

# **Алгоритмы и структуры данных**

## *1.2.1 Перекрестные ссылки. Бинарное дерево поиска «Описание спецификации программы»*

Студент гр. 3530904/90002

Ф.И.О Мэн Цзянин

## 1. Общая постановка задачи

- 1) Для разрабатываемого словаря реализовать основные операции:
  - INSERT (ключ, значение) – добавить запись с указанным ключом и значением
  - SEARCH (ключ)- найти запись с указанным ключом
  - DELETE (ключ)- удалить запись с указанным ключом
- 2) Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием.
- 3) Программа должна быть написана в соответствии со стандартом программирования: C++  
Programming Style Guidelines (<http://geosoft.no/development/cppstyle.html>).
- 4) Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

## 2. Требования

Разработать и реализовать алгоритм формирования перекрестных ссылок:

- прочитать текст и вывести его с добавлением последовательных номеров строк;
- собрать все слова, встречающиеся в тексте;
- сформировать таблицу, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк его нахождения (по возрастанию номеров строк)

Для реализации задания использовать бинарное дерево поиска, узел которого может содержать:

- Ключ – слово
- Информационная часть – ссылка на список, содержащий номера строк

## 3. Спецификация

- 1) Файл должен быть открыт
  - i. В противном случае выводим ошибку, содержащую текст:  
«cant\_open\_the\_file»
- 2) root, который в бинарном дереве поиска нельзя было nullptr
- 3) все данные должны иметь правильный тип
  - i. Поля Номер строки(row), столбца(col), счетчик(count) и длина слова(length) должны иметь тип данных unsigned int
  - ii. Номер строки, столбца и длина слова должны больше чем 1
- 4) Если пустая строка пропускается, счетчик не накапливается

5) Для повторяющихся слов

- i. В одной статье может быть несколько повторяющихся слов, и номера строк и столбцов этих слов должны храниться в определенном контейнере
- ii. При печати функции(`printWordAndInfo`), которая выводит слова, должен быть предоставлен внешний интерфейс, чтобы пользователь мог выбрать, следует ли печатать повторяющиеся слова
  - a) Если `«bool repetitive_word = true»`: выводит все повторяющиеся слова вместе с их информацией
  - b) Если `«bool repetitive_word = false»`: выводит это слово, номер строк и столбцов, которые появляются в первый раз
- б) Для одного и того же слова, но с большой буквы или с “ ’s ”. Все буквы должны быть преобразованы в маленькую букву.

**Пример:**

|           |   |         |
|-----------|---|---------|
| Hello     | → | hello   |
| World     | → | world   |
| student's | → | student |
| Student's | → | student |

Рис.1

## 4. Структура данных

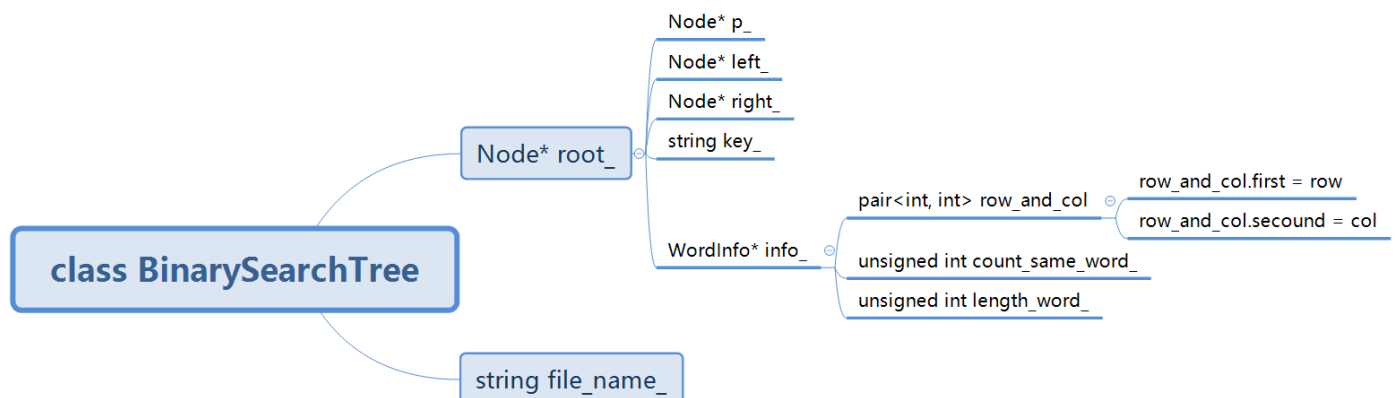


Рис.2

## 5. Интерфейс

- Сохранить имя файла, которое пользователь вводит

```
void openFile(string file_name);
```

