Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнил:

 студент группы 20ВВ2

Лазутин Д.Д.

 Принял:

 к.т.н., доцент Юрова О.В.

Пенза 2020

### **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.

*Примечание: задания 1 и 2 выполнены в одной программе.*

**Описание метода решения:**

Объявляем переменные: матрицу G, её размер S, счётчики i и j, будущие вводимые номера вершин t\_1 и t\_2.

Вводим размер матрицы смежности, она заполняется и выводятся на экран. Далее вводим вид операции, который мы хотим выполнить. Можно выбрать: удаление вершины, удаление ребра, отождествление вершин или стягивание ребра. Чтобы операция совершилась, вводим необходимые номер(а) вершины (2 вершин) графа. Также проверяется, существуют ли введённые номера вершин.

Каждая из 4 операций реализована соответствующей функцией, которая в дальнейшем вызывается.

**Листинг** LZ\_3\_Unar.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

int Equal\_V(int \*\*M, int v\_1, int v\_2, int Size)

{

int i, j, min, max;

if (v\_1 < v\_2) {min=v\_1-1; max=v\_2-1;}

else {min=v\_2-1; max=v\_1-1;}

//Процесс отождествления вершин

//Вершина min получает доп. рёбра от вершины max по принципу Лог. ИЛИ

for(i=min, j=0;j<Size;j++)

{

if (M[min][j]+M[max][j]>=1)

M[min][j]=1;

}

for(j=min, i=0;i<Size;i++)

{

if (M[i][min]+M[i][max]>=1)

M[i][min]=1;

}

//Удаление вершины max

for(i=0;i<Size-1;i++)

{

for(j=max;j<Size-1;j++)

{

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

}

}

for(j=0;j<Size-1;j++)

{

for(i=max;i<Size-1;i++)

{

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

}

}

for(i=0;i<Size-1;i++)

{

if (i==min)

M[i][i]=1;

else

M[i][i]=0;

}

//Вывод итоговой матрицы

for(i=0;i<Size-1;i++)

{

if(i==min)

printf("\n%2d/%2d|", min+1, max+1);

else

{

if(i>=max)

printf("\n%5d|", i+2);

else

printf("\n%5d|", i+1);

}

for(j=0;j<Size-1;j++)

{

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

printf("%2d", M[i][j]);

}

}

return(Size-1);

}

int Rib\_S(int \*\*M, int v\_1, int v\_2, int Size)

{

int i, j, min, max;

if (v\_1 < v\_2) {min=v\_1-1; max=v\_2-1;}

else {min=v\_2-1; max=v\_1-1;}

//Процесс отождествления вершин

//Вершина min получает доп. рёбра от вершины max по принципу Лог. ИЛИ

for(i=min, j=0;j<Size;j++)

{

if (M[min][j]+M[max][j]>=1)

M[min][j]=1;

}

for(j=min, i=0;i<Size;i++)

{

if (M[i][min]+M[i][max]>=1)

M[i][min]=1;

}

//Удаление вершины max и "стерилизация" всех вершин от петель

for(i=0;i<Size-1;i++)

{

for(j=max;j<Size-1;j++)

{

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

}

}

for(j=0;j<Size-1;j++)

{

for(i=max;i<Size-1;i++)

{

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

}

}

for(i=0;i<Size-1;i++)

M[i][i]=0;

//Вывод итогой матрицы

for(i=0;i<Size-1;i++)

{

if(i==min)

printf("\n%2d.%2d|", min+1, max+1);

else

{

if(i>=max)

printf("\n%5d|", i+2);

else

printf("\n%5d|", i+1);

}

for(j=0;j<Size-1;j++)

{

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

printf("%2d", M[i][j]);

}

}

return(Size-1);

}

int Del\_V(int \*\*M, int v, int Size)

{

int i=0, j=0;

//Удаление вершины v

for(i=0;i<Size-1;i++)

{

for(j=v-1;j<Size-1;j++)

{

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

}

}

for(j=0;j<Size-1;j++)

{

for(i=v-1;i<Size-1;i++)

{

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

}

}

//Вывод итоговой матрицы

for(i=0;i<Size-1;i++)

M[i][i]=0;

for(i=0;i<Size-1;i++)

