

exit(1);

}

p->vertex = number;

p->next = NULL;

return p;

}

struct List\_head\* creation\_element\_head(int number, int\*\* M, int size) {

struct List\_head\* p = NULL;

struct List\_body\* q, \* save = NULL;

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

{

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

exit(1);

}

p->next = NULL;

p->number = number;

p->head = NULL;

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

if (M[number][i] == 1) {

q = creation\_element(i);

if (p->head == NULL) {

p->head = q;

save = q;

}

else {

save->next = q;

save = q;

}

}

}

return p;

}

void creation\_list(int\*\* M, struct List\_head\* list, int size, struct List\_head\*\* head\_spisok) {

struct List\_head\* h, \* save = NULL;

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

h = creation\_element\_head(i, M, size);

if ((\*head\_spisok) == NULL) {

(\*head\_spisok) = h;

save = h;

}

else {

save->next = h;

save = h;

}

}

}

void review(int size, struct List\_head\*\* head\_spisok) {

struct List\_body\* save;

struct List\_head\* save\_head = (\*head\_spisok);

cout << "\n";

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

save = save\_head->head;

cout << "[" << save\_head->number << "]";

while (save != NULL) {

cout << " -> " << save->vertex;

save = save->next;

}

cout << "\n";

save\_head = save\_head->next;

}

cout << "\n";

}

void output\_mas(int\*\* M, int size) {

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

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

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

}

cout << "\n";

}

}

void input\_output(int\*\* M, int size) {

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

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

if (i == j) {

// С вероятностью 50% создаем петлю

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

}

else {

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

M[j][i] = M[i][j];

}

}

}

output\_mas(M, size);

}

void rename(int size2) {

for (int i = 0; i < size2; i++) {

cout << "\nВведите название вершины номер " << i + 1 << ": ";

cin >> name[i];

}

name\_number = FALSE;

}

string perem;

int count\_on\_name(int size) {

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

if ((perem.compare(name[i])) == 0) {

return i;

}

}

}

int\*\* identification(int\*\* M, int\* size, bool input) {

int\*\* M3, vertex1, vertex2;

M3 = new int\* [(\*size) - 1]; // создаём двумерный массив

for (int i = 0; i < (\*size) - 1; i++) {

M3[i] = new int[(\*size) - 1];

}

if (input) {

cout << "Введите вершины для отождествления\n 1 вершина : ";

cin >> vertex1;

cout << " 2 вершина : ";

cin >> vertex2;

}

else {

cout << "Введите вершины инцидентные ребру, которое вы хотите стянуть\n 1 вершина : ";

cin >> vertex1;

cout << " 2 вершина : ";

cin >> vertex2;

}

// Проверяем наличие ребра между вершинами

if (M[vertex1][vertex2] == 1 || M[vertex2][vertex1] == 1) {

// Удаляем ребро между вершинами

M[vertex1][vertex2] = 0;

M[vertex2][vertex1] = 0;

// Объединяем вершины

for (int i = 0; i < (\*size) - 1; i++) {

for (int j = i; j < (\*size) - 1; j++) {

if (i == min(vertex1, vertex2) && j >= max(vertex1, vertex2)) {

M3[i][j] = (M[min(vertex1, vertex2)][j + 1] || M[max(vertex1, vertex2)][j + 1]);

M3[j][i] = M3[i][j];

if (i == min(vertex1, vertex2)) {

M3[i][i] = 1;

}

}

else if (i >= max(vertex1, vertex2) && j >= max(vertex1, vertex2)) {

M3[i][j] = M[i + 1][j + 1];

M3[j][i] = M3[i][j];

}

else if (i == min(vertex1, vertex2)) {

M3[i][j] = (M[min(vertex1, vertex2)][j] || M[max(vertex1, vertex2)][j]);

M3[j][i] = M3[i][j];

}

else if (j >= max(vertex1, vertex2)) {

M3[i][j] = M[i][j + 1];

