

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ 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 группы 24ВВВЗ:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Изучение унарных и бинарных операций над графами.

Лабораторное задание

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) две матрицы M_1 , M_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.

2. * 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 $G = G_1 \cup G_2$
- б) пересечения $G = G_1 \cap G_2$
- в) кольцевой суммы $G = G_1 \oplus G_2$

Результат выполнения операции выведите на экран.

Задание 4 *

1. Для матричной формы представления графов выполните операцию декартова произведения графов $G = G_1 \times G_2$. Результат выполнения операции выведите на экран.

Пояснительный текст к программам

Все унарные операции над графами можно объединить в две группы. Первую группу составляют операции, с помощью которых из исходного графа G_1 , можно построить граф G_2 с меньшим числом элементов. В группу входят операции удаления ребра или вершины, отождествления вершин, стягивание ребра. Вторую группу составляют операции, позволяющие строить графы с большим числом элементов. В группу входят операции расщепления вершин, добавления ребра.

Отождествление вершин. В графе G_1 выделяются вершины u, v . Определяют окружение Q_1 вершины u , и окружение Q_2 вершины v , вычисляют их объединение $Q = Q_1 \cup Q_2$. Затем над графом G_1 выполняются следующие преобразования:

- из графа G_1 удаляют вершины u, v ($H_1 = G_1 - u - v$);
- к графу H_1 присоединяют новую вершину z ($H_1 = H_1 + z$);
- вершину z соединяют ребром с каждой из вершин $w_i \in Q$ ($G_2 = H_1 + zw_i, i = 1, 2, 3, \dots$).

Стягивание ребра. Данная операция является операцией отождествления смежных вершин u, v в графе G_1 .

Наиболее важными бинарными операциями являются: объединение, пересечение, декартово произведение и кольцевая сумма.

Объединение. Граф G называется объединением или наложением графов G_1 и G_2 , если $V_G = V_1 \cup V_2$; $E_G = E_1 \cup E_2$ (рис. 1).

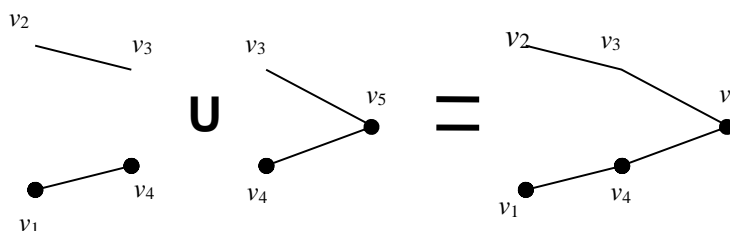


Рис. 1. Объединение графов G_1, G_2

Объединение графов G_1 и G_2 называется дизъюнктивным, если $V_1 V_2 = \emptyset$. При дизъюнктивном объединении никакие два из объединяемых графов не должны иметь общих вершин.

Пересечение. Граф G называется пересечением графов G_1, G_2 , если $V_G = V_1 V_2$ и $U_G = U_1 U_2$ (рис.2). Операция "пересечения" записывается следующим образом: $G = G_1 G_2$.

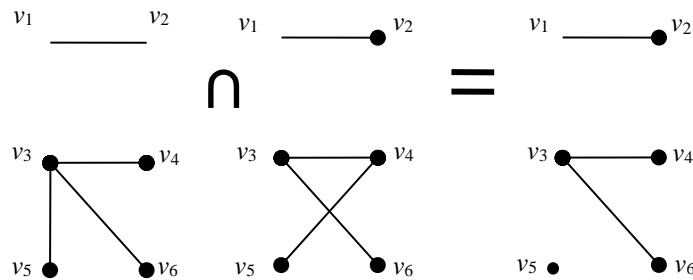


Рис.2. Пересечение графов G_1, G_2 .

Декартово произведение. Граф G называется декартовым произведением графов G_1 и G_2 если $V_G = V_1 \times V_2$ — декартово произведение множеств вершин графов G_1, G_2 , а множество ребер U_G задается следующим образом: вершины (z_i, v_k) и (z_j, v_l) смежны в графе G тогда и только тогда, когда $z_i = z_j (i = j)$, а v_k и v_l смежны в G_2 или $v_k = v_l (k = l)$, смежны в графе G_1 (см. рис.3).

