Министерство образования и науки Российской Федерации

ФГБОУ ВО Московский государственный технологический университет «СТАНКИН»

**Отчет по лабораторной работе №1**

По дисциплине: «Теория графов и тензорное счисление».

Вариант №1

Выполнил: студент группы ИДБ-15-15 Павлов А.А.

Проверил: преподаватель Назаренко К.М.

**Москва 2016**

**Задание №1**

На вход в программу подаётся файл «matrix.txt» , содержащий граф в виде матрицы инцидентности. Ввиду возможности присутствия в файле некорректных данных производится их проверка.В результате выполнения программы необходимо вывести матрицу смежности и список рёбер,построенные по исходной матрице инцидентности,в два отдельных файла.

В программе создаётся класс Graph,содержащий определённый набор методов и полей,с помощью некоторых из которых решаются как вопросы,напрямую касающиеся графа,так и второстепенные задачи получения матрицы из файла и проблемы взаимодействия с другими основообразующими классами.

class Graph

{

public:

Graph(string s): is\_tree(true),is\_ograph(false), full\_path (s.substr(0, s.find\_last\_of("\\/") + 1)) {}

void process\_graph();

void process\_tree();

private:

//Считать из файла матрицу инцидентности

bool get\_matrix();

//Проанализировать полученную из файла матрицу инцидентности

bool analyze();

//Вывести матрицу инцидентности на экран

void output\_matrix();

//Заполнить матрицу смежности из матрицы инцидентности

void fill\_adj\_matrix();

//Добавить ребро в матрицу смежности

void ins\_edge\_to\_adj\_matrix(Edge &edge, bool is\_oedge);

//Вывести матрицу смежности в файл и на экран

void output\_adj\_matrix();

//Заполнить массив ребер из матрицы смежности

void fill\_adj\_arr();

//Вывести массив ребер в файл и на экран

void output\_adj\_arr();

//Матрица смежности графа

Tmatrix adj\_matrix;

//Матрица инцидентности графа

Tmatrix matrix;

//Массив ребер графа

vector <Edge> edg;

bool is\_tree, is\_ograph;

size\_t vertex\_qty, edge\_qty;

Tree tree;

// Наименование файла данных

const string matrix\_fname = "matrix.txt";

const string adj\_matrix\_fname = "adj\_matrix.txt";

const string edges\_fname = "edges.txt";

// Каталог запуска приложения

string full\_path;

// Полное наименование файла данных

string full\_fname ;

};

Считывание файла производится с помощью метода Graph.get\_matrix(). При отсутствии файла matrix.txt в каталоге программы выводится сообщение об отсутствии этого файла. Дополнительно производится проверка на наличии данных в этом файле. При их отсутствии выводится соответствующее сообщение об ошибке. После считывания данных на экран выводится информация о количестве вершин и ребер графа.

Проверка корректности считанной из файла матрицы производится при помощи метода Graph.analyze() , принадлежащего классу Graph. В ходе осуществления анализа производится проверка корректности данных по количеству связей между инцидентными элементами графа. В случае неверно заданной матрицы на экране выводятся следующие сообщения в зависимости от типа ошибки:

1. При наличии недопустимых чисел в матрице инцидентности, а именно присутствии других чисел кроме 1(вершина выхода ребра), -1(вершина входа ребра) и 2(петля):

“Недопустимое число <значение элемента> в строке <номер строки> столбце <номер столбца> матрицы ”

1. При одновременном присутствии единиц и двоек в столбце:

"Недопустимо одновременное присутствие единиц и двоек в столбце <номер столбца>

1. При наличии только нулевых элементов в столбце:

"Недопустимо одновременное отсутствие единиц и двоек в столбце <номер столбца>”

Дополнительно, по наличию элементов со значениями -1, определяется является ли данный граф ориентированным, а по соотношению количества вершин и ребер и проверке связности вершин - является ли граф деревом. Результаты анализа выводятся соответствующими сообщениями на экран. Затем на экран выводится матрица инцидентности, считанная из файла matrix.txt.

Входные данные. Матрица инцидентности:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 |
| -1 | 0 | 1 | 1 | 1 |
| 0 | -1 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | -1 |

Следующим этапом является построение матрицы смежности по связям вершин из матрицы инцидентности и вывод ее на экран и в файл adj\_matrix.txt методами Graph.fill\_adj\_matrix и Graph.output\_adj\_matrix соответственно.