M3[j][i] = M3[i][j];

}

else if (j == min(vertex1, vertex2)) {

M3[i][j] = M[i][min(vertex1, vertex2)] || M[i][max(vertex1, vertex2)];

M3[j][i] = M3[i][j];

}

else {

M3[i][j] = M[i][j];

M3[j][i] = M3[i][j];

}

}

}

output\_mas(M3, (\*size) - 1);

(\*size)--;

return M3;

}

else {

cout << "Невозможно стянуть - нет ребра между вершинами.\n";

return M; // или другие действия по вашему выбору

}

}

void check\_of\_vertexes(int\* vertex, int size) {

while ((\*vertex) >= size || (\*vertex) < 0) {

cout << "Ошибка(отсутствие таковой вершины)\n Попробуйте ещё раз: ";

cin >> (\*vertex);

}

}

struct List\_head\* search\_of\_vertex(struct List\_head\* save, int vertex) {

while (save->number != vertex) {

save = save->next;

}

return save;

}

void identification\_spisok(struct List\_head\*\* head\_spisok, bool input, int\* size) {

int vertex1, vertex2;

struct List\_head\* save1 = NULL, \* save2 = NULL;

struct List\_body\* save1\_soed = NULL, \* save2\_soed = NULL, \* save2\_1\_soed = NULL;

if (input) {

cout << "Введите вершины для отождествления\n 1 вершина: ";

cin >> vertex1;

check\_of\_vertexes(&vertex1, \*size);

cout << " 2 вершина: ";

cin >> vertex2;

check\_of\_vertexes(&vertex2, \*size);

}

else {

cout << "Введите вершины инцидентные ребру ,которое вы хотите стянуть\n 1 вершина: ";

cin >> vertex1;

check\_of\_vertexes(&vertex1, \*size);

cout << " 2 вершина: ";

cin >> vertex2;

check\_of\_vertexes(&vertex2, \*size);

}

save1 = search\_of\_vertex(\*head\_spisok, vertex1);

save2 = search\_of\_vertex(\*head\_spisok, vertex2);

save1\_soed = save1->head;

save2\_soed = save2->head;

if (save1->head == NULL) {

save1->head = save2->head;

}

else {

while (save2\_soed != NULL && save2->head != NULL) {

save1\_soed = save1->head;

if (save2\_soed != NULL && save1\_soed != NULL && save1\_soed->vertex > save2\_soed->vertex) {

save2\_1\_soed = save2\_soed;

save2\_soed = save2\_soed->next;

save2\_1\_soed->next = save1->head;

save1->head = save2\_1\_soed;

}

else {

while ((save1\_soed->next->vertex != save2\_soed->vertex) && (save1\_soed->next != NULL) && (save1\_soed->next->vertex < save2\_soed->vertex)) {

save1\_soed = save1\_soed->next;

}

if (save1\_soed->next->vertex == save2\_soed->vertex) {

save2\_soed = save2\_soed->next;

}

else if (save1\_soed->next == NULL) {

save2\_1\_soed = save2\_soed;

save2\_soed = save2\_soed->next;

save1\_soed->next = save2\_1\_soed;

save2\_1\_soed->next = NULL;

}

else {

save2\_1\_soed = save2\_soed;

save2\_soed = save2\_soed->next;

save2\_1\_soed->next = save1\_soed->next;

save1\_soed->next = save2\_1\_soed;

}

}

}

}

save2 = search\_of\_vertex(\*head\_spisok, vertex2 - 1);

save2->next = save2->next->next;

(\*size)--;

}

int\*\* splitting(int\*\* M, int\* size) {

int\*\* M3, vertex;

M3 = new int\* [\*size + 1]; // создаём двумерный массив

for (int i = 0; i < \*size + 1; i++) {

M3[i] = new int[\*size + 1];