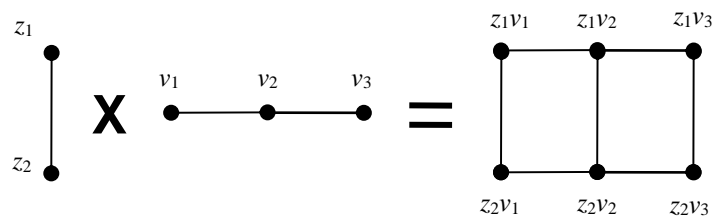
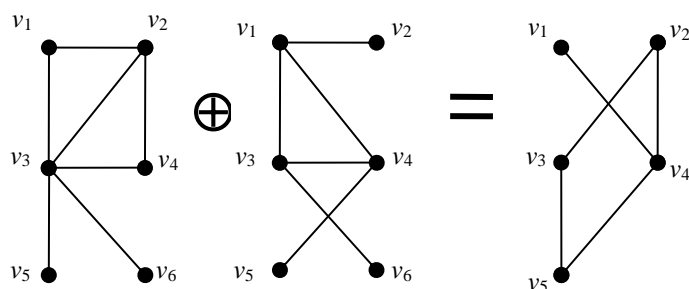


Рис. 3. Декартово произведение графов G_1, G_2

Кольцевая сумма графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: $G = G_1 \oplus G_2$ (рис.4).



Результаты программ

Рисунок 4 - Результаты работы **zadanie1.cpp**

```
Генерация двух графов и вывод их матриц и списков смежности
Введите количество вершин G1: 7
Введите количество вершин G2: 2

Матрица смежности G1:
Матрица смежности (7 x 7):
0 1 1 0 1 1 1
1 1 1 0 1 1 0
1 1 0 1 0 0 1
0 0 1 1 0 1 0
1 1 0 0 1 1 0
1 1 0 1 1 1 0
1 0 1 0 0 0 0

Список смежности G1:
Список смежности:
0: 1 2 4 5 6
1: 0 1 2 4 5
2: 0 1 3 6
3: 2 3 5
4: 0 1 4 5
5: 0 1 3 4 5
6: 0 2

Матрица смежности G2:
Матрица смежности (2 x 2):
0 1
1 1

Список смежности G2:
Список смежности:
0: 1
1: 0 1
```

Рисунок 5 - Результаты работы **zadanie2.cpp**

```
Введите количество вершин графа: 5
Исходный граф:
Матрица смежности:
0: 0 0 0 0 0
1: 0 1 0 0 0
2: 0 0 0 0 0
3: 0 0 0 0 1
4: 0 0 0 1 1

Списки смежности:
0: нет соседей
1: 1
2: нет соседей
3: 4
4: 3, 4

Отождествление вершин (матрица)
Введите вершину u для отождествления: 3
Введите вершину v для отождествления: 4
Матрица смежности:
0: 0 0 0 0
1: 0 1 0 0
2: 0 0 0 0
3: 0 0 0 1

Отождествление вершин (списки смежности)

Списки смежности:
0: нет соседей
1: 1
2: нет соседей
3: 3

Стягивание ребра (матрица)
Введите вершину u: 1
Введите вершину v: 2

Расщепление вершины (матрица)
Введите вершину для расщепления: 1
Матрица смежности:
0: 0 0 0 0 0
1: 0 1 0 0 1
2: 0 0 0 0 0
3: 0 0 0 1 0
4: 0 1 0 0 0

Расщепление вершины (списки смежности)

Списки смежности:
0: нет соседей
1: 1, 4
2: нет соседей
3: 3
4: 1
```