- Выводит текст и вывести его с добавлением последовательных номеров строк в файл

```
void printTextWithRowNum();
```

- Найти запись с указанным ключом

```
void searchWord(string word_search);
```

- Удалить запись с указанным ключом

```
void deleteWord(string dele_word);
```

- Собрать все слова, встречающиеся в тексте

```
void makeTree();
```

- Печать на экране

```
void outputDictionaryOrderInFile(bool repetitive_word);
```

- true == repetitive\_word

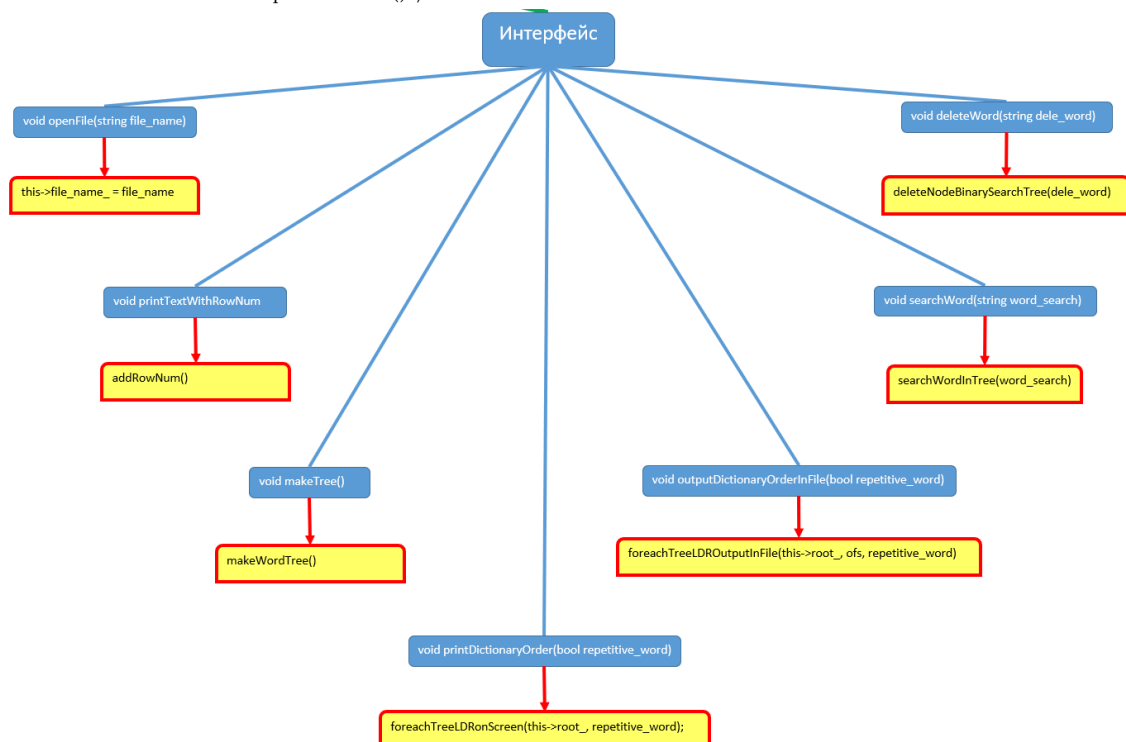
вывод повторяющихся слов

- false == repetitive\_word

не вывод повторяющихся слов

- Выводить в файл

```
void outputInFile();
```



## 6. Методы

```
➤ void insertNode(Node* nodeInsert, unsigned int count_word_text);
```

Каждое слово и его информации (row, col, count, length) – это один узел в бинарном дереве поиска. Функция этой функции заключается в вставке узлов в двоичное дерево поиска.

➤ **void makeWordTree();**

С помощью предыдущей функции (void insertNode(Node\* nodeInsert, unsigned int count\_word\_text);), эта функция вставляет все слова из статьи в двоичное дерево поиска по лексикографическому порядку. После преобразования слова в нижний регистр и удалил “:”, “.”, “’s”.

