Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Дискретная математика

«Определение свойств отношения»

Выполнил

Студент группы РИС-22-2б

Прядеин И.А.

Проверил доцент кафедры

ИТАС

Рустамханова Г. И.

Пермь 2023

**Постановка задачи:**

Дана матрица связей, состоящая из 6 строк и 6 столбцов, элементы которой являются 0 и 1.

Реализовать ввод матрицы из файла.

Программа должна определить все свойства связей исходной матрицы.

**Алгоритм работы:**

Считывание матрицы из файла происходит с помощью использования класса fstream. Создаётся объект с названием inputFile, если файл открыть не удалось, последующих действий по определению свойств не происходит. В противном случае с помощью метода std::getline() считывается строка. Происходит попытка преобразования из строкового типа в целочисленный. Каждый элемент считываемой строки записывается в матрицу.



Рис. 1 - Считывание матрицы из файл

Определение рефлексивности выполнено при помощи цикла с автоматическим перебором, проходящего по главной диагонали матрицы. В случае, если значение не равно 1, что означает отсутствие связи между одним и тем же элементом, функция isReflexive возвращает false.

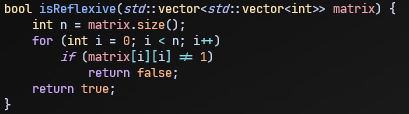


Рис. 2 - Функция определения рефлексивности

Определение антирефлексивности схоже функции определения рефлексивности. Однако, если элемент главной диагонали матрицы равен 1, функция возвращает false.

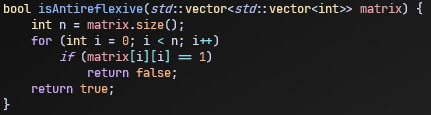


Рис. 3 - Функция определения антирефлексивности

Определение симметричности происходит путём перебора каждого элемента матрицы. В случае если связь между элементами двусторонняя, возвращается истина.

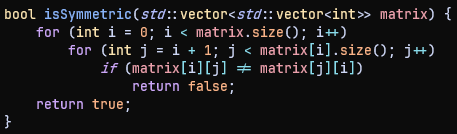


Рис. 4 - Функция определения симметричности

Свойство антисимметричности выполняется в том случае, если матрица антирефлексивна и связь между элементами, кроме элементов главной диагонали противоположная.

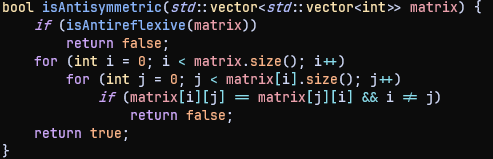


Рис. 5 - Определение антисимметричности

Определения свойства ассиметричности схоже с определение антисимметрии, только тем отличием, что отношение должно быть антирефлексивно, поэтому в начале функции отношение проверяется на наличие свойства антирфлексивности

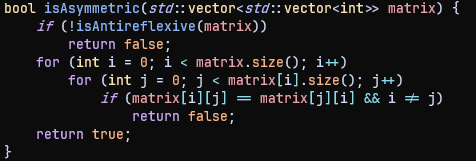


Рис. 6 - Определение свойства ассиметричности

Определение свойства транзитивности происходит с помощью трёх циклов. Первые два проходят по каждому элементу матрицы, и если находиться элемент равный единице, происходит проверка на наличие связи транзитивности с другими элементами матрицы.

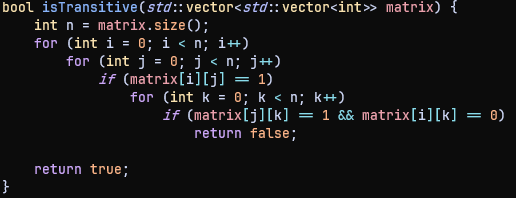


Рис. 7. - Определение свойства транзитивности.

Определение полноты (связности) происходит путём проверки на наличие связей между элементами, если в матрице присутствует пара не связанная между собой, функция возвращает false.