Рисунок 6 - Результаты работы **zadanie3.cpp**

```
Объединение, пересечение, кольцевая сумма
Введите количество вершин G1: 5
Введите количество вершин G2: 3

G1:
Матрица смежности (5 x 5):
1 1 0 0 1
1 1 1 0 0
0 1 0 0 0
0 0 0 0 0
1 0 0 0 1

G2:
Матрица смежности (3 x 3):
0 0 1
0 1 0
1 0 0

Выберите операцию:
1 - Объединение
2 - Пересечение
3 - Кольцевая сумма
Введите номер операции: 3

Результат:
Матрица смежности (5 x 5):
1 1 1 0 1
1 0 1 0 0
1 1 0 0 0
0 0 0 0 0
1 0 0 0 1
```

Рисунок 7 - Результаты работы **zadanie3.cpp**

```
Объединение, пересечение, кольцевая сумма
Введите количество вершин G1: 5
Введите количество вершин G2: 3

G1:
Матрица смежности (5 x 5):
1 1 0 1 0
1 1 0 0 0
0 0 1 0 1
1 0 0 1 0
0 0 1 0 0

G2:
Матрица смежности (3 x 3):
0 0 0
0 0 0
0 0 0

Выберите операцию:
1 - Объединение
2 - Пересечение
3 - Кольцевая сумма
Введите номер операции: 2

Результат:
Матрица смежности (3 x 3):
0 0 0
0 0 0
0 0 0
```

Рисунок 8 - Результаты работы **zadanie3.cpp**

```
Объединение, пересечение, кольцевая сумма
Введите количество вершин G1: 5
Введите количество вершин G2: 3

G1:
Матрица смежности (5 x 5):
1 0 1 1 0
0 0 0 1 1
1 0 0 1 1
1 1 1 0 1
0 1 1 1 1

G2:
Матрица смежности (3 x 3):
1 1 1
1 1 1
1 1 0

Выберите операцию:
1 - Объединение
2 - Пересечение
3 - Кольцевая сумма
Введите номер операции: 1

Результат:
Матрица смежности (5 x 5):
1 1 1 1 0
1 1 1 1 1
1 1 0 1 1
1 1 1 0 1
0 1 1 1 1
```

Рисунок 9 - Результаты работы **zadanie4.cpp**

```

Декартово произведение графов ===
Введите количество вершин G1: 5
Введите количество вершин G2: 6

G1:
Матрица смежности (5 x 5):
0 0 1 1 0
0 1 1 1 0
1 1 0 0 1
1 1 0 0 1
0 0 1 1 0

G2:
Матрица смежности (6 x 6):
0 1 0 1 0 1
1 0 1 1 1 0
0 1 0 1 0 1
1 1 1 1 0 0
0 1 0 0 1 1
1 0 1 0 1 1

Декартово произведение (5 * 6 = 30 вершин):
Матрица смежности (30 x 30):
0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0
0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0
0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 1

```

Вывод: в ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №6 – Унарные и бинарные операции над графами.

Приложение А Листинг

Файл `zadanie1.cpp`

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <limits>
#include <string>

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (std::cin.peek() != '\n') {
            std::cout << "Ошибка: введено не целое число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        return value;
    }
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int** allocArr(int n) {
    int** arr = new int*[n];
    for (int i = 0; i < n; ++i) {
        arr[i] = new int[n];
        for (int j = 0; j < n; ++j) arr[i][j] = 0;
    }
    return arr;
}

void freeArr(int** arr, int n) {
```

```

    if (!arr) return;
    for (int i = 0; i < n; ++i) delete[] arr[i];
    delete[] arr;
}

void printArr(int** arr, int n) {
    std::cout << "Матрица смежности (" << n << " x " << n << "):\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) std::cout << arr[i][j] << " ";
        std::cout << "\n";
    }
}

void toAdjList(int** arr, int n) {
    std::cout << "Список смежности:\n";
    for (int i = 0; i < n; ++i) {
        std::cout << i << ": ";
        for (int j = 0; j < n; ++j) if (arr[i][j]) std::cout << j << " ";
        std::cout << "\n";
    }
}

int** generateGraph(int n) {
    int** arr = allocArr(n);
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j)
            arr[i][j] = arr[j][i] = rand() % 2;
    return arr;
}

