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. 22ВВВ2:

Юртаев Д. Ю.

Шатин Д. Д.

Харитонов А. А.

Приняли:

Митрохин М.А.

Акифьев И.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.

**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;

struct spisok\_soed {

int vershina;

struct spisok\_soed\* next;

};

struct spisok\_head {

int nomber;

struct spisok\_soed\* head;

struct spisok\_head\* next;

};

struct spisok\_head\* head\_spisok1 = NULL, \* head\_spisok2 = NULL;

string\* name;

bool name\_nomber = TRUE;

struct spisok\_soed\* creation\_element(int nomber) {

struct spisok\_soed\* p = NULL;

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

{

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

exit(1);

}

p->vershina = nomber;

p->next = NULL;

return p;

}

struct spisok\_head\* creation\_element\_head(int nomber, int\*\* M, int size) {

struct spisok\_head\* p = NULL;

struct spisok\_soed\* q, \* save = NULL;

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

{

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

exit(1);

}

p->next = NULL;

p->nomber = nomber;

p->head = NULL;

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

if (M[nomber][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\_spisok(int\*\* M, struct spisok\_head\* spisok, int size, struct spisok\_head\*\* head\_spisok) {

struct spisok\_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 spisok\_head\*\* head\_spisok) {

struct spisok\_soed\* save;

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

cout << "\n";

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

save = save\_head->head;

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

while (save != NULL) {

cout << " -> " << save->vershina;

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 = i; j < size; j++) {

if (i == j) { M[i][j] = 0; }

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\_nomber = FALSE;

}

string perem;

int shet\_po\_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, vershina1, vershina2;

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

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

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

}

if (name\_nomber) {

if (input) {

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

cin >> vershina1;

cout << " 2 вершина : ";

cin >> vershina2;

}

else {

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

cin >> vershina1;

cout << " 2 вершина : ";

cin >> vershina2;

}

}

else {

if (input) {

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

cin >> perem;

vershina1 = shet\_po\_name(\*size);

cout << " 2 вершина : ";

cin >> perem;

vershina2 = shet\_po\_name(\*size);

}

else {

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

cin >> perem;

vershina1 = shet\_po\_name(\*size);

cout << " 2 вершина : ";

cin >> perem;

vershina2 = shet\_po\_name(\*size);

}

}

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

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

if (i == min(vershina1, vershina2) && j >= max(vershina1, vershina2)) {

M3[i][j] = (M[min(vershina1, vershina2)][j + 1] || M[max(vershina1, vershina2)][j + 1]);

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

}

else

if (i >= max(vershina1, vershina2) && j >= max(vershina1, vershina2)) {

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

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

}

else if (i == min(vershina1, vershina2)) {

M3[i][j] = (M[min(vershina1, vershina2)][j] || M[max(vershina1, vershina2)][j]);

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

}

else if (j >= max(vershina1, vershina2)) {

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

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

}

else if (j == min(vershina1, vershina2)) {

M3[i][j] = M[i][min(vershina1, vershina2)] || M[i][max(vershina1, vershina2)];

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;

}

void proverka\_vershin(int\* vershina, int size) {

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

cout << "Такой вершины нет\n Попробуйте ещё раз: ";

cin >> (\*vershina);

}

}

struct spisok\_head\* poisk\_vershini(struct spisok\_head\* save, int vershina) {

while (save->nomber != vershina) {

save = save->next;

}

return save;

}

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

int vershina1, vershina2;

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

struct spisok\_soed\* save1\_soed = NULL, \* save2\_soed = NULL, \* save2\_1\_soed = NULL;

if (input) {

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

cin >> vershina1;

proverka\_vershin(&vershina1, \*size);

cout << " 2 вершина: ";

cin >> vershina2;

proverka\_vershin(&vershina2, \*size);

}

else {

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

cin >> vershina1;

proverka\_vershin(&vershina1, \*size);

cout << " 2 вершина: ";

cin >> vershina2;

proverka\_vershin(&vershina2, \*size);

}

save1 = poisk\_vershini(\*head\_spisok, vershina1);

save2 = poisk\_vershini(\*head\_spisok, vershina2);

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->vershina > save2\_soed->vershina) {

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

save1\_soed = save1\_soed->next;

}

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

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 = poisk\_vershini(\*head\_spisok, vershina2 - 1);

