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”

Выполнил студент гр. 22ВВВ3:
Кулахметов С.И.

Приняли:
к.т.н., доцент Юрова О.В.
к.э.н., доцент Акифьев И.В.

Пенза 2023

Цель работы

Рассмотреть унарные и бинарные операции, выполняемые над графами, такие как: отождествление вершин, стягивание ребра, расщепление вершины, объединение, пересечение и кольцевая сумма графов. Выполнить лабораторное задание.

Лабораторное задание

Задание 1

1. Сгенерировать (используя генератор случайных чисел) две матрицы M_1, M_2 смежности неориентированных графов G_1, G_2. Вывести сгенерированные матрицы на экран.

2. Для указанных графов преобразовать представление матриц смежности в списки смежности. Вывести полученные списки на экран.

Задание 2

1. Для матричной формы представления графов выполнить операции:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин вводить с клавиатуры. Результат выполнения операции вывести на экран.

Задание 3

1. Для матричной формы представления графов выполнить операции:

а) объединения

б) пересечения

в) кольцевой суммы

Результат выполнения операции вывести на экран.

Пояснительный текст к прорамме

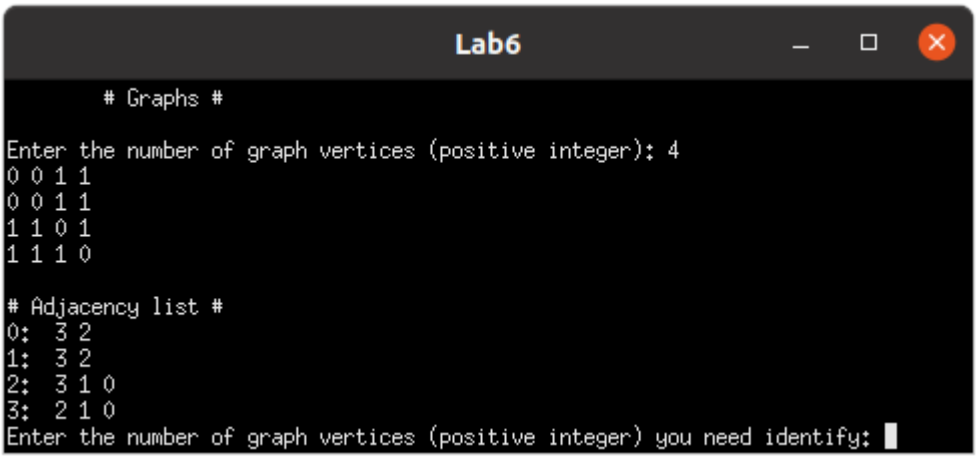
Программа разделена на 2 основных файла, в которых реализована лабораторная работа: *Lab6_Task2/main.c* — содержит код заданий 1 и 2; *Lab6_Task3/main.c* — содержит код 3 задания. Первый файл содержит следующие функции: *VertexIdentification()* - отождествление вершин, *EdgeContraction()* - стягивание ребра, SplitVertex() — расщепление вершины. Второй файл содержит следующие функции: *GraphUnion()* - объединение графов, *GraphIntersection()* -

пересечение графов, *GraphXOR()* - кольцевая сумма графов. Все операции осуществляются с графами, представленными матрицами смежности.

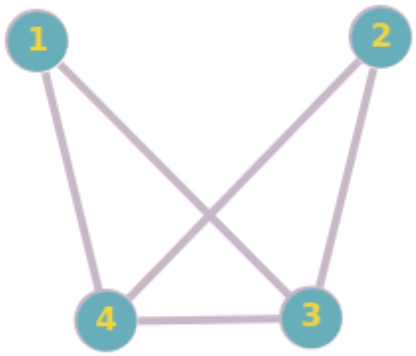
Результаты работы программы

Задание 1

Построение матрицы и списков смежности.

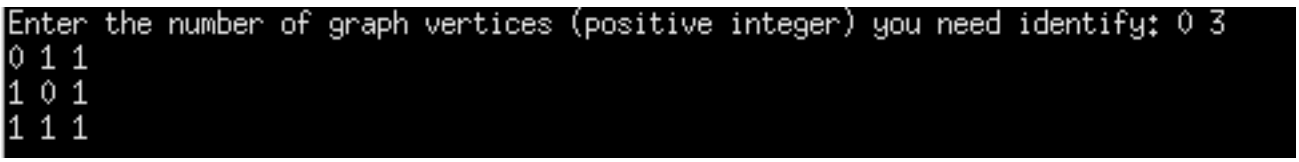


Граф, сгенерированный по матрице смежности.

