МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им.
Н. Э. Баумана

КАФЕДРА ПРОЕКТИРОВАНИЕ И ТЕХНОЛОГИЯ ПРОИЗВОДСТВА
ЭЛЕКТРОННОЙ АППАРАТУРЫ

Отчет о выполнении практического задания №3
«Теоретико-множественные операции»

по курсу «Функциональная логика и теория алгоритмов»

Выполнил: студент каф. ИУ4-21Б
Степанова Диана Станиславовна
Проверил:

Москва 2024

Цель работы

Написать на языке С и отладить программу, целью данной программы является анализ и обработка матрицы инцидентности графа для определения связности графа и создания визуализации графа в формате DOT.

1. Данные

Исходные данные
- Входные данные представлены матрицей инцидентности графа, считываемой из файла matrix_of_incendence%d.txt, где %d - номер матрицы.
- Максимальное количество строк матрицы: 5
- Максимальное количество столбцов матрицы: 40

Выходные данные
- Создание файла graph.dot, содержащего описание графа в формате DOT.
- Файл graph.png с визуализацией графа в формате PNG.

2. Выполнение

Программа реализована на языке С.

В коде определены следующие структуры:
- Edge - структура, представляющая ребро графа. Содержит указатели на вершины first и second, а также флаг observed.
- Vertex - структура, представляющая вершину графа. Содержит значение вершины value и флаг observed.
- Graph - структура, описывающая граф. Включает массив рёбер, массив вершин, матрицу инцидентности, размеры вершин и рёбер.

Функция get_size
Эта функция определяет количество строк и столбцов в файле с матрицей. Она читает посимвольно файл, подсчитывает количество символов до достижения символа новой строки и присваивает это значение основной переменной rows. Во время подсчета символов до достижения знака новой строки также подсчитывается количество столбцов. Таким образом, после выполнения функции, переменная rows содержит количество строк в файле, а переменная currentColumns - количество столбцов.

Функция ReadMatrixFromFile
Эта функция считывает матрицу смежности из файла. Сначала она открывает файл и использует функцию get_size, чтобы определить количество вершин и рёбер. Затем, она считывает числа из файла и сохраняет их в матрицу смежности.

Функция GetEdges
Функция определяет вершины, соединенные каждым ребром в графе, а также помечает их как наблюдаемые. Она проходит по всем рёбрам и для каждого ребра ищет пару вершин, связанных с этим ребром. Алгоритм ищет первую вершину, которая не была наблюдаема, связывает её с ребром и меткой о наблюдении, затем ищет вторую ненаблюденную вершину,

связывает её с этим ребром и меткой о наблюдении, после чего переходит к следующему ребру.

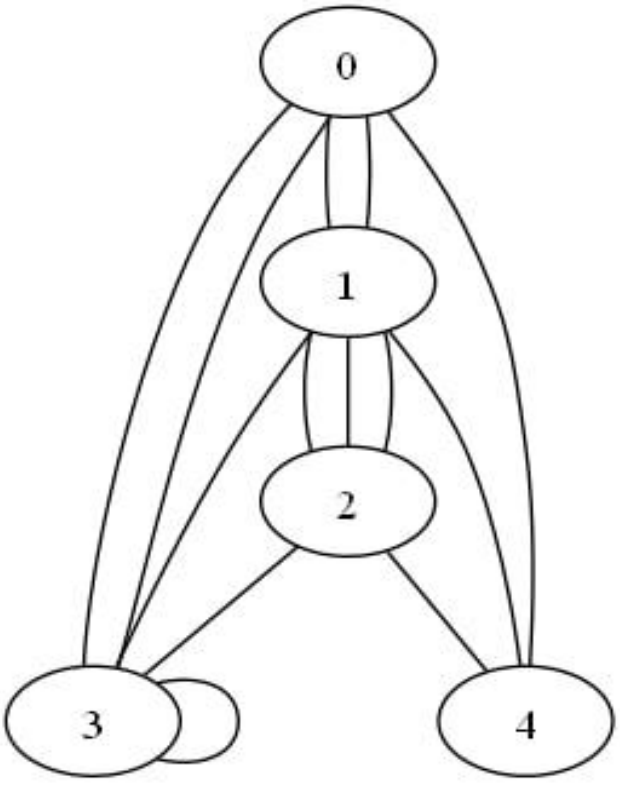
Функция WriteDotFile
Эта функция записывает граф в файл в формате DOT. Сначала функция открывает файл для записи. Затем она записывает пары вершин, соединенных ребрами. Далее она записывает одиночные вершины, которые не соединены рёбрами. В конце функция закрывает файл.