Матрица смежности:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Затем строится по связям вершин из матрицы инцидентности список ребер графа методом Graph.fill\_adj\_arr и также выводится на экран и в файл edges.txt методом Graph.output\_adj\_arr.

Список ребер графа:

0--1

0--2

2--3

1--4

1—5

Если граф является деревом, то производится построение дерева в виде связанного списка методом Graph.from\_incidence, и вывод его на экран с указанием номеров вершин и связей между ними методом Graph.print.

Вывод полного дерева:

0

|

+-1

| |

| +-4

| |

| +-5

|

+-2

|

+-3

Затем выводится список вершин, глубже заданной, в виде дерева методом Graph.print(<номер заданной вершины>). В данном примере заданной является вершина с номером 1.

Вывод списка вершин глубже первой в виде дерева

1

|

+-4

|

+-5

**Задание №2**

Заданием №2 является вывод на экран построенного дерева из строки представления в виде 1 и 0, указывающих обход вершин в глубину и в обратном направлении соответственно, а, также, построение строки представления из этого же дерева.

Системоообразующими классами в этом задании является уже упомянутый ранее класс Graph,а также Tree и TreeNode,во взаимодействии которых кроется осуществление таких задач,как проход по дереву,его очистка,получение дерева из кодированной нулями и единицами строки и преобразование дерева в неё.

class TreeNode

{

public:

TreeNode() {}

TreeNode(int v\_no):vertex\_no(v\_no) {}

int parents = 0;

void print(string str) const;

void to\_binary(string& values);

bool add\_child(TreeNode \*child);

void clear\_tree(TreeNode \*node);

TreeNode\* find(int vert\_no);

private:

vector<TreeNode \*> children;

int vertex\_no = 0;

};

class Tree

{

public:

TreeNode \*root = nullptr;

void print() const;

void print(int vert\_no) const

{

TreeNode \*r = root->find(vert\_no);

if (r)

r->print("");

}

bool from\_binary(string values);

void to\_binary(string& values);

bool from\_incidence(Tmatrix &);

void clear\_data();

};

inline void Tree::print() const

{

if (root)

root->print("");

}

Построение дерева осуществляется методом Tree.from\_binary , а вывод методом Graph.print.

Вывод дерева , построенного из строки 1101010010

0

|

+-1

| |

| +-2

| |

| +-3

| |

| +-4

|

+-5

Затем построенное дерево выводится обратно в строку представления методом Tree. to\_binary и эта строка выводится на экран:

Вывод строки представления дерева

1101010010

**Задание №3**

Целью задания №3 является построение и анализ шахматной позиции на шахматной доске размером 4x4 из трех фигур-белых короля и ферзя, а, также, черного короля. Координаты фигур задаются в файле chess.txt в виде шахматной нотации, то есть латинскими буквами A,B,C,D и цифрами от 1 до 4.

Для выполнения задания разработан класс Tchess с соответствующими методами.

В программе наличествуют два перечисления – Tchessman – описывающее фигуры на шахматной доске и Tattack\_result,называющее возможные исходы игровых ситуаций.

enum Tchessman {w\_king, w\_queen, b\_king, last};

enum Tattack\_result {CHECK, CHECKMATE, STALEMATE, NOTHING};

Структура Tpos задаёт параметры, характеризующие положение фигур на доске, а основной класс подпрограммы – Tchess – содержит набор основных методов,а также выполняет работу по получению данных из фа йла.

typedef short int Tint;

struct Tpos {

Tint y, x;

bool operator ==(const Tpos &other) const

{

if ((y == other.y) && (x == other.x))

return true;

else

return false;

}

bool operator !=(const Tpos &other) const

{

if ((y != other.y) || (x != other.x))

return true;

else

return false;

}

Tpos(Tint p\_y = 0,T Tint p\_x = 0): y(p\_y), x(p\_x) {}

};

Узел дерева позиций после очередного хода белых или чёрных

typedef std::vector<std::vector<Tpos> > Tmoves\_matrix;

class Node

{

public:

Node(Tpos \*pos\_arr, Tattack\_result p\_status = NOTHING, Node \*p\_parents = nullptr, Tchessman p\_chessman = LAST):

parents(p\_parents), status(p\_status), chessman(p\_chessman)