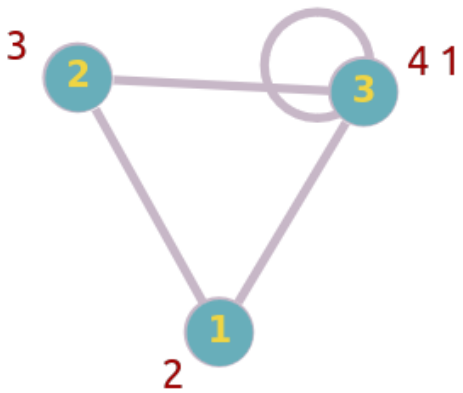


Задание 2

Отождествление вершин.



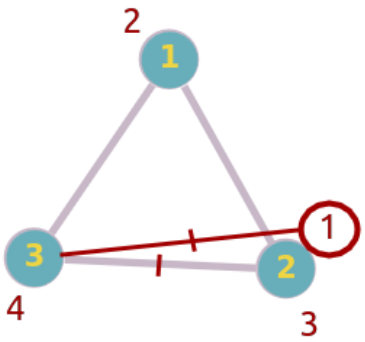
Новый граф с отождествлёнными вершинами.



Стягивание ребра. Для этого необходимо выбрать 2 вершины ребро между которыми мы хотим стянуть.

```
Enter the number of graph vertices (positive integer) whose edge need to be cont
racted: 0 2
0 1 1
1 0 1
1 1 0
```

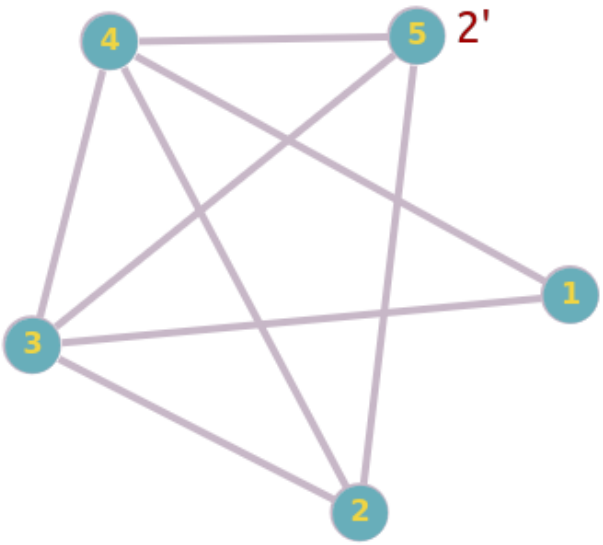
Новый граф со стянутым ребром.



Расщепление вершины — это операция, противоположная отождествлению. Для этого выбирвем вершину, которую хотим расщепить.

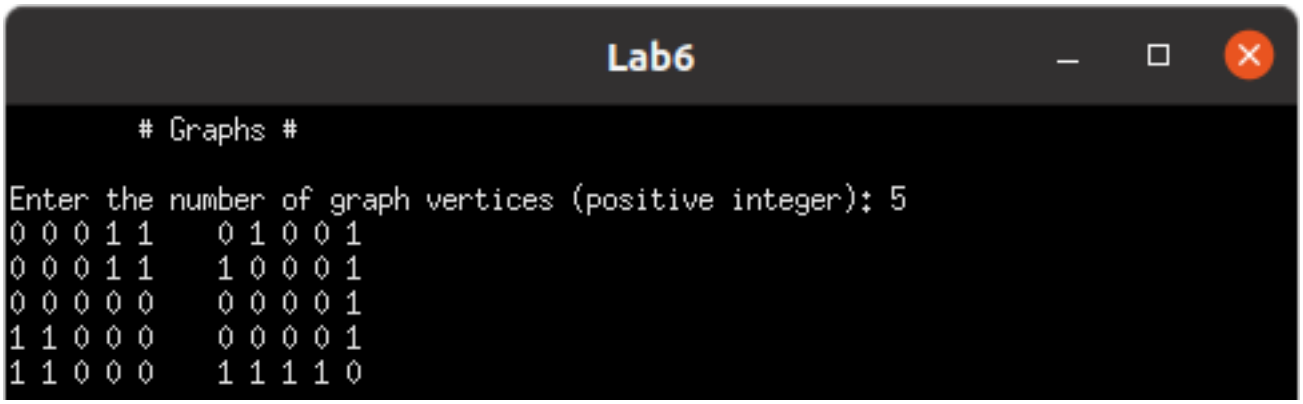
```
Enter the number of graph vertices (positive integer) you need split: 1
0 0 1 1 0
0 0 1 1 1
1 1 0 1 1
1 1 1 0 1
0 1 1 1 0
```

Новый граф с расщеплённой вершиной, отмеченной штрихом.