int main() {
    clearScreen();
    srand(static_cast<unsigned int>(time(nullptr)));

    std::cout << "Генерация двух графов и вывод их матриц и списков
смежности\n";

    int n1 = isInteger("Введите количество вершин G1: ");
    while (n1 < 2) n1 = isInteger("Минимум 2. Повторите: ");

    int n2 = isInteger("Введите количество вершин G2: ");
    while (n2 < 2) n2 = isInteger("Минимум 2. Повторите: ");

    int** M1 = generateGraph(n1);
    int** M2 = generateGraph(n2);

    std::cout << "\nМатрица смежности G1:\n";
    printArr(M1, n1);

```

```

std::cout << "\nСписок смежности G1:\n";
toAdjList(M1, n1);

std::cout << "\nМатрица смежности G2:\n";
printArr(M2, n2);
std::cout << "\nСписок смежности G2:\n";
toAdjList(M2, n2);

freeArr(M1, n1);
freeArr(M2, n2);

std::cout << "\n\n";

return 0;
}

```

Файл zadanie2.cpp

```

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <limits>
#include <locale>

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

struct AdjList {
    int** lists;
    int* lengths;
    int n;
};

AdjList* createAdjListFromMatrix(int** matrix, int n) {
    AdjList* adj = new AdjList;
    adj->n = n;
    adj->lists = (int**)malloc(n * sizeof(int*));
    adj->lengths = (int*)calloc(n, sizeof(int));

    for (int i = 0; i < n; ++i) {
        int count = 0;
        for (int j = 0; j < n; ++j)
            if (matrix[i][j]) count++;
        adj->lengths[i] = count;
    }
}

```

```

        adj->lists[i] = (int*)malloc(count * sizeof(int));
        int idx = 0;
        for (int j = 0; j < n; ++j)
            if (matrix[i][j]) adj->lists[i][idx++] = j;
    }
    return adj;
}

void printAdjList(AdjList* adj) {
    std::cout << "\nСписки смежности:\n";
    for (int i = 0; i < adj->n; ++i) {
        std::cout << i << ": ";
        if (adj->lengths[i] == 0) std::cout << "нет соседей";
        else {
            for (int j = 0; j < adj->lengths[i]; ++j) {
                std::cout << adj->lists[i][j];
                if (j < adj->lengths[i] - 1) std::cout << ", ";
            }
        }
        std::cout << "\n";
    }
}

// Фри память списка
void freeAdjList(AdjList* adj) {
    if (!adj) return;
    for (int i = 0; i < adj->n; ++i) free(adj->lists[i]);
    free(adj->lists);
    free(adj->lengths);
    delete adj;
}

int** allocArr(int n) {
    int** arr = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; ++i)
        arr[i] = (int*)calloc(n, sizeof(int));
    return arr;
}

void freeArr(int** arr, int n) {
    if (!arr) return;
    for (int i = 0; i < n; ++i) free(arr[i]);
    free(arr);
}

int** copyArr(int** src, int n) {
    int** dst = allocArr(n);
    for (int i = 0; i < n; ++i)

```

```

        for (int j = 0; j < n; ++j)
            dst[i][j] = src[i][j];
    return dst;
}

```

```

void printMatrix(int** arr, int n) {
    std::cout << "Матрица смежности:\n";
    for (int i = 0; i < n; ++i) {
        std::cout << i << ": ";
        for (int j = 0; j < n; ++j)
            std::cout << arr[i][j] << " ";
        std::cout << "\n";
    }
}

```

```

int** generateGraph(int n) {
    int** arr = allocArr(n);
    for (int i = 0; i < n; ++i) {
        for (int j = i; j < n; ++j) {
            arr[i][j] = arr[j][i] = rand() % 2;
        }
    }
    return arr;
}

```

// Отождествление