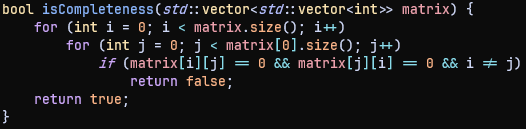


Рис. 8. - Определение свойства полноты

**Результат тестирования:**

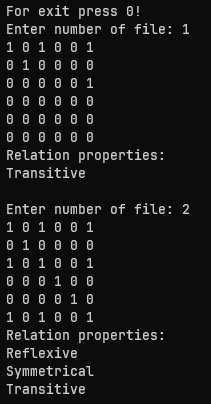


Рис. 9 - Результат тестирования 1 и 2 файлов

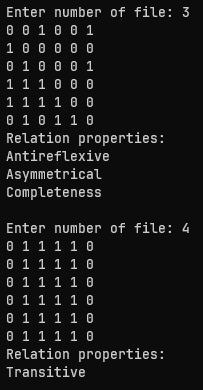


Рис. 10 - Результат тестирования 3 и 4 файла

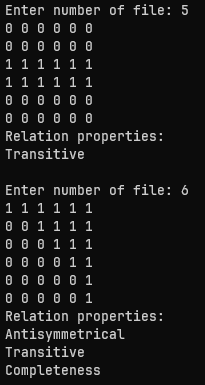


Рис. 11 - Результат тестирования 5 и 6 файлов

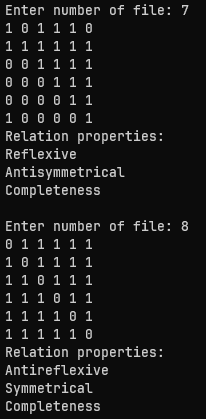


Рис. 12 - Результат тестирования 7 и 8 файлов

**Исходный код:**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

bool isReflexive(std::vector<std::vector<int>> matrix) {

int n = matrix.size();

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

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

return false;

return true;

}

bool isAntireflexive(std::vector<std::vector<int>> matrix) {

int n = matrix.size();

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

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

return false;

return true;

}

bool isSymmetric(std::vector<std::vector<int>> matrix) {

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

for (int j = i + 1; j < matrix[i].size(); j++)

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

return false;

return true;

}

bool isAntisymmetric(std::vector<std::vector<int>> matrix) {

if (isAntireflexive(matrix))

return false;

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

for (int j = 0; j < matrix[i].size(); j++)

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

return false;

return true;

}

bool isAsymmetric(std::vector<std::vector<int>> matrix) {

if (!isAntireflexive(matrix))

return false;

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

for (int j = 0; j < matrix[i].size(); j++)

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

return false;

return true;

}

bool isTransitive(std::vector<std::vector<int>> matrix) {

int n = matrix.size();

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

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

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

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

if (matrix[j][k] == 1 && matrix[i][k] == 0)

return false;

return true;

}

bool isCompleteness(std::vector<std::vector<int>> matrix) {

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

for (int j = 0; j < matrix[0].size(); j++)

if (matrix[i][j] == 0 && matrix[j][i] == 0 && i != j)

return false;

return true;

}

int main() {

int num;

std::cout << "For exit press 0!\n";

do {

std::vector<std::vector<int>> matrix(6, std::vector<int>(6));

std::string file;

std::cout << "Enter number of file: ";

std::cin >> num;

while (num <= 0 && num >= 9) {

std::cout << "Invalid number!\n";

std::cin >> num;

}

file = "m" + std::to\_string(num) + ".txt";

std::ifstream inputFile(file);

if (!inputFile.is\_open()) {

std::cout << "Unable to open file\n";

} else {

std::string line;

int i = 0;

bool flag = true;

while (std::getline(inputFile, line) && flag) {

int j = 0;

std::stringstream ss(line);

std::string temp;

while (ss >> temp) {

try {

int number = std::stoi(temp);

flag = number == 1 || number == 0;

if (!flag) {

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

break;

}

matrix[i][j] = number;

j++;

} catch (std::invalid\_argument& e) {

std::cerr << "Invalid argument: " << e.what() << "\n";

} catch (std::out\_of\_range& e) {

std::cerr << "Out of range: " << e.what() << "\n";

}

}

i++;

}

inputFile.close();

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

std::cout << matrix[i][j] << " ";

}

std::cout << "\n";

}

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

if (isReflexive(matrix))

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

if (isAntireflexive(matrix))

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

if (isSymmetric(matrix))

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

if (isAntisymmetric(matrix))

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

if (isAsymmetric(matrix))

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

if (isTransitive(matrix))

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

if (isCompleteness(matrix))

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

std::cout << "\n";

}

} while (num != 0);

system("pause");

return 0;

}