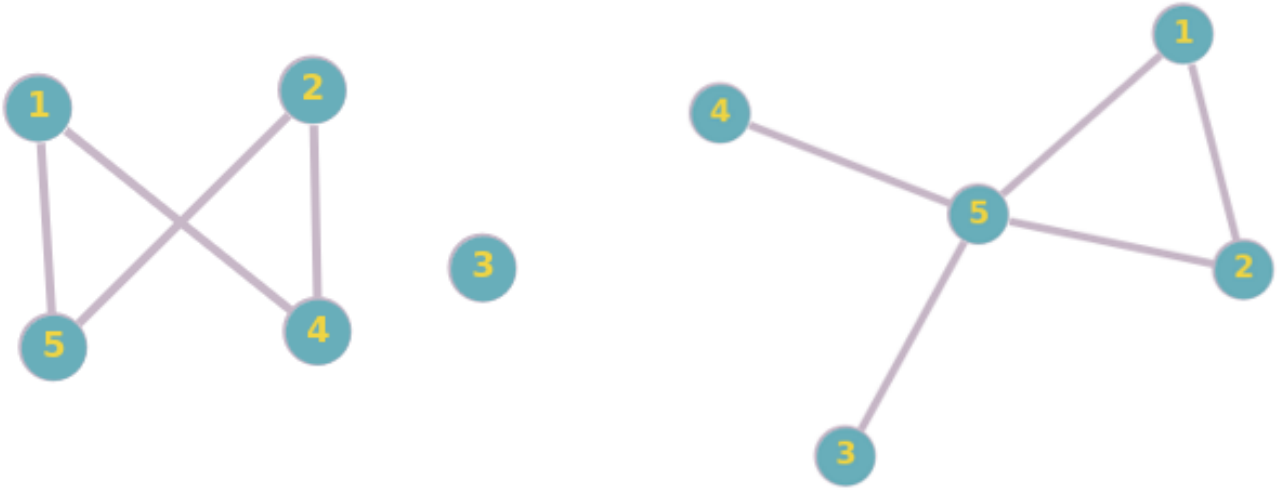


Задание 3

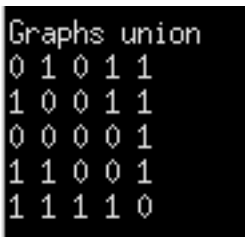
Сгенерированы матрицы смежности 2-х графов, над которыми будут осуществляться бинарные операции.



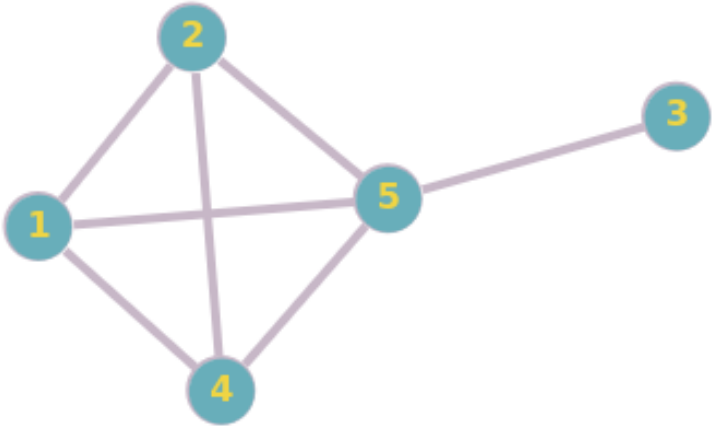
Графы, сгенерированные по полученным матрицам смежности.



Матрица смежности графа, полученного в результате объединения.