}

if (name\_number) {

cout << "Введите вершину для расщипления : ";

cin >> vertex;

}

else {

cout << "Введите вершину для расщипления : ";

cin >> perem;

vertex = count\_on\_name(\*size);

}

// Первая строка равна первому столбцу

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

M3[0][i + 1] = M[i][vertex];

M3[i + 1][0] = M[vertex][i];

}

// Заполнение остальной части матрицы

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

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

M3[i + 1][j + 1] = M[i][j];

M3[i][j] = 0;

//M3[0][0] = 0;

}

}

if (M[vertex][vertex] == 0) {

cout << "Невозможно стянуть\n";

return M; // или другие действия по вашему выбору

}

output\_mas(M3, \*size + 1);

(\*size)++;

return M3;

}

void splitting\_list(struct List\_head\*\* head\_spisok, int\* size, int\*\* M) {

int vertex, colvo\_reber = 0;

struct List\_head\* save1 = NULL, \* q = NULL;

struct List\_body\* save1\_soed = NULL, \* save2\_soed = NULL;

cout << "Введите вершину для расщипления : ";

cin >> vertex;

check\_of\_vertexes(&vertex, \*size);

save1 = search\_of\_vertex(\*head\_spisok, vertex);

save1\_soed = save1->head;

while (save1\_soed != NULL) {

save1\_soed = save1\_soed->next;

colvo\_reber++;

}

save1\_soed = save1->head;

for (int i = 0; i < ((colvo\_reber / 2) - 1); i++) {

save1\_soed = save1\_soed->next;

}

save2\_soed = save1\_soed->next;

save1\_soed->next = NULL;

q = creation\_element\_head(\*size, M, 0);

q->head = save2\_soed;

save1 = search\_of\_vertex(\*head\_spisok, (\*size) - 1);

save1->next = q;

(\*size)++;

}

void Union(int\*\* M1, int\*\* M2, int size1, int size2) {

int\*\* M3, size;

size = max(size1, size2);

M3 = new int\* [size];

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

M3[i] = new int[size];

}

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

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

if (i < min(size1, size2) && j < min(size1, size2)) {

M3[i][j] = M3[j][i] = (M1[i][j] | M2[i][j]);

}

else if (size1 < size2) {

M3[i][j] = M3[j][i] = M2[i][j];

}

else { M3[i][j] = M3[j][i] = M1[i][j]; }

}

}

output\_mas(M3, size);

}

void intersection(int\*\* M1, int\*\* M2, int size1, int size2) {

int\*\* M3, size;

size = min(size1, size2);

M3 = new int\* [size];

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

M3[i] = new int[size];

}

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

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

if (i < min(size1, size2) && j < min(size1, size2)) {

M3[i][j] = M3[j][i] = (M1[i][j] & M2[i][j]);

}

else {

M3[i][j] = M3[j][i] = 0;

}

}

}

output\_mas(M3, size);

}

void circle\_sum(int\*\* M1, int\*\* M2, int size1, int size2) {

int\*\* M3, size, \* colvo\_reber, otstup;

size = max(size1, size2);

M3 = new int\* [size];

colvo\_reber = new int[size];

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

M3[i] = new int[size];

colvo\_reber[i] = 0;

}

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

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

if (i < min(size1, size2) && j < min(size1, size2)) {

if (M1[i][j] == M2[i][j]) {

M3[i][j] = M3[j][i] = 0;

}

else { M3[i][j] = M3[j][i] = 1; }

}

else if (size1 < size2) {

M3[i][j] = M3[j][i] = M2[i][j];

}

else { M3[i][j] = M3[j][i] = M1[i][j]; }

}

}

output\_mas(M3, size);

size1 = 0;

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

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

if (M3[i][j] == 1) {

colvo\_reber[i]++;

}

}

if (colvo\_reber[i] > 0) {

size1++;

}

}

M1 = new int\* [size1];

for (int i = 0; i < size1; i++) {

M1[i] = new int[size1];