Функция IsConnected
Эта функция предназначена для определения связности графа на основе теоремы о рукопожатиях. Она вычисляет общую степень графа, сравнивает её с удвоенным количеством рёбер и определяет связность графа на основе этого сравнения.

Основная функция main:
- Считывает номер матрицы из пользовательского ввода.
- Читает матрицу из файла, инициализирует структуры графа.
- Определяет рёбра графа, проверяет связность.
- Записывает описание графа в файл DOT и создаёт изображение графа.

3. Результаты работы.

45 матрица инцидентности



4. Вывод

В результате выполнения программы была проведена обработка матрицы инцидентности графа для определения его связности и создания визуализации в формате DOT. Полученные результаты могут быть использованы для анализа структуры графов и их визуального представления.

5. Код

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_ROWS 5
#define MAX_COLS 40

struct Edge {
    struct Vertex * first, * second;
    char observed;
};
struct Vertex{
    int value;
    char observed;
};
struct Graph{
    struct Edge edges[MAX_COLS];
    struct Vertex vertices[MAX_ROWS];
    int matrix[MAX_ROWS][MAX_COLS], vertices_size, edges_size;
};

void get_size(int* rows, int* currentColumns, FILE *file) {
    int columns = 0;
    char tmp, ch;
    while (fscanf(file, "%c", &ch) == 1) {
        if (ch != '\n' && ch != ' ') {
            if (*rows == 0) columns++;
            tmp = ch;
        } else if (ch == '\n') {
            if (*rows == 0) columns++;
            (*rows)++;
            if (*rows == 1)*currentColumns = columns;
            columns = 0;
        }
    }
}
void ReadMatrixFromFile(const char* inputFilename, int matrix_of_incidence[MAX_ROWS][MAX_COLS], int* vertices_size, int* edges_size) {
    FILE *file = fopen(inputFilename, "r");
    get_size(vertices_size, edges_size, file);
    (*edges_size)--;
    fseek(file, 0, SEEK_SET);
    for (int i = 0; i < (*vertices_size); i++)
        for (int j = 0; j < (*edges_size); j++) fscanf(file, "%d", &matrix_of_incidence[i][j]);
    fclose(file);
}
void GetEdges(struct Graph * graph) {
    for (int j = 0; j < graph->edges_size; j++) {
        char loop = 1;
        for (int i = 0; i < graph->vertices_size; i++) {
            if (graph->matrix[i][j] && !graph->edges[j].observed) {
                graph->edges[j].first = &graph->vertices[i];
                graph->vertices[i].value = i;
                graph->edges[j].observed = 1;
                graph->vertices[i].observed = 1;
            }
            else if (graph->matrix[i][j]) {
                graph->edges[j].second = &graph->vertices[i];
                graph->vertices[i].observed = 1;
                graph->vertices[i].value = i;
                loop = 0;
                break;
            }
        }
        if (loop) graph->edges[j].second = graph->edges[j].first;
    }
}
void WriteDotFile(struct Graph *graph) {
    FILE* writer = fopen("graph.dot", "w");
    fprintf(writer, "graph {\n");
    for (int i = 0; i < graph->edges_size; i++) fprintf(writer, "%d -- %d;\n", *graph->edges[i].first, *graph->edges[i].second);
    for (int i = 0; i < graph->vertices_size; i++)
        if (!graph->vertices[i].observed) fprintf(writer, "%d;\n", i);
    fprintf(writer, "}\n");
    fclose(writer);
}
int IsConnected(struct Graph *graph) {
    int totalDegree = 0;
    for (int i = 0; i < graph->vertices_size; i++) {
        int vertexDegree = 0;
        for (int j = 0; j < graph->edges_size; j++)
            if (graph->edges[j].first->value == i || graph->edges[j].second->value == i) vertexDegree++;
        totalDegree += vertexDegree;
    }
    if (totalDegree == 2 * graph->edges_size) return 1
    else return 0; // Graph is not connected
}

int main() {
    int file_number;
    char inputFilename[200];
    printf("Enter the number of matrix\n>> ");
    scanf("%d", &file_number);
    sprintf(inputFilename, "tests/matrix_of_incendence%d.txt", file_number);
    struct Graph graph = {.edges_size = 0, .vertices_size = 0};
    ReadMatrixFromFile(inputFilename, graph.matrix, &graph.vertices_size, &graph.edges_size);
    for (int i = 0; i < graph.edges_size; i++) graph.edges[i].observed = 0;
    for (int i = 0; i < graph.vertices_size; i++) graph.vertices[i].observed = 0;
    GetEdges(&graph);
    IsConnected(&graph);
    WriteDotFile(&graph);
    system("dot -Tpng graph.dot -o graph.png");
    return 0;
}
```