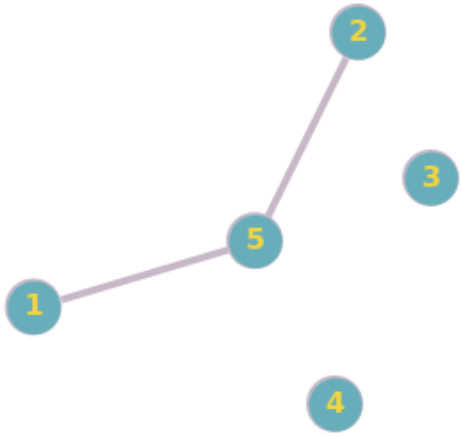
Результирующий граф объединения.



Матрица смежности графа, полученного в результате пересечения.

```
Graphs intersection
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
0 0 0 0 0
1 1 0 0 0
```

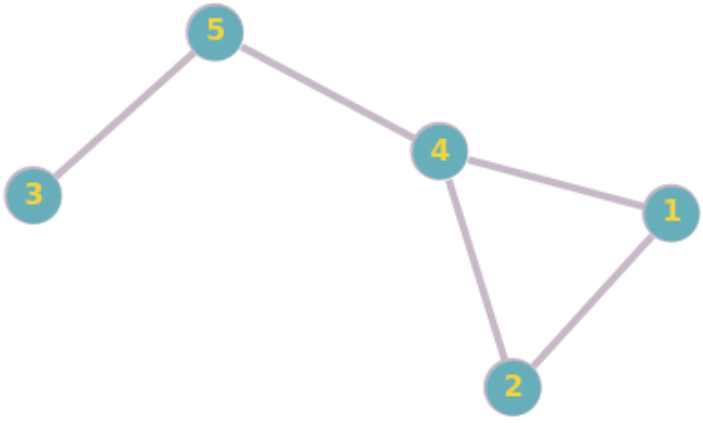
Результирующий граф пересечения.



Матрица смежности графа, полученного в результате кольцевой суммы.

```
Ring sum of graphs
0 1 0 1 0
1 0 0 1 0
0 0 0 0 1
1 1 0 0 1
0 0 1 1 0
```

Результирующий граф кольцевой суммы.



Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации на языке Си алгоритмов для осуществления унарных и бинарных операций над графами, представленными матрицами смежности.

Ссылка на *GitHub* репозиторий с лабораторной работой

https://github.com/KulakhmetovS/Lab6

Приложение А

Листинг программы

Файл Lab6_Task2/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int** Creategraph(int **, int);

int size;

// <---------- Список смежности ---------->
struct node
{
   int vertex;
   struct node* next;
};
struct node* createNode(int);
struct Graph
{
   int numVertices;
   struct node** adjLists;
};
struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);
// <---------- ! ---------->


int **VertexIdentification(int **, int, int, int);
int **EdgeContraction(int **, int, int, int);
int **SplitVertex(int **, int, int);


int main()
{
   int i, j, num;
   int **graph = NULL, **graphClone = NULL, **Graph = NULL;

   printf("\t# Graphs #\n\n");
   printf("Enter the number of graph vertices (positive integer): ");
   scanf("%d", &size);
   num = size - 1;

   // Creating the graph
   graph = Creategraph(graph, size);
   graphClone = Creategraph(graphClone, size);

   // Printing the matrix
   for(i = 0; i < size; i++)
   {
      for(j = 0; j < size; j++)
      {
         printf("%d ", graph[i][j]);
         graphClone[i][j] = graph[i][j];
      }
      printf("\n");
```

```
    }


// <---------- Список смежности ---------->
    struct Graph* graf = createGraph(size);

    for(i = 0; i < size; i++)
    {
       for(j = i; j < size; j++)
       {
          if(graph[i][j] == 1)
          {
             addEdge(graf, i, j);
          }
       }
    }

    printf("\n# Adjacency list #\n");
    printGraph(graf);

    int vertex1, vertex2;
    printf("Enter the number of graph vertices (positive integer) you need identify: ");
    scanf("%d%d", &vertex1, &vertex2);

    Graph = VertexIdentification(graph, size, vertex1, vertex2);

    for(i = 0; i < size - 1; i++)
    {
       for(j = 0; j < size - 1; j++)
          printf("%d ", Graph[i][j]);
       printf("\n");
    }
    printf("\n");

    for(i = 0; i < size; i++)
       for(j = 0; j < size; j++)
          graph[i][j] = graphClone[i][j];


    Graph = NULL, vertex1 = 0, vertex2 = 0;

     printf("Enter the number of graph vertices (positive integer) whose edge need to
be contracted: ");
    scanf("%d%d", &vertex1, &vertex2);

