Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Дискретная математика

Лабораторная работа № 2

Тема: «Отношения матриц»

Выполнил: студент группы РИС-24-3б

Караваев, А.А

г. Пермь – 2025

# Основная часть

1. Определяем матрицу 6\*6, как двумерный массив
2. Создаем методы проверки отношений
3. Создать метод вывода для матриц и ответов для них
4. Функция isReflexive

bool isReflexive(const vector<vector<bool>>& matrix) {

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

if (matrix[i][i] == 0) {

return false;

}

}

return true;

} Возвращает true только если все диагональные элементы равны 1. Функция определения рефлексивности.

1. Функция isAntireflexive

bool isAntireflexive(const vector<vector<bool>>& matrix) {

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

if (matrix[i][i] == 1) {

return false;

}

}

return true;

} Возвращает true только если все диагональные элементы равны 0. Функция определения антирефлексивности..

1. Функция isSymmetric

bool isSymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (matrix[i][j] != matrix[j][i]) {

return false;

}

}

}

return true;

} Возвращает true только если матрица симметрична относительно главной диагонали. Функция определения симметричности.

1. Функция isAntisymmetric

bool isAntisymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (matrix[i][j] && matrix[j][i]) {

return false;

}

}

}

return true;

} Возвращает true если таких пар не найдено. Функция определения антисимметричности.

1. Функция isAsymmetric

bool isAsymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

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

if (matrix[i][j] && matrix[j][i]) {

return false;

}

}

}

return true;

} Более строгое условие, чем антисимметричность. Функция определения асимметричности.

1. Функция isTransitive

bool isTransitive(const vector<vector<bool>>& matrix) {

int n = matrix.size();

vector<vector<bool>> R2(n, vector<bool>(n, false));

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

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

for (int k = 0; k < n; k++) {

if (matrix[i][k] && matrix[k][j]) {

R2[i][j] = true;

}

}

}

}

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

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

if (R2[i][j] && !matrix[i][j]) {

return false;

}

}

}

return true;

} Возвращает true если R² ⊆ R. Функция определения транзитивности.

1. Функция isConnected

bool isConnected(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (!matrix[i][j] && !matrix[j][i]) {

return false;

}

}

}

return true;

} Проходит по всем парам различных элементов (i,j), где i ≠ j. Проверяет, что для каждой пары хотя бы одно из отношений matrix[i][j] или matrix[j][i] истинно. Возвращает true если все пары связаны хотя бы в одном направлении. Функция определения связности

1. Функция analyzeMatrix

void analyzeMatrix(const vector<vector<bool>>& matrix, int matrixNumber) {

bool reflexive = isReflexive(matrix);

bool antireflexive = isAntireflexive(matrix);

bool symmetric = isSymmetric(matrix);

bool antisymmetric = isAntisymmetric(matrix);

bool asymmetric = isAsymmetric(matrix);

bool transitive = isTransitive(matrix);

bool connected = isConnected(matrix);

cout << "Результаты анализа матрицы " << matrixNumber << ":" << endl;

cout << "Рефлексивность: " << (reflexive ? "ДА" : "НЕТ") << endl;

cout << "Антирефлексивность: " << (antireflexive ? "ДА" : "НЕТ") << endl;

cout << "Симметричность: " << (symmetric ? "ДА" : "НЕТ") << endl;

cout << "Антисимметричность: " << (antisymmetric ? "ДА" : "НЕТ") << endl;

cout << "Асимметричность: " << (asymmetric ? "ДА" : "НЕТ") << endl;

cout << "Транзитивность: " << (transitive ? "ДА" : "НЕТ") << endl;

cout << "Связность: " << (connected ? "ДА" : "НЕТ") << endl;

cout << endl;

bool isEquivalence = reflexive && symmetric && transitive;

cout << "Типы отношений для матрицы " << matrixNumber << ":" << endl;

cout << "Отношение эквивалентности: " << (isEquivalence ? "ДА" : "НЕТ") << endl;

PartialOrder(antisymmetric, transitive, reflexive, antireflexive, connected);

cout << endl;

} Вызывает все функции проверки свойств для данной матрицы

1. Функция PartialOrder

void PartialOrder(bool antisymmetric, bool transitive, bool reflexive, bool antireflexive, bool connected) {

bool a1 = antireflexive && antisymmetric && transitive && connected;

bool a2 = antireflexive && antisymmetric && transitive;

bool a3 = reflexive && antisymmetric && transitive && connected;

bool a4 = reflexive && antisymmetric && transitive;

cout << "Отношение строгого полного порядка: " << (a1 ? "ДА" : "НЕТ") << endl;

cout << "Отношение строгого частичного порядка: " << (a2 ? "ДА" : "НЕТ") << endl;

cout << "Отношение нестрогого полного порядка: " << (a3 ? "ДА" : "НЕТ") << endl;

cout << "Отношение нестрогого частичного порядка: " << (a4 ? "ДА" : "НЕТ") << endl;