{

if(i>=v-1)

printf("\n%4d|", i+2);

else

printf("\n%4d|", i+1);

for(j=0;j<Size-1;j++)

{

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

printf("%2d", M[i][j]);

}

}

return(Size-1);

}

void Del\_Rib(int \*\*M, int v\_1, int v\_2, int Size)

{

int i=v\_1-1, j=v\_2-1;

M[i][j]=0;M[j][i]=0;

//Удаление ребра между вершин v\_1 и v\_2

for(i=0;i<Size;i++)

{

printf("\n%4d|", i+1);

for(j=0;j<Size;j++)

{

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

printf("%2d", M[i][j]);

}

}

}

void main()

{

int \*\*G=NULL, i=0, j=0, t\_1=0, t\_2=0, S=0;

int Type;

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Введите размерность матрицы смежности: ");

scanf("%d", &S);

while(S<=0)

{

printf("Некорректно введённое число! Введите целое положительное число: ");

scanf("%d", &S);

}

G=(int\*\*)malloc(S\*sizeof(int));

for(i=0;i<S;i++)

{

G[i]=(int\*)malloc(S\*sizeof(int));

for(j=0;j<S;j++)

{

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

if(G[i][i]!=0)

G[i][j]=0;

}

}

for(i=0;i<S;i++)

{

printf("\n%4d|", i+1);

for(j=0;j<S;j++)

{

G[j][i]=G[i][j];

printf("%2d", G[i][j]);

}

}

printf("\nУкажите вид операции: 1-Отождествление вершин, 2-Стягивание ребра, 3-Удаление вершины, 4-Удаление ребра\n");

scanf("%d", &Type);

while(Type<1 || Type>4)

{

printf("Некорректно введённой номер вида операции! Введите число от 1 до 4.\n");

scanf("%d", &Type);

}

switch(Type)

{

case(1):

{

printf("Какие вершины нужно отождествить?\n");

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

while (t\_1==t\_2)

{

printf("Введён номер одной и той же вершины! Введите номера разных вершин.\n");

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

}

while((t\_1<1 || t\_1>S)||(t\_2<1 || t\_2>S))

{

printf("Введён номер несуществующей вершины! Введите числа в пределах размера матрицы смежности.\n");

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

}

S=Equal\_V(G, t\_1, t\_2,S);

break;

}

case(2):

{

printf("Ребро между какими вершинами нужно стянуть?\n");

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

while (t\_1==t\_2)

{

printf("Введён номер одной и той же вершины! Введите номера разных вершин.\n");

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

}

while((t\_1<1 || t\_1>S)||(t\_2<1 || t\_2>S))

{

printf("Введён номер несуществующей вершины! Введите числа в пределах размера матрицы смежности.\n");

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

}

S=Rib\_S(G, t\_1, t\_2,S);

break;

}

case(3):

{

printf("Какую вершину нужно удалить? ");scanf("%d", &t\_1);

while(t\_1<1 || t\_1>S)

{

printf("Такой вершины не существует! Введите число в пределах размера матрицы смежности.\n");

scanf("%d", &t\_1);

}

S=Del\_V(G, t\_1, S);

break;

}

case(4):

{

printf("Ребро между какими вершинами нужно удалить?\n");scanf("%d%d", &t\_1, &t\_2);

while (t\_1==t\_2)

{

printf("Введён номер одной и той же вершины! Введите номера разных вершин.\n");

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

}

while((t\_1<1 || t\_1>S)||(t\_2<1 || t\_2>S))

{

printf("Введён номер несуществующей вершины! Введите числа в пределах размера матрицы смежности.\n");

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

}

Del\_Rib(G, t\_1, t\_2, S);

break;

}

}

printf("\n");

free(G);

free(G);

}

**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 \*** (В разработке)

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

Результат выполнения операции выведите на экран.

*Примечание: задания 1 и 3 выполнены в одной программе.*

**Описание метода решения:**

Объявлены переменные: используемые матрицы G\_1 и G\_2 и итоговая матрица G, размеры 1-й и 2-й матрицы S\_1 и S\_2 соответственно, счётчики i и j, тип бинарной операции Type.  
Создаём, заполняем и выводим на экран матрицы G\_1 и G\_2. Вводим номер одной из предлагаемых операций. Далее вызывается функция Bin\_Op, которая в зависимости от выбранного номера операции совершает объединение, пересечение или кольцевую сумму графов, выраженных через G\_1 и G\_2. Результатом будет матрица G, которая будет выведена на экран.