    Graph = EdgeContraction(graphClone, size, vertex1, vertex2);

    for(i = 0; i < size - 1; i++)
    {
       for(j = 0; j < size - 1; j++)
          printf("%d ", Graph[i][j]);
       printf("\n");
    }
    printf("\n");

    printf("Enter the number of graph vertices (positive integer) you need split: ");
    scanf("%d", &vertex1);
    Graph = SplitVertex(graph, size, vertex1);

    for(i = 0; i < size + 1; i++)
```

```
    {
        for(j = 0; j < size + 1; j++)
            printf("%d ", Graph[i][j]);
        printf("\n");
    }

    free(graphClone);
    free(Graph);
    free(graph);

    return 0;
}


int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            if(graph[i][j] == 1);
        }

    return graph;
}


// <---------- Список смежности ---------->
struct node* createNode(int v)
{
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices)
{
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices * sizeof(struct node*));

    int i;
    for (i = 0; i < vertices; i++)
        graph->adjLists[i] = NULL;

    return graph;
```

```
}

void addEdge(struct Graph* graph, int src, int dest)
{
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node* temp = graph->adjLists[v];
        printf("%d:  ", v);
        while (temp)
        {
            printf("%d ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

int **VertexIdentification(int **graph, int size, int vertex1 , int vertex2)
{
    int res = 0, val1, val2, i, j;
    int **Graph = NULL;

    Graph = (int **)(malloc(sizeof(int *) * (size - 1)));
    for(i = 0; i < size - 1; i++)
        Graph[i] = (int *)(malloc(sizeof(int *) * (size - 1)));

    // ---------- logical addition of strings ----------
    for(i = 0; i < size; i++)
    {
        //graph[vertex1][i] || graph[vertex2][i];
        val1 = graph[vertex1][i];
        val2 = graph[vertex2][i];

        __asm(
        "movl %[val1], %%eax\n\t"
        "movl %[val2], %%ebx\n\t"
        "orl %%ebx, %%eax\n\t"
        :"=a" (res)
        :[val1] "m" (val1), [val2] "m" (val2)
        :"cc"
        );

        graph[vertex2][i] = res;
    }

    for(i = 0; i < size; i++)
```

```
    {
        val1 = graph[i][vertex1];
        val2 = graph[i][vertex2];

        __asm(
        "movl %[val1], %%eax\n\t"
        "movl %[val2], %%ebx\n\t"
        "orl %%ebx, %%eax\n\t"
        :"=a" (res)
        :[val1] "m" (val1), [val2] "m" (val2)
        :"cc"
        );

        graph[i][vertex2] = res;
    }

    for (i  = vertex1; i < size - 1; i++)
        for (j = 0; j < size; j++)
            graph[i][j] = graph[i + 1][j];

    for (i = 0; i < size; i++)
        for (j = vertex1; j < size - 1; j++)
            graph[i][j] = graph[i][j + 1];

    for (i = 0; i < size - 1; i++)
        for (j = 0; j < size - 1; j++)
            Graph[i][j] = graph[i][j];

    return Graph;
}

int **EdgeContraction(int **graph, int size, int vertex1 , int vertex2)
{
    int **Graph = NULL;
    Graph = (int **)(malloc(sizeof(int *) * (size - 1)));

    Graph = VertexIdentification(graph, size, vertex1, vertex2);

    for(int i = 0; i < size - 1; i++)
        for(int j = 0; j < size - 1; j++)
                if(i == j) Graph[i][j] = 0;

    return Graph;
}

int **SplitVertex(int **graph, int size, int vertex)
{
    int i, j;
    int **Graph = NULL;

    Graph = (int **)(malloc(sizeof(int *) * (size + 1)));
    for(i = 0; i < size + 1; i++)
        Graph[i] = (int *)(malloc(sizeof(int *) * (size + 1)));


    for(i = 0; i < size; ++i)
        for(j = 0; j < size; ++j)
            Graph[i][j] = graph[i][j];

    for(i = 0; i < size; ++i)
```

```
    {
        Graph[i][size] = graph[i][vertex];
        Graph[size][i] = graph[vertex][i];
    }