} Определяет типы порядков

1. Функция printMatrix

void printMatrix(const vector<vector<bool>>& matrix) {

for (auto row : matrix) {

for (bool element : row) {

cout << element << " ";

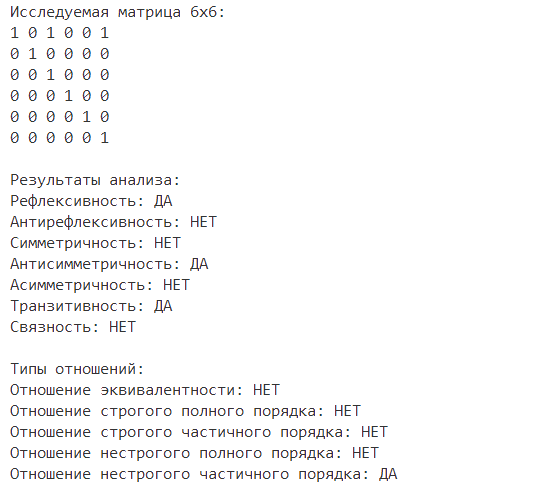
}

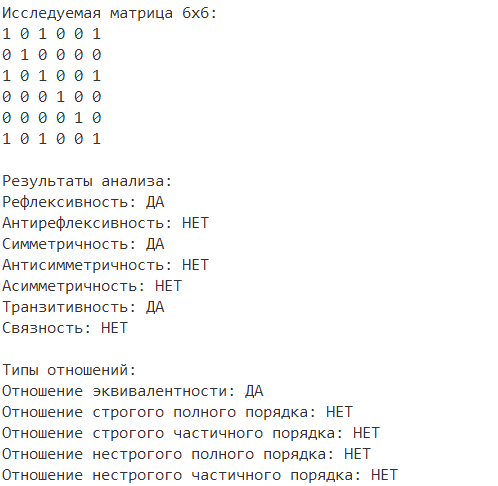
cout << endl;

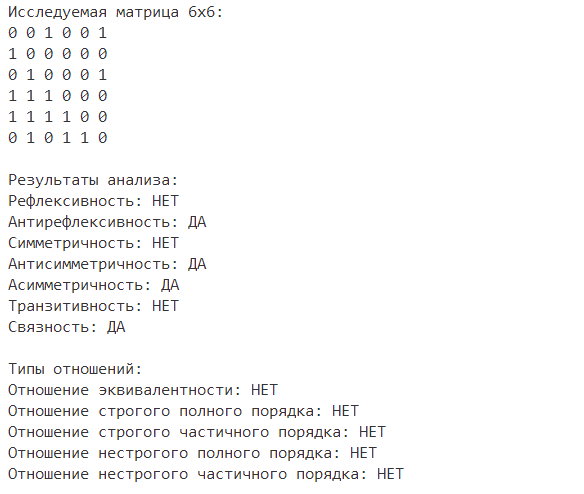
}

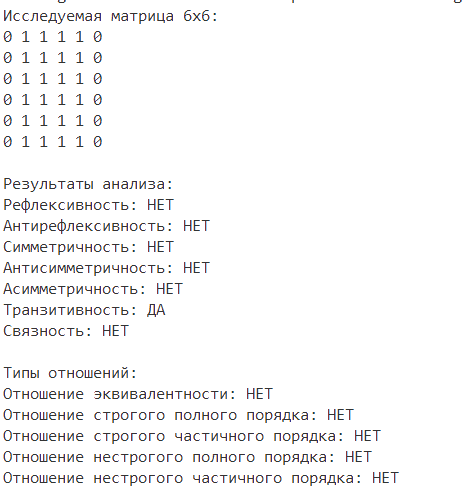
}вывод матрицы

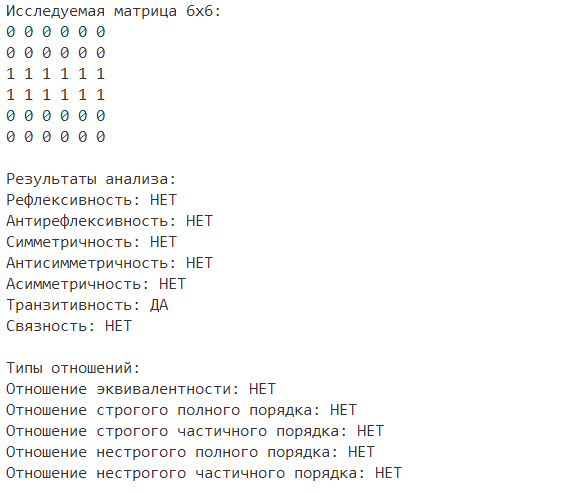
Выходные данные:

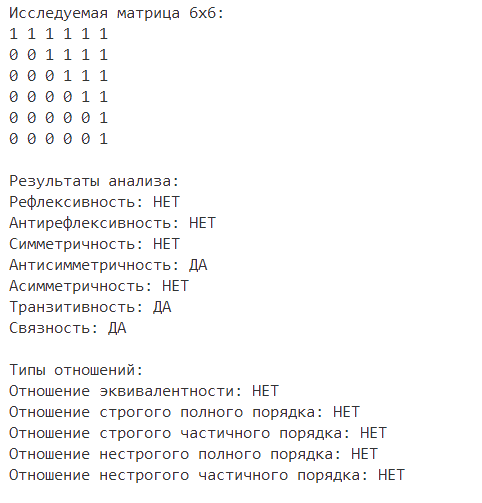


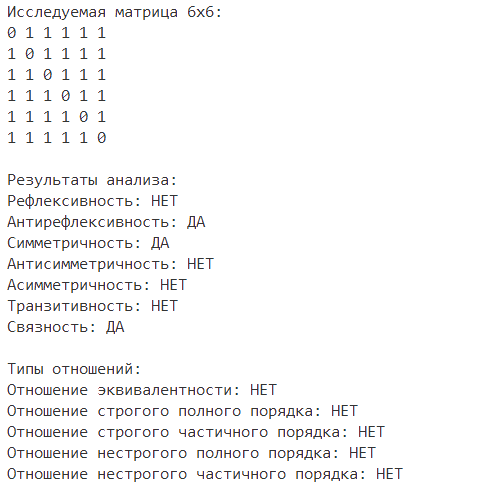
****

****

****

****

****

****

Ссылка на github: https://github.com/Prefix008/Diskretka\_labs

**Приложение А**

#include <iostream>

#include <vector>

using namespace std;

bool isReflexive(const vector<vector<bool>>& matrix);

bool isAntireflexive(const vector<vector<bool>>& matrix);

bool isSymmetric(const vector<vector<bool>>& matrix);

bool isAntisymmetric(const vector<vector<bool>>& matrix);

bool isAsymmetric(const vector<vector<bool>>& matrix);

bool isTransitive(const vector<vector<bool>>& matrix);

bool isConnected(const vector<vector<bool>>& matrix);

void printMatrix(const vector<vector<bool>>& matrix);

void PartialOrder(bool antisymmetric, bool transitive, bool reflexive, bool antireflexive, bool connected) {

bool a1 = antireflexive && antisymmetric && transitive && connected;

bool a2 = antireflexive && antisymmetric && transitive;

bool a3 = reflexive && antisymmetric && transitive && connected;

bool a4 = reflexive && antisymmetric && transitive;

cout << "Отношение строгого полного порядка: " << (a1 ? "ДА" : "НЕТ") << endl;

cout << "Отношение строгого частичного порядка: " << (a2 ? "ДА" : "НЕТ") << endl;

cout << "Отношение нестрогого полного порядка: " << (a3 ? "ДА" : "НЕТ") << endl;

cout << "Отношение нестрогого частичного порядка: " << (a4 ? "ДА" : "НЕТ") << endl;

}

int main() {

setlocale(LC\_ALL, "rus");

vector<vector<bool>> matrix = {

{1, 0, 1, 0, 0, 1},

{0, 1, 0, 0, 0, 0},

{0, 0, 1, 0, 0, 0},

{0, 0, 0, 1, 0, 0},

{0, 0, 0, 0, 1, 0},

{0, 0, 0, 0, 0, 1}

};