**Листинг** LZ\_3\_Binar.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

void Bin\_Op(int \*\*M\_1, int \*\*M\_2, int \*\*M, int Size\_1, int Size\_2, int Type)

{

int i, j, min, max;

if(Size\_1<Size\_2) {min=Size\_1; max=Size\_2;}

else {min=Size\_2; max=Size\_1;}

M=(int\*\*)malloc(max\*sizeof(int));

for(i=0;i<max;i++)

{

\*(M+i)=(int\*)malloc(max\*sizeof(int));

for(j=0;j<max;j++)

{

if(max==Size\_1)

M[i][j]=M\_1[i][j];

else

M[i][j]=M\_2[i][j];

}

}

switch(Type)

{

case(1):

{

for(i=0;i<min;i++)

for(j=0;j<min;j++)

if((M\_1[i][j]==0)&&(M\_2[i][j]==0))

M[i][j]=0;

else M[i][j]=1;

break;

}

case(2):

{

for(i=0;i<min;i++)

for(j=0;j<min;j++)

if((M\_1[i][j]==1)&&(M\_2[i][j]==1))

M[i][j]=1;

else M[i][j]=0;

break;

}

case(3):

{

for(i=0;i<min;i++)

for(j=0;j<min;j++)

if((M\_1[i][j]!=M\_2[i][j]))

M[i][j]=1;

else M[i][j]=0;

break;

}

}

for(i=0;i<max;i++)

{

printf("\n%4d|", i+1);

for(j=0;j<max;j++)

{

printf("%2d", M[i][j]);

}

}

printf("\n");

free(M);

}

void main()

{

int \*\*G\_1=NULL, \*\*G\_2=NULL, \*\*G=NULL, i=0, j=0, S\_1=0, S\_2=0;

int Type=0;

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Введите размерности матриц смежности:\n");

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

while(S\_1<1 || S\_2<1)

{

printf("Некорректно введённое число! Введите целые положительные числа для указания размеров матриц:\n");

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

}

printf("\nГраф G\_1:");

G\_1=(int\*\*)malloc(S\_1\*sizeof(int));

for(i=0;i<S\_1;i++)

{

G\_1[i]=(int\*)malloc(S\_1\*sizeof(int));

for(j=0;j<S\_1;j++)

{

G\_1[i][j]=rand()%2;

if(G\_1[i][i]!=0)

G\_1[i][j]=0;

}

}

for(i=0;i<S\_1;i++)

{

printf("\n%4d|", i+1);

for(j=0;j<S\_1;j++)

{

G\_1[j][i]=G\_1[i][j];

printf("%2d", G\_1[i][j]);

}

}

printf("\n");

printf("\nГраф G\_2:");

G\_2=(int\*\*)malloc(S\_2\*sizeof(int));

for(i=0;i<S\_2;i++)

{

G\_2[i]=(int\*)malloc(S\_2\*sizeof(int));

for(j=0;j<S\_2;j++)

{

G\_2[i][j]=rand()%2;

if(G\_2[i][i]!=0)

G\_2[i][j]=0;

}

}

for(i=0;i<S\_2;i++)

{

printf("\n%4d|", i+1);

for(j=0;j<S\_2;j++)

{

G\_2[j][i]=G\_2[i][j];

printf("%2d", G\_2[i][j]);

}

}

printf("\nУкажите вид операции: 1-Объединение, 2-Пересечение, 3-Кольцевая сумма\n");

scanf("%d", &Type);

while(Type<1 || Type>3)

{

printf("Некорректно введённой номер вида операции! Введите число от 1 до 3.\n");

scanf("%d", &Type);

}

printf("\nИтоговый граф G:");

Bin\_Op(G\_1, G\_2, G, S\_1, S\_2, Type);

free(G\_2);

free(G\_1);

}

**Результаты работы программ**

Результаты работы программы для 1+2-го заданий представлены на рис.1-4.