{

for (Tint i = 0; i < CHESSMAN\_QTY; ++i)

after\_move\_pos[i] = pos\_arr[i];

}

std::vector<Node \*> children;

//Массив позиций фигур после очередного хода белых или черных

Tpos after\_move\_pos[CHESSMAN\_QTY];

Node \*parents;

Tattack\_result status;

//Фигура, сделавшая ход

Tchessman chessman;

};

class Tchess

{

// Каталог запуска приложения

std::string full\_path;

// Полное наименование файла данных

std::string full\_fname ;

// Шахматная доска

Tint board[BOARD\_SIZE][BOARD\_SIZE];

// Указатель на корневой элемент дерева положений

Node \*root;

// bool set\_pos(Tchessman ch\_man , Tpos pos);

void set\_pos(Tpos \*pos\_arr, Tchessman ch\_man , std::string XY);

void check\_pos(Tpos \*pos\_arr) const;

void analyze(Tpos \*p\_arr, Tmoves\_matrix &moves\_arr, Tattack\_result &res);

void show\_board(Tpos \*pos\_arr) const;

// void print\_board() const;

void add\_node(Node \*parent, Tint p\_level, bool white);

Node \* search\_checkmate\_pos(Node \*ptr);

public:

Tchess(std::string s);

void process\_chess();

struct TException {

std::string mess;

TException(std::string mess\_): mess(mess\_) {}

};

};

На экран выводится начальное задание в виде:

Задано:

Позиция белого короля: A4

Позиция белого ферзя: C4

Позиция черного короля: A1

Начальная позиция:

A B C D

4: БК + БФ +

3: + + + +

2: + 0 + 0

1: ЧК 0 + 0

В конструкторе создается и заполняется корневой узел дерева позиций, производится расстановка фигур на шахматной доске в соответствии с заданными координатами с проверкой их на правильность задания в конструкторе класса методом Tchess.set\_pos и проверкой начальной позиции методом Tchess.check\_pos.

Проверяется:

1. Нахождение фигур в пределах доски.
2. Несовпадение позиций двух и более фигур.
3. Нахождение черного короля не под боем белых фигур.

Потом производится анализ начальной позиции, заполняется массив возможных ходов белых фигур, а также массив ссылок на узлы дерева позиций, отображающие ответные ходы черным королем на каждый ход белых.

Затем вызывается метод Tchess.process\_chess, в котором рекурсивно вызывается метод Tchess.add\_node для построения узлов дерева позиций, представляющих собой ходы белых фигур и ответные ходы черного короля.

В методе Tchess.add\_node анализируется текущая позиция фигур, заполняется массив возможных ходов белых фигур и разрешенных ответных ходов черного короля, а также заполняется массив ссылок на узлы дерева позиций, отображающие ответные ходы черным королем на каждый ход белых.

Количество уровней дерева ограничено заданной в тексте программы глубиной построения.

Ключевым этапом выполнения задания является поиск матовой позиции методом Tchess.search\_checkmate\_pos, возвращающим ссылку на найденный узел или пустую ссылку в случае неудачного поиска. Поиск осуществляется проходом по дереву в ширину, что обеспечивает нахождение наиболее короткого пути, приводящего к матовой позиции, проходом по ссылкам от найденного узла к вершине дерева.

Заключительным этапом выполнения задания является вывод перечня позиций фигур, названия ходящей фигуры, ее хода и игровой ситуации.

<БК A4;БФ C4;ЧК A1>; Начальная позиция;

<БК B3;БФ C4;ЧК A1>; ход БК B3;

<БК B3;БФ C4;ЧК B1>; ход ЧК B1;

<БК A3;БФ C4;ЧК B1>; ход БК A3;

<БК A3;БФ C4;ЧК A1>; ход ЧК A1;

<БК A3;БФ A2;ЧК A1>; ход БФ A2; Мат

После этого выводится шахматная доска с конечной матовой позицией фигур (БК-белый король, БФ-белый ферзь, ЧК-черный король) и обозначением полей доски под боем белых фигур символом +, а не под боем символом 0.

Конечная позиция

A B C D

4: + + + 0

3: БК + 0 0

2: БФ + + +

1: ЧК + 0 0