Пример:

*“This Prestwick House Literary Touchstone Edition includes a glossary and reader’s notes to help the modern reader fully appreciate London’s masterful weaving of science, philosophy, and the storyteller’s”*

Бинарное дерево поиска должно:

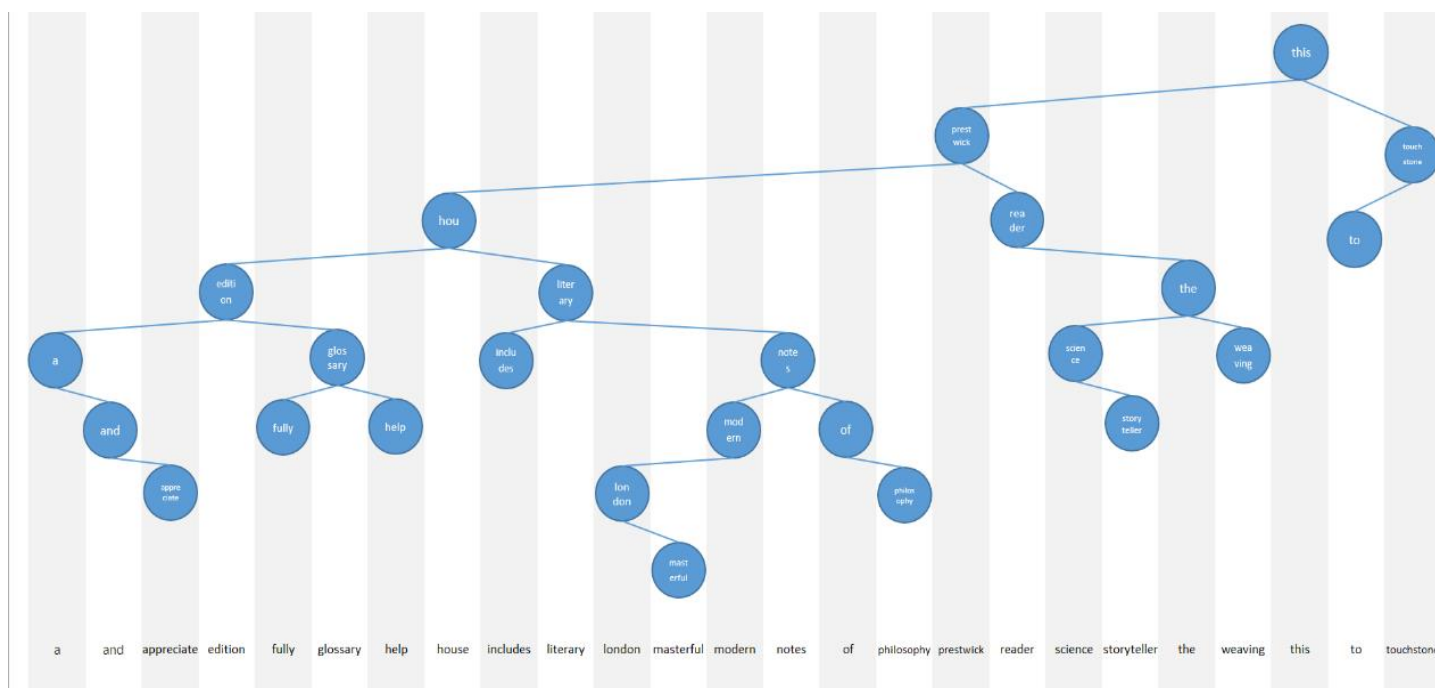


Рис. 3 бинарное дерево поиска.