    Graph[vertex][size] = 1;
    Graph[size][vertex] = 1;

    return Graph;
}
```

Файл Lab6_Task3/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int** Creategraph(int **, int);

int **GraphUnion(int **, int**, int);
int **GraphIntersection(int **, int**, int);
int **GraphXOR(int **, int**, int);
void PrintGraph(int **, int);

int main()
{
    int i, j, size;
    int **graph1 = NULL, **graph2 = NULL, **Graph = NULL;

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);

    // Creating the graph
    graph1 = Creategraph(graph1, size);
    sleep(1);
    graph2 = Creategraph(graph2, size);

    // Printing the matrix
    for(i = 0; i < size; i++)
    {
        for(j = 0; j < size; j++)
        {
            printf("%d ", graph1[i][j]);
        }
        printf("    ");
        for(j = 0; j < size; j++)
        {
            printf("%d ", graph2[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    Graph = GraphUnion(graph1, graph2, size);

    printf("Graphs union\n");
    PrintGraph(Graph, size);
```

```
    for(i = 0; i < size; i++)
        free(Graph[i]);
    free(Graph);

    Graph = GraphIntersection(graph1, graph2, size);

    printf("Graphs intersection\n");
    PrintGraph(Graph, size);

    for(i = 0; i < size; i++)
        free(Graph[i]);
    free(Graph);

    Graph = GraphXOR(graph1, graph2, size);

    printf("Ring sum of graphs\n");
    PrintGraph(Graph, size);

    for(i = 0; i < size; i++)
        free(Graph[i]);
    free(Graph);
    for(i = 0; i < size; i++)
        free(graph1[i]);
    free(graph1);
    for(i = 0; i < size; i++)
        free(graph2[i]);
    free(graph2);
    return 0;
}


int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            if(graph[i][j] == 1);
        }

    return graph;
}

int **GraphUnion(int **graph1, int** graph2, int size)
{
    int **Graph = NULL;
    int res, val1, val2;
```

```
    Graph = (int **)(malloc(sizeof(int *) * size));
    for(int i = 0; i < size; i++)
        Graph[i] = (int *)(malloc(sizeof(int *) * size));

    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            Graph[i][j] = graph2[i][j];

    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            {
                val1 = graph1[i][j];
                val2 = Graph[i][j];
                __asm(
                        "movl %[val2], %%eax\n\t"
                        "movl %[val1], %%ebx\n\t"
                        "orl %%ebx, %%eax\n\t"
                        :"=a" (res)
                        :[val1] "m" (val1), [val2] "m" (val2)
                        :"cc"
                        );

                Graph[i][j] = res;
            }

    return Graph;
}

int **GraphIntersection(int **graph1, int**graph2, int size)
{
    int **Graph = NULL;
    int res, val1, val2;

    Graph = (int **)(malloc(sizeof(int *) * size));
    for(int i = 0; i < size; i++)
        Graph[i] = (int *)(malloc(sizeof(int *) * size));

    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            Graph[i][j] = graph2[i][j];

    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            {
                val1 = graph1[i][j];
                val2 = Graph[i][j];
                __asm(
                        "movl %[val2], %%eax\n\t"
                        "movl %[val1], %%ebx\n\t"
                        "andl %%ebx, %%eax\n\t"
                        :"=a" (res)
                        :[val1] "m" (val1), [val2] "m" (val2)
                        :"cc"
                        );

                Graph[i][j] = res;
            }

    return Graph;
```

```
}

int **GraphXOR(int **graph1, int **graph2, int size)
{
   int **Graph = NULL;
   int res, val1, val2;

   Graph = (int **)(malloc(sizeof(int *) * size));
   for(int i = 0; i < size; i++)
      Graph[i] = (int *)(malloc(sizeof(int *) * size));

   for(int i = 0; i < size; i++)
      for(int j = 0; j < size; j++)
         Graph[i][j] = graph2[i][j];

   for(int i = 0; i < size; i++)
      for(int j = 0; j < size; j++)
         {
            val1 = graph1[i][j];
            val2 = Graph[i][j];
            __asm(
                  "movl %[val2], %%eax\n\t"
                  "movl %[val1], %%ebx\n\t"
                  "xorl %%ebx, %%eax\n\t"
                  :"=a" (res)
                  :[val1] "m" (val1), [val2] "m" (val2)
                  :"cc"
                  );

            Graph[i][j] = res;
         }

   return Graph;
}

void PrintGraph(int **Graph, int size)
{
   for(int i = 0; i < size; i++)
   {
      for(int j = 0; j < size; j++)
      {
         printf("%d ", Graph[i][j]);
      }
      printf("\n");
   }
   printf("\n");
}
```