```

int** mergeVertices(int** arr, int n, int u, int v, int& newN) {
    if (u == v || u < 0 || v < 0 || u >= n || v >= n) return nullptr;
    newN = n - 1;
    int** newArr = allocArr(newN);

    int idx_i = 0;
    for (int i = 0; i < n; ++i) {
        if (i == v) continue;
        int idx_j = 0;
        for (int j = 0; j < n; ++j) {
            if (j == v) continue;
            if (i == u && j == u) newArr[idx_i][idx_j] = arr[u][u] || arr[v][v];
            else if (i == u) newArr[idx_i][idx_j] = arr[u][j] || arr[v][j];
            else if (j == u) newArr[idx_i][idx_j] = arr[i][u] || arr[i][v];
            else newArr[idx_i][idx_j] = arr[i][j];
            idx_j++;
        }
        idx_i++;
    }
    return newArr;
}

```

```
// Стягивание
int** contractEdge(int** arr, int n, int u, int v, int& newN) {
    if (arr[u][v] == 0) return nullptr;
    return mergeVertices(arr, n, u, v, newN);
}
```

```
// Расщепление матрица
int** splitVertex(int** arr, int n, int v, int& newN) {
    newN = n + 1;
    int** newArr = allocArr(newN);

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            newArr[i][j] = arr[i][j];

    int newVertex = n;

    newArr[v][newVertex] = newArr[newVertex][v] = 1;
    newArr[newVertex][newVertex] = 0;

    for (int i = 0; i < n; ++i) {
        if (i != v) newArr[i][newVertex] = 0;
        if (i != v) newArr[newVertex][i] = 0;
    }

    return newArr;
}
```

```
// Расщепление список
AdjList* splitVertexAdjList(AdjList* adj, int v) {
    if (!adj || v < 0 || v >= adj->n) return nullptr;

    int newN = adj->n + 1;
    AdjList* newAdj = new AdjList;
    newAdj->n = newN;
    newAdj->lists = (int**)malloc(newN * sizeof(int*));
    newAdj->lengths = (int*)calloc(newN, sizeof(int));

    for (int i = 0; i < adj->n; ++i) {
        int len = adj->lengths[i];
        newAdj->lists[i] = (int*)malloc(len * sizeof(int));
        for (int j = 0; j < len; ++j)
            newAdj->lists[i][j] = adj->lists[i][j];
        newAdj->lengths[i] = len;
    }

    int newVertex = adj->n;
```

```

    newAdj->lengths[v]++;
    newAdj->lists[v] = (int*)realloc(newAdj->lists[v], newAdj->lengths[v] *
sizeof(int));
    newAdj->lists[v][newAdj->lengths[v] - 1] = newVertex;

    newAdj->lengths[newVertex] = 1;
    newAdj->lists[newVertex] = (int*)malloc(sizeof(int));
    newAdj->lists[newVertex][0] = v;

    return newAdj;
}

```

```

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (std::cin.peek() != '\n') {
            std::cout << "Ошибка: введено не целое число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        return value;
    }
}

```

```

int main() {
    setlocale(LC_ALL, "RUS");
    srand(time(NULL));
    clearScreen();

    int n = isInteger("Введите количество вершин графа: ");
    while (n < 3) n = isInteger("Минимум 3. Повторите: ");

    int** arr = generateGraph(n);

    std::cout << "Исходный граф:\n";
    printMatrix(arr, n);

    AdjList* currentAdjList = createAdjListFromMatrix(arr, n);
    printAdjList(currentAdjList);
}

```

```

int newN;
int** result = copyArr(arr, n);
int currentSize = n;

// Отождествление
std::cout << "\nОтождествление вершин (матрица)\n";
int u = isInteger("Введите вершину u для отождествления: ");
int v = isInteger("Введите вершину v для отождествления: ");

int** temp = mergeVertices(result, currentSize, u, v, newN);
if (temp) {
    freeArr(result, currentSize);
    result = temp;
    currentSize = newN;
    printMatrix(result, currentSize);

    freeAdjList(currentAdjList);
    currentAdjList = createAdjListFromMatrix(result, currentSize);
    std::cout << "\nОтождествление вершин (списки смежности)\n";
    printAdjList(currentAdjList);
}