➤ **void addRowNum();**

Эта функция считывает статью из файла и выводит статью с добавленным номером строки в output.txt. Кроме того, она также подсчитывает общее количество слов в статье, количество строк.

```

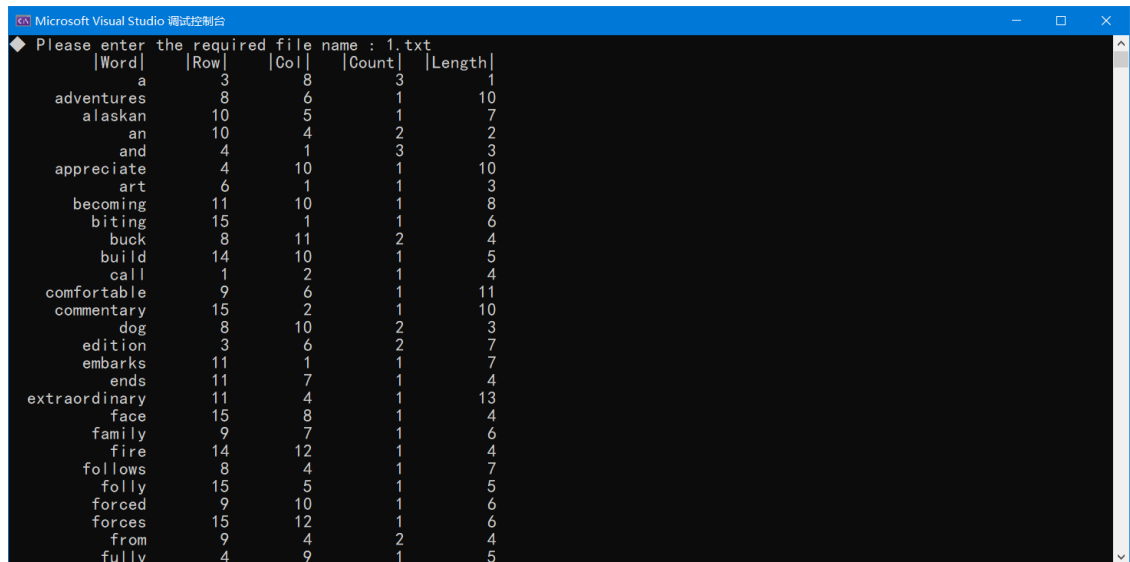
1 - The Call of the Wild 【Words in this row : 5】
2 -
3 - This Prestwick House Literary Touchstone Edition includes a glossary 【Words in this row : 9】
4 - and reader's notes to help the modern reader fully appreciate 【Words in this row : 10】
5 - London's masterful weaving of science, philosophy, and the storyteller's 【Words in this row : 9】
6 - art. 【Words in this row : 1】
7 -
8 - This gripping story follows the adventures of the loyal dog Buck, 【Words in this row : 11】
9 - who is stolen from his comfortable family home and forced into the 【Words in this row : 12】
10 - harsh life of an Alaskan sled dog. Passed from master to master, Buck 【Words in this row : 13】
11 - embarks on an extraordinary journey that ends with his becoming the 【Words in this row : 11】
12 - legendary leader of a wolf pack. 【Words in this row : 6】
13 -
14 - Included in this Edition is the short story, To Build a Fire, London's 【Words in this row : 13】
15 - biting commentary on human folly in the face of indomitable natural forces. 【Words in this row : 12】
=====
Amount of words in the article : 112
Amount of row in the article : 15
=====

```

Рис.4 метод void addRowNum()

➤ **void foreachTreeLDRonScreen(Node\* root, bool repetitive\_word);**

Сформировать таблицу и печать на экране, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк, столбец его нахождения (по возрастанию номеров строк), количество одинаковых слов и количество букв в слове. В аргументе функции **bool repetitive\_word** – это выводит ли повторные слова.



| Word          | Row | Col | Count | Length |
|---------------|-----|-----|-------|--------|
| a             | 3   | 8   | 3     | 1      |
| adventures    | 8   | 6   | 1     | 10     |
| alaskan       | 10  | 5   | 1     | 7      |
| an            | 10  | 4   | 2     | 2      |
| and           | 4   | 1   | 3     | 3      |
| appreciate    | 4   | 10  | 1     | 10     |
| art           | 6   | 1   | 1     | 3      |
| becoming      | 11  | 10  | 1     | 8      |
| biting        | 15  | 1   | 1     | 6      |
| buck          | 8   | 11  | 2     | 4      |
| build         | 14  | 10  | 1     | 5      |
| call          | 1   | 2   | 1     | 4      |
| comfortable   | 9   | 6   | 1     | 11     |
| commentary    | 15  | 2   | 1     | 10     |
| dog           | 8   | 10  | 2     | 3      |
| edition       | 3   | 6   | 2     | 7      |
| embarks       | 11  | 1   | 1     | 7      |
| ends          | 11  | 7   | 1     | 4      |
| extraordinary | 11  | 4   | 1     | 13     |
| face          | 15  | 8   | 1     | 4      |
| family        | 9   | 7   | 1     | 6      |
| fire          | 14  | 12  | 1     | 4      |
| follows       | 8   | 4   | 1     | 7      |
| folly         | 15  | 5   | 1     | 5      |
| forced        | 9   | 10  | 1     | 6      |
| forces        | 15  | 12  | 1     | 6      |
| from          | 9   | 4   | 2     | 4      |
| fully         | 4   | 9   | 1     | 5      |