}

for (int i = 0; i < size1; i++) {

otstup = 0;

for (int j = 0; j < size1; j++) {

if (colvo\_reber[j] = 0) { otstup++; }

M1[i][j] = M3[i + otstup][j + otstup];

}

}

}

int\*\* dekartovo\_proizvedenie(int\*\* M1, int size1, int\*\* M2, int size2) {

int n1 = size1;

int n2 = size2;

int productSize = n1 \* n2;

int\*\* productGraph = new int\* [productSize + 1];

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

productGraph[i] = new int[productSize + 1];

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

productGraph[i][j] = 0;

}

}

for (int i1 = 0; i1 < n1; ++i1) {

for (int i2 = 0; i2 < n2; ++i2) {

for (int j1 = 0; j1 < n1; ++j1) {

for (int j2 = 0; j2 < n2; ++j2) {

int v1 = i1 \* n2 + i2 + 1;

int v2 = j1 \* n2 + j2 + 1;

if (i1 == j1 && M2[i2][j2]) {

productGraph[v1][v2] = 1;

}

if (i2 == j2 && M1[i1][j1]) {

productGraph[v1][v2] = 1;

}

}

}

}

}

output\_mas(productGraph, n1 \* n2);

return productGraph;

}

// Ваша функция main должна вызывать эту новую функцию для операции декартова произведения

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int\*\* M1, \*\* M2, size1, size2, choose1, choose2;

struct List\_head\* spisok1, \* spisok2;

bool condition = TRUE;

cout << " Матрицы смежности \n \n Введите количество вершин графа №1: ";

cin >> size1;

cout << " Введите количество вершин графа №2: ";

cin >> size2;

M1 = new int\* [size1]; // создаём двумерный массив

M2 = new int\* [size2];

name = new string[size2];

spisok1 = new struct List\_head[size1];

spisok2 = new struct List\_head[size2];

for (int i = 0; i < size1; i++) {

M1[i] = new int[size1];

}

for (int i = 0; i < size2; i++) {

M2[i] = new int[size2];

}

cout << "M1:\n";

input\_output(M1, size1);

cout << "\nM2:\n";

input\_output(M2, size2);

cout << "\n";

cout << "Выберите действие\n 1) Матрицы\n 2) Списки\n Выберите действие: ";

cin >> choose2;

switch (choose2) {

case(1):

while (condition) {

cout << "\n 1) Отождествление вершин\n 2) Стягивание ребра\n 3) Расщипление вершины\n 4) Объединение\n 5) Пересечение\n 6) Кольцевая сумма\n 7) Декартово произведение \n 8) Закончить\n Выберите действие: ";

cin >> choose1;

switch (choose1) {

case(1):

cout << "\n";

M2 = identification(M2, &size2, TRUE);

break;

case(2):

cout << "\n";

M2 = identification(M2, &size2, FALSE);

break;

case(3):

cout << "\n";

M2 = splitting(M2, &size2);

break;

case(4):

cout << "\n";

Union(M1, M2, size1, size2);

break;

case(5):

cout << "\n";

intersection(M1, M2, size1, size2);

break;

case(6):

cout << "\n";

circle\_sum(M1, M2, size1, size2);

break;

case(7):

cout << "\n";

dekartovo\_proizvedenie(M1, size1, M2, size2);

break;

case(8):

condition = FALSE;

break;

default:

cout << "Ошибка";

}

}

break;

case(2):

creation\_list(M1, spisok1, size1, &head\_list1);

review(size1, &head\_list1);

creation\_list(M2, spisok2, size2, &head\_spisok2);

review(size2, &head\_spisok2);

while (condition) {

cout << "\n 1) Отождествление вершин\n 2) Стягивание ребра\n 3) Расщипление вершины\n 4) Завершить\n Выберите действие: ";

cin >> choose1;

switch (choose1) {

case(1):

identification\_spisok(&head\_list1, TRUE, &size1);

review(size1, &head\_list1);

break;

case(2):

identification\_spisok(&head\_list1, FALSE, &size1);

review(size1, &head\_list1);

break;

case(3):

splitting\_list(&head\_list1, &size1, M1);

review(size1, &head\_list1);

break;

case(4):

condition = FALSE;

break;

default:

cout << "Ошибка";