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

(\*size)--;

}

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

int\*\* M3, vershina, colvo\_connect = 0, counter = 0;

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

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

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

}

if (name\_nomber) {

cout << "Введите вершину для расшипления : ";

cin >> vershina;

}

else {

cout << "Введите вершину для расшипления : ";

cin >> perem;

vershina = shet\_po\_name(\*size);

}

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

if (M[vershina][i] == 1) { colvo\_connect++; }

}

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

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

if (j == vershina)

{

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

if (counter < (colvo\_connect / 2)) {

M3[i][j] = 1;

M3[i][\*size] = M3[\*size][i] = 0;

counter++;

}

else {

M3[i][j] = 0;

M3[i][\*size] = M3[\*size][i] = 1;

}

}

else {

M3[i][j] = 0;

M3[i][\*size] = M3[\*size][i] = 0;

}

}

else {

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

}

}

}

M3[vershina][\*size] = M3[\*size][vershina] = 1;

M3[\*size][\*size] = 0;

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

(\*size)++;

return M3;

}

void rashiplenie\_spisok(struct spisok\_head\*\* head\_spisok, int\* size, int\*\* M) {

int vershina, colvo\_reber = 0;

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

struct spisok\_soed\* save1\_soed = NULL, \* save2\_soed = NULL;

cout << "Введите вершину для расшипления : ";

cin >> vershina;

proverka\_vershin(&vershina, \*size);

save1 = poisk\_vershini(\*head\_spisok, vershina);

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 = poisk\_vershini(\*head\_spisok, (\*size) - 1);

save1->next = q;

(\*size)++;

}

void obedinen(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 peresechenie(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 main() {

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int\*\* M1, \*\* M2, size1, size2, vibor1, vibor2;

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

bool uslovie = 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 spisok\_head[size1];

spisok2 = new struct spisok\_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 >> vibor2;

switch (vibor2) {

case(1):

while (uslovie) {

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) Закончить\n Выберите действие: ";

cin >> vibor1;

switch (vibor1) {

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 = rashiplenie(M2, &size2);

break;

case(4):

cout << "\n";

obedinen(M1, M2, size1, size2);

break;

case(5):

cout << "\n";

peresechenie(M1, M2, size1, size2);

break;

case(6):

cout << "\n";

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

break;

case(7):

uslovie = FALSE;

break;

default:

cout << "Ошибка";

}

}

break;

case(2):

creation\_spisok(M1, spisok1, size1, &head\_spisok1);

review(size1, &head\_spisok1);

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

review(size2, &head\_spisok2);

while (uslovie) {

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

cin >> vibor1;

switch (vibor1) {

case(1):

identification\_spisok(&head\_spisok1, TRUE, &size1);

review(size1, &head\_spisok1);

break;

case(2):

identification\_spisok(&head\_spisok1, FALSE, &size1);

review(size1, &head\_spisok1);

break;

case(3):

rashiplenie\_spisok(&head\_spisok1, &size1, M1);

review(size1, &head\_spisok1);

break;

case(4):

uslovie = FALSE;

break;

default:

cout << "Ошибка";

}

}

break;

case(3):

break;

default:

cout << "Ошибка";

}

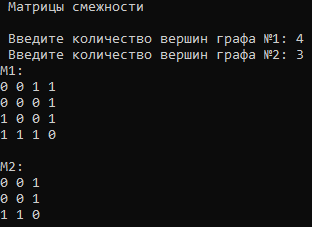
system("pause");

return 0;

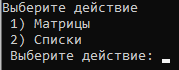
}

**Результат работы программы.**

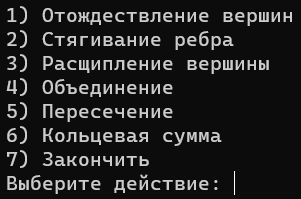
* 1. Сгенерировали две матрицы смежности. Заполнили их и вывели на экран.



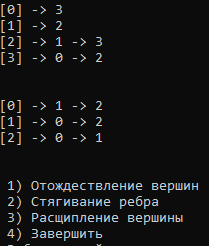
* 1. Делаем выбор: работа с матрицами или со списками.



Работа с матрицами:



Работа со списками:



* 1. Выбираем вариант и выполняем желаемое действие.

**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.