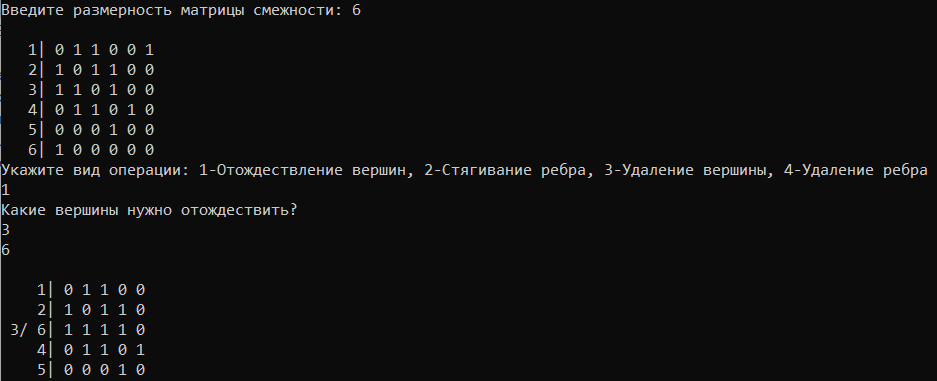


Рис.1.

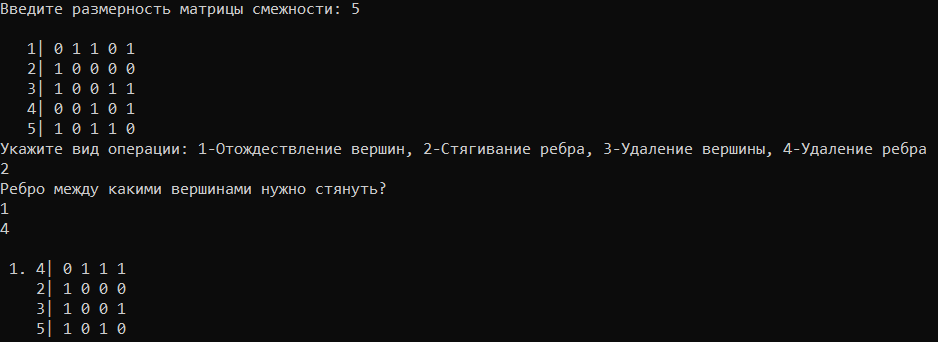


Рис.2.

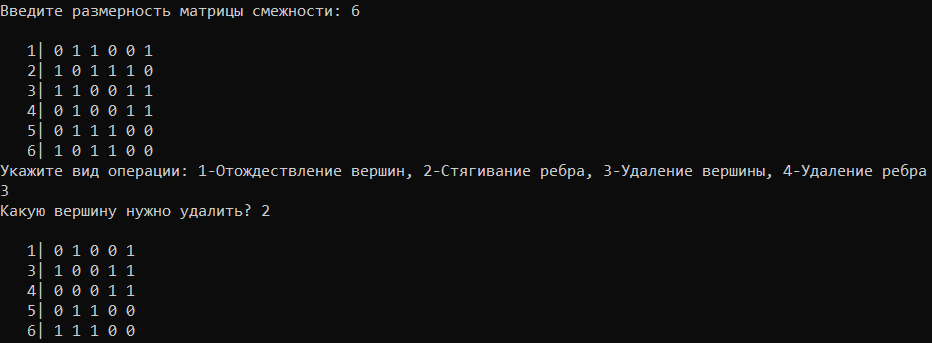


Рис.3.

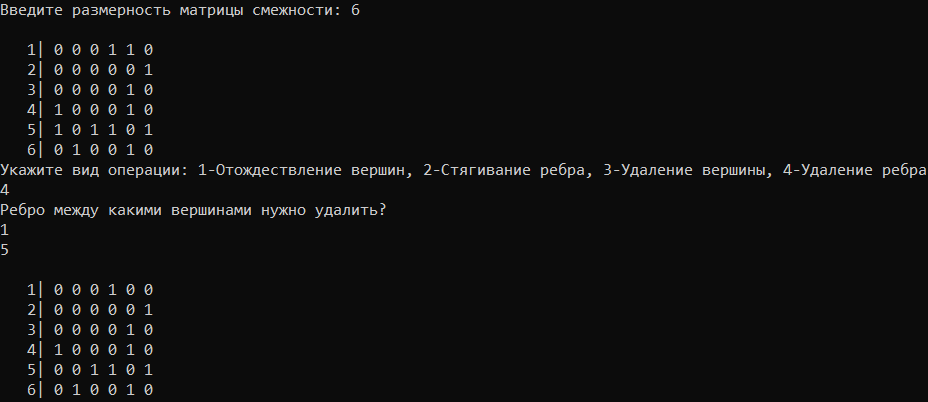


Рис.4.