Рис.5 Сформировать таблицу и печать на экране

- **void foreachTreeLDROutputInFile(Node\* root, ofstream& ofs, bool repetitive\_word);**

Сформировать таблицу и выводить в файл, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк, столбец его нахождения (по возрастанию номеров строк), количество одинаковых слов и количество букв в слове. В аргументе функции **bool repetitive\_word** – это выводит ли повторные слова

| Word       | Row | Col | Count | Length |
|------------|-----|-----|-------|--------|
| a          | 3   | 8   | 3     | 1      |
| adventures | 8   | 6   | 1     | 10     |
| alaskan    | 10  | 5   | 1     | 7      |
| an         | 10  | 4   | 2     | 2      |
| and        | 4   | 1   | 3     | 3      |
| appreciate | 4   | 10  | 1     | 10     |

....

Пример

|            |    |   |   |    |
|------------|----|---|---|----|
| touchstone | 3  | 5 | 1 | 10 |
| weaving    | 5  | 3 | 1 | 7  |
| who        | 9  | 1 | 1 | 3  |
| wild       | 1  | 5 | 1 | 4  |
| with       | 11 | 8 | 1 | 4  |
| wolf       | 12 | 5 | 1 | 4  |

Рис.6

- **void changeCaseAndClearSign(string& word);**

Для одного и того же слова, но с большой буквы или с “ ’s ” или на конце слова имеет “ ,”, “ .”. Эта функция может преобразовать все буквы в маленькую букву и удалить “ ,”, “ .”.

|          |   |         |
|----------|---|---------|
| This     | → | this    |
| reader's | → | reader  |
| London's | → | london  |
| science, | → | science |
| art.     | → | art     |

Рис. 7

- **bool isHaveSameWord(string word, pair<unsigned int, unsigned int> row\_col\_word)**

Эта функция предназначена для определения того, существует ли одно и то же слово в двоичном дереве поиска при вставке нового слова. Если есть однородное слово **return true**, если нет **return false**.

- **Node\* searchWordInTree(string word\_search);**

Поиск, существует ли введенное пользователем слово.

Пример:

- Искать fire

```
=====Search Word=====
Word : fire
Row : 14
Col : 12
```

Рис.8 искать fire

- Искать student

```
=====Search Word=====
Don't have word 【student】!
```

Рис.9 искать student

- **string& clearAllSpace(string& str);**

Чтобы удаление пробелов в строке, при вычислении длины строки.

- **bool deleteNodeBinarySearchTree(string dele\_key);**

Удаляет узлы двоичного дерева поиска на основе введенных пользователем слов.

Пример:

- удалить file

```
=====Delete Word=====
Succeed delete word 【fire】!
```

Рис.10 удалить file

- удалить student

```
4=====Delete Word=====
4
4Delete fail, don't have word 【student】!
4
```

Рис. 11 удалить students



## 7. Для некорректного ввода данных:

| Спецификация | Ожидаемый вывод                           |  | Данные          | Ожидаемый вывод      |  | Операция                   | Ожидаемый вывод                    |  |
|--------------|---|--|-----------------|----------------------|--|----------------------------|------------------------------------|--|
| 2.t          | ERROR<br>---<br>cant_open_the_file<br>--- |  | root_ = nullptr | ERROR - nullptr_root |  | Неправильный выбор         | No corresponding operation !       |  |
| 1. txt       |   |  |                 |                      |  | Нет инициализации          | no_data                            |  |
| 1.c          |   |  |                 |                      |  | Неверные данные при search | Don't have word [...] !            |  |
| 1txt         |   |  | word in null    | ERROR - null_word    |  | Неверные данные при delete | delete fail, don't have word [...] |  |
| 1            |   |  |                 |                      |  |                            |                                    |  |
| txt          |   |  |                 |                      |  |                            |                                    |  |

Таблица 1