}

}

break;

case(3):

break;

default:

cout << "Ошибка";

}

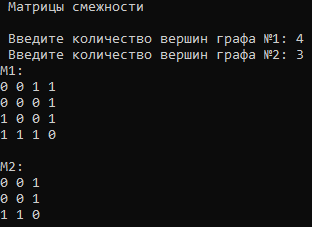
system("pause");

return 0;

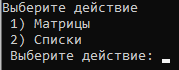
}

**Результат работы программы.**

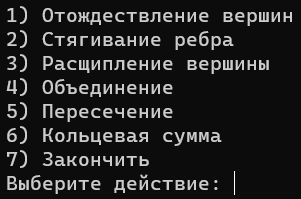
* 1. Сгенерировали две матрицы смежности. Заполнили их и вывели на экран.



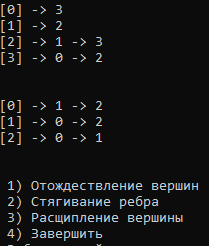
* 1. Делаем выбор: работа с матрицами или со списками.



Работа с матрицами:

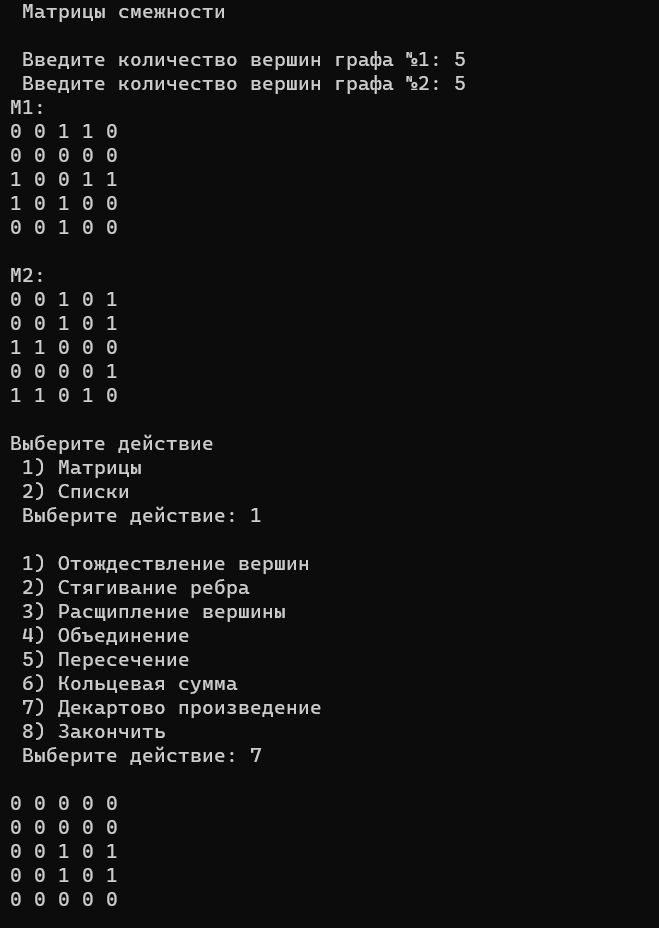


Работа со списками:



* 1. Выбираем вариант и выполняем желаемое действие.

**Задание 4:**

****

**Вывод**

В ходе выполнения лабораторной работы научились выполнять разные унарные и бинарные операции над списками и матрицами смежности. Реализовали разные алгоритмы, а также добавили функцию переименования вершин, для использования в операциях.