Результат работы программы для 1+3-го задания представлен на рис.5-7.

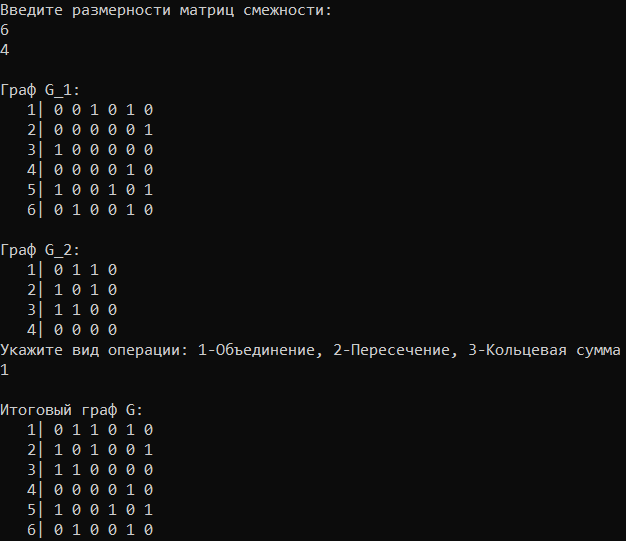


Рис.5.

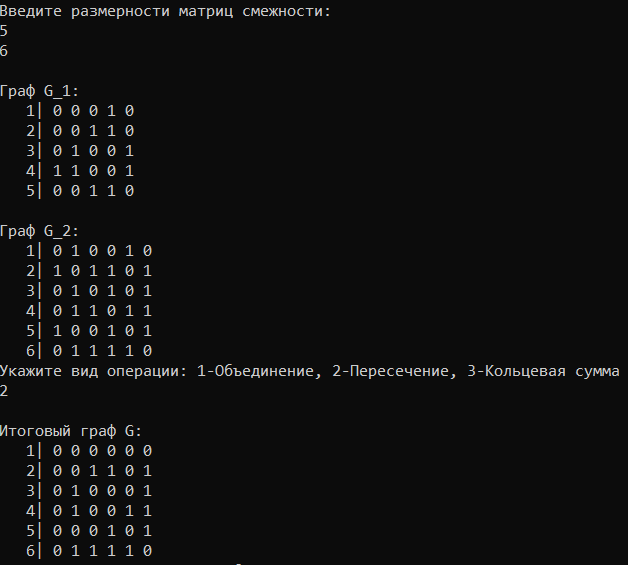


Рис.6.

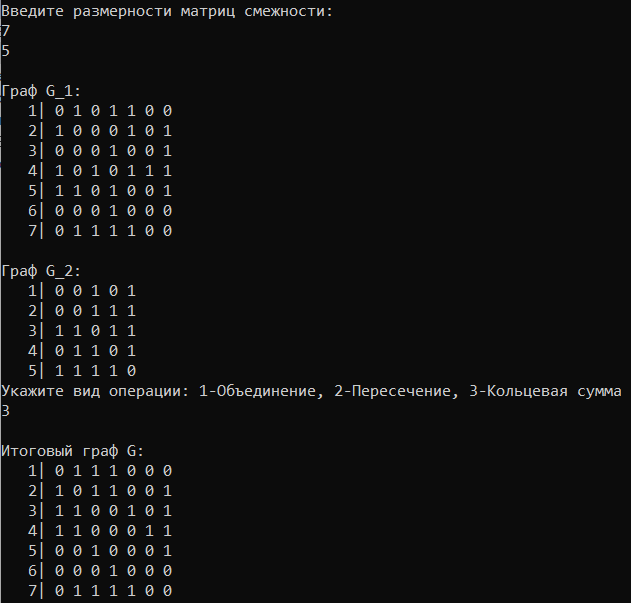


Рис.7.