// Стягивание
std::cout << "\nСтягивание ребра (матрица)\n";
u = isInteger("Введите вершину u: ");
v = isInteger("Введите вершину v: ");

temp = contractEdge(result, currentSize, u, v, newN);
if (temp) {
    freeArr(result, currentSize);
    result = temp;
    currentSize = newN;
    printMatrix(result, currentSize);

    freeAdjList(currentAdjList);
    currentAdjList = createAdjListFromMatrix(result, currentSize);
    std::cout << "\nСтягивание ребра (списки смежности)\n";
    printAdjList(currentAdjList);
}

// Расщепление
std::cout << "\nРасщепление вершины (матрица)\n";
int w = isInteger("Введите вершину для расщепления: ");
temp = splitVertex(result, currentSize, w, newN);
if (temp) {
    freeArr(result, currentSize);
    result = temp;
    currentSize = newN;
}

```

```

    printMatrix(result, currentSize);

    AdjList* visualAdj = splitVertexAdjList(currentAdjList, w);
    std::cout << "\nРасщепление вершины (списки смежности)\n";
    printAdjList(visualAdj);
    freeAdjList(visualAdj);
}

freeArr(result, currentSize);
freeArr(arr, n);
freeAdjList(currentAdjList);

std::cout << "\n\n";

return 0;
}

```

Файл zadanie3.cpp

```

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <limits>
#include <string>

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (std::cin.peek() != '\n') {
            std::cout << "Ошибка: введено не целое число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
    }
}

```

```

    }
    return value;
}
}

int** allocArr(int n) {
    int** arr = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; ++i) {
        arr[i] = (int*)malloc(n * sizeof(int));
        for (int j = 0; j < n; ++j) arr[i][j] = 0;
    }
    return arr;
}

void freeArr(int** arr, int n) {
    if (!arr) return;
    for (int i = 0; i < n; ++i) free(arr[i]);
    free(arr);
}

void printArr(int** arr, int n) {
    std::cout << "Матрица смежности (" << n << " x " << n << "):\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            std::cout << arr[i][j] << " ";
        std::cout << "\n";
    }
}

int** generateGraph(int n) {
    int** arr = allocArr(n);
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j) {
            arr[i][j] = arr[j][i] = rand() % 2;
        }
    return arr;
}

int** unionGraphs(int** a1, int** a2, int n1, int n2, int& newN) {
    newN = (n1 > n2 ? n1 : n2);
    int** g = allocArr(newN);
    for (int i = 0; i < newN; ++i)
        for (int j = 0; j < newN; ++j) {
            int v1 = (i < n1 && j < n1) ? a1[i][j] : 0;
            int v2 = (i < n2 && j < n2) ? a2[i][j] : 0;
            g[i][j] = v1 || v2;
        }
    return g;
}

```

```

}

int** intersectGraphs(int** a1, int** a2, int n1, int n2, int& newN) {
    newN = (n1 < n2 ? n1 : n2);
    int** g = allocArr(newN);
    for (int i = 0; i < newN; ++i)
        for (int j = 0; j < newN; ++j)
            g[i][j] = a1[i][j] && a2[i][j];
    return g;
}

int** xorGraphs(int** a1, int** a2, int n1, int n2, int& newN) {
    newN = (n1 > n2 ? n1 : n2);
    int** g = allocArr(newN);
    for (int i = 0; i < newN; ++i)
        for (int j = 0; j < newN; ++j) {
            int v1 = (i < n1 && j < n1) ? a1[i][j] : 0;
            int v2 = (i < n2 && j < n2) ? a2[i][j] : 0;
            g[i][j] = (v1 + v2) % 2;
        }
    return g;
}

int main() {
    srand(time(NULL));
    clearScreen();

    std::cout << "Объединение, пересечение, кольцевая сумма\n";

    int n1 = isInteger("Введите количество вершин G1: ");
    while (n1 < 2) n1 = isInteger("Минимум 2. Повторите: ");
    int n2 = isInteger("Введите количество вершин G2: ");
    while (n2 < 2) n2 = isInteger("Минимум 2. Повторите: ");

    int** g1 = generateGraph(n1);
    int** g2 = generateGraph(n2);

    std::cout << "\nG1:\n"; printArr(g1, n1);
    std::cout << "\nG2:\n"; printArr(g2, n2);

    std::cout << "\nВыберите операцию:\n1 — Объединение\n2 —
Пересечение\n3 — Кольцевая сумма\n";
    int op;
    while (true) {
        op = isInteger("Введите номер операции: ");
        if (op >= 1 && op <= 3) break;
        std::cout << "Ошибка: неверный выбор\n";
    }
}

```