 /\*vector<vector<bool>> matrix = {

     {1, 0, 1, 0, 0, 1},

     {0, 1, 0, 0, 0, 0},

     {1, 0, 1, 0, 0, 1},

     {0, 0, 0, 1, 0, 0},

     {0, 0, 0, 0, 1, 0},

     {1, 0, 1, 0, 0, 1}

 };\*/

 /\*vector<vector<bool>> matrix = {

     {0, 0, 1, 0, 0, 1},

     {1, 0, 0, 0, 0, 0},

     {0, 1, 0, 0, 0, 1},

     {1, 1, 1, 0, 0, 0},

     {1, 1, 1, 1, 0, 0},

     {0, 1, 0, 1, 1, 0}

 };\*/

 /\*vector<vector<bool>> matrix = {

     {0, 1, 1, 1, 1, 0},

     {0, 1, 1, 1,1, 0},

     {0, 1, 1, 1, 1, 0},

     {0, 1, 1, 1, 1, 0},

     {0, 1, 1, 1, 1, 0},

     {0, 1, 1, 1, 1, 0}

 };\*/

/\* vector<vector<bool>> matrix = {

     {0, 0, 0, 0, 0, 0},

     {0, 0, 0, 0, 0, 0},

     {1, 1, 1, 1, 1, 1},

     {1, 1, 1, 1, 1, 1},

     {0, 0, 0, 0, 0, 0},

     {0, 0, 0, 0, 0, 0}

 };\*/

 /\*vector<vector<bool>> matrix= {

     {1, 1, 1, 1, 1, 1},

     {0, 0, 1, 1, 1, 1},

     {0, 0, 0, 1, 1, 1},

     {0, 0, 0, 0, 1, 1},

     {0, 0, 0, 0, 0, 1},

     {0, 0, 0, 0, 0, 1}

 };\*/

/\* vector<vector<bool>> matrix = {

     {1, 0, 1, 1, 1, 0},

     {1, 1, 1, 1, 1, 1},

     {0, 0, 1, 1, 1, 1},

     {0, 0, 0, 1, 1, 1},

     {0, 0, 0, 0, 1, 1},

     {1, 0, 0, 0, 0, 1}

 };\*/

//  vector<vector<bool>> matrix = {

//      {0, 1, 1, 1, 1, 1},

//      {1, 0, 1, 1, 1, 1},

//      {1, 1, 0, 1, 1, 1},

//      {1, 1, 1, 0, 1, 1},

//      {1, 1, 1, 1, 0, 1},

//      {1, 1, 1, 1, 1, 0}

//  };

cout << "Исследуемая матрица 6x6:" << endl;

printMatrix(matrix);

cout << endl;

bool reflexive = isReflexive(matrix);

bool antireflexive = isAntireflexive(matrix);

bool symmetric = isSymmetric(matrix);

bool antisymmetric = isAntisymmetric(matrix);

bool asymmetric = isAsymmetric(matrix);

bool transitive = isTransitive(matrix);

bool connected = isConnected(matrix);

cout << "Результаты анализа:" << endl;

cout << "Рефлексивность: " << (reflexive ? "ДА" : "НЕТ") << endl;

cout << "Антирефлексивность: " << (antireflexive ? "ДА" : "НЕТ") << endl;

cout << "Симметричность: " << (symmetric ? "ДА" : "НЕТ") << endl;

cout << "Антисимметричность: " << (antisymmetric ? "ДА" : "НЕТ") << endl;

cout << "Асимметричность: " << (asymmetric ? "ДА" : "НЕТ") << endl;

cout << "Транзитивность: " << (transitive ? "ДА" : "НЕТ") << endl;

cout << "Связность: " << (connected ? "ДА" : "НЕТ") << endl;

cout << endl;

bool isEquivalence = reflexive && symmetric && transitive;

cout << "Типы отношений:" << endl;

cout << "Отношение эквивалентности: " << (isEquivalence ? "ДА" : "НЕТ") << endl;

PartialOrder(antisymmetric, transitive, reflexive, antireflexive, connected);

return 0;

}

bool isReflexive(const vector<vector<bool>>& matrix) {

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

if (matrix[i][i] == 0) {

return false;

}

}

return true;

}

bool isAntireflexive(const vector<vector<bool>>& matrix) {

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

if (matrix[i][i] == 1) {

return false;

}

}

return true;

}

bool isSymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (matrix[i][j] != matrix[j][i]) {

return false;

}

}

}

return true;

}

bool isAntisymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (matrix[i][j] && matrix[j][i]) {

return false;

}

}

}

return true;

}

bool isAsymmetric(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

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

if (matrix[i][j] && matrix[j][i]) {

return false;

}

}

}

return true;

}

bool isTransitive(const vector<vector<bool>>& matrix) {

int n = matrix.size();

vector<vector<bool>> R2(n, vector<bool>(n, false));

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

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

for (int k = 0; k < n; k++) {

if (matrix[i][k] && matrix[k][j]) {

R2[i][j] = true;

}

}

}

}

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

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

if (R2[i][j] && !matrix[i][j]) {

return false;

}

}

}

return true;

}

bool isConnected(const vector<vector<bool>>& matrix) {

int n = matrix.size();

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

for (int j = i + 1; j < n; j++) {

if (!matrix[i][j] && !matrix[j][i]) {

return false;

}

}

}

return true;

}

void printMatrix(const vector<vector<bool>>& matrix) {

for (auto row : matrix) {

for (bool element : row) {

cout << element << " ";

}

cout << endl;

}

}