```

int newN;
int** res = nullptr;
if (op == 1) res = unionGraphs(g1, g2, n1, n2, newN);
else if (op == 2) res = intersectGraphs(g1, g2, n1, n2, newN);
else if (op == 3) res = xorGraphs(g1, g2, n1, n2, newN);

if (res) {
    std::cout << "\nРезультат:\n";
    printArr(res, newN);
    freeArr(res, newN);
}

freeArr(g1, n1);
freeArr(g2, n2);

std::cout << "\n\n";

return 0;
}

```

Файл zadanie4.cpp

```

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <limits>
#include <string>

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (std::cin.peek() != '\n') {

```

```

        std::cout << "Ошибка: введено не целое число.\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        continue;
    }
    return value;
}
}

```

```

int** allocArr(int n) {
    int** arr = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; ++i) {
        arr[i] = (int*)malloc(n * sizeof(int));
        for (int j = 0; j < n; ++j) arr[i][j] = 0;
    }
    return arr;
}

```

```

void freeArr(int** arr, int n) {
    if (!arr) return;
    for (int i = 0; i < n; ++i) free(arr[i]);
    free(arr);
}

```

```

void printArr(int** arr, int n) {
    std::cout << "Матрица смежности (" << n << " x " << n << "):\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            std::cout << arr[i][j] << " ";
        std::cout << "\n";
    }
}

```

```

int** generateGraph(int n) {
    int** arr = allocArr(n);
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j) {
            arr[i][j] = arr[j][i] = rand() % 2;
        }
    return arr;
}

```

```

int** cartesianProduct(int** a1, int** a2, int n1, int n2, int& newN) {
    newN = n1 * n2;
    int** g = allocArr(newN);
    for (int i1 = 0; i1 < n1; ++i1) {
        for (int j1 = 0; j1 < n2; ++j1) {
            for (int i2 = 0; i2 < n1; ++i2) {

```

```

        for (int j2 = 0; j2 < n2; ++j2) {
            if (j1 == j2 && a1[i1][i2] == 1) {
                g[i1 * n2 + j1][i2 * n2 + j2] = a1[i1][i2];
            }
            if (i1 == i2 && a2[j1][j2] == 1) {
                g[i1 * n2 + j1][i2 * n2 + j2] = a2[j1][j2];
            }
        }
    }
}
return g;
}

```

```

int main() {
    srand(time(NULL));
    clearScreen();

```

```

    std::cout << "Декартово произведение графов ===\n";

```

```

    int n1 = isInteger("Введите количество вершин G1: ");
    while (n1 < 2) n1 = isInteger("Минимум 2. Повторите: ");
    int n2 = isInteger("Введите количество вершин G2: ");
    while (n2 < 2) n2 = isInteger("Минимум 2. Повторите: ");

```

```

    int** g1 = generateGraph(n1);
    int** g2 = generateGraph(n2);

```

```

    std::cout << "\nG1:\n"; printArr(g1, n1);
    std::cout << "\nG2:\n"; printArr(g2, n2);

```

```

    int newN;
    int** res = cartesianProduct(g1, g2, n1, n2, newN);

```

```

    std::cout << "\nДекартово произведение (" << n1 << " × " << n2 << " = " <<
newN << " вершин):\n";
    printArr(res, newN);

```

```

    freeArr(g1, n1);
    freeArr(g2, n2);
    freeArr(res, newN);

```

```

    std::cout << "\n\n";

```

```

    return 0;
}

```