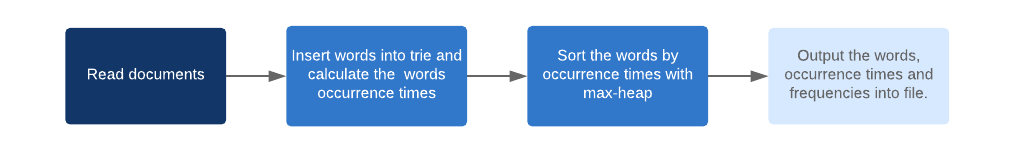
Words Pattern Extraction Design

# **Abstract**

The purpose of this program is to extract the most commonly used words combinations in the language. The following figure briefly describes the execution flow of the program. The input languages are English and Chinese.



1.1 the execution flow of the program

# **English Text**

In order to extract the English words patterns, firstly, we need to find the common words. Secondly, we find the common words patterns according to the order that common words appear in sentences.

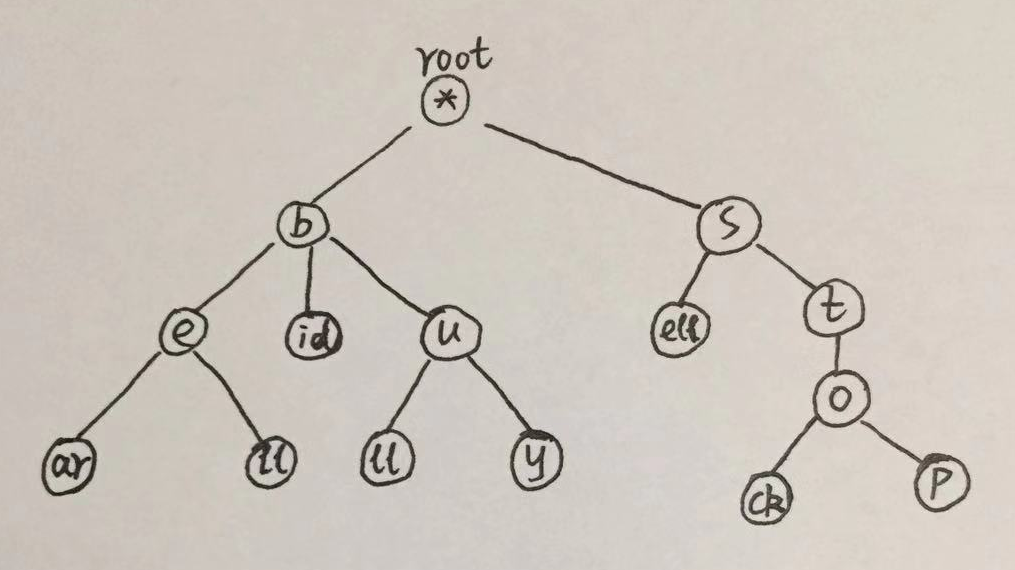
## **Process Design**

When our program starts, it reads the documents in the folder named "file" in current directory. The program finds common words and words patterns based on these documents.

Common Words:

A common word is a word that frequently occurring in the texts. In our program, the number of occurrences time can be set to determine the range of common words.

Step 1: Use a single word as input unit. Use the compressed trie structure to store the words and words occurrence times information. Read as many documents as possible in different categories to ensure better accuracy of word frequency.



2.1 The structure of a compressed trie

Step 2: Traverse the trie and use max-heap structure to sort the words based on the occurrence times of the words.

Step 3: Output the sorted words information and at the same time calculate the occurrence frequency of each word.

Words Patterns:

A word pattern is frequently occurring adjacent or non-adjacent common words group in the sentences. In our program, we separate adjacent words with spaces and non-adjacent words with ‘#’, and common words with distances less than or equal to 5 are considered in the same pattern. Otherwise, two common words separated by more than 5 non-common words are considered in two different patterns.

For finding words patterns, we use a similar design like it finds common words.

Step 1: Extract top x (eg: 1000) words from the file which created in the third step of common words as common words, and store these common words in a compressed trie, so that it can judge if a word is common words when finding words patterns.

Step 2: Use a *sentence* as input unit. Use the standard trie to store the sequence of common words that appear in the sentences and record the occurrence times. Read as many documents as possible to make it as accurate as possible. (*Note: The sentence here is a continuous sequence words and it is separated by comma, period, colon and semicolon etc. rather than a sentence ending by a period in the language.*)

Step 3: Traverse the trie and use max-heap structure to sort the words patterns based on the occurrence times of them.

Step 4: Output the sorted words patterns information and at the same time calculate the occurrence frequency of them.

## **Structure Design**

**trie\_node**

struct trie\_node

{

std::string key;

trie\_node\* child[Max]; //link list of children

unsigned int freq; //Occurrence times. Only when it big than 0, the node is the end of a word

bool feature; //true: this node has more than one characters; false: this node has only one character

};

Trie node structure for storing words.

**sentence\_trie\_node**

struct sentence\_trie\_node

{

std::string key;

sentence\_trie\_node\* child[MAX\_SEN]; //link list of children

unsigned int freq; //pattern occurrence times

bool feature; //true: multi-character; false: single-character

};

Trie node structure for storing words patterns.

**heap\_node**

struct heap\_node

{

std::string key;

unsigned int freq;

heap\_node(std::string k, unsigned int f) {key = k; freq = f;}

};

Max-Heap node structure for sorting words and word patterns.

**trie\_single**

class trie\_single

{

private:

trie\_node\* T\_root;

void delNode(trie\_node\* node); //Delete nodes

trie\_node\* createNode(std::string key); //Create a new node

int calIndex(char letter); //Calculate the index of the insert key

void output\_d(trie\_node\* node, std::string output\_word); //Display the trie

void output\_tr(trie\_node\* node, max\_heap\* cMax\_Heap, std::string output\_word); //Output the tire into max-heap for sorting

public:

trie\_single();

~trie\_single();

public:

void insert(std::string key);

unsigned int search(std::string key); //Return occurrence time of the word

void traverse(max\_heap\* cmax\_heap); //Output the tire into max-heap for sorting

void display();

};

Common words class structure

**word\_freq**

class word\_freq

{

private:

std::vector<std::string> vfname; //Texts name will be read

unsigned int words\_num; //The total number of words has read

bool isVaild(char\* con); //Judge if the charcter is an English letter

public:

word\_freq();

~word\_freq();

//The function that should be called for getting the words information. out\_fname: the name of the result file

void cal\_words\_freq(std::string out\_fname = "word\_freq.txt");

void insert\_words(std::string fname, trie\_single \*const ctrie\_single); //Insert words into trie

void write\_to\_file(std::string out\_fname, max\_heap \*const cmax\_heap); //Write the words information into a file

//Get the common words from file "fname", save the words into "ctrie\_single", "limit\_count" is the smallest count of words

     void get\_common\_words(std::string fname, trie\_single \*const ctrie\_single, int limit\_count);

int getFileNum()

{

return vfname.size();

}

unsigned int get\_words\_num()

{

return words\_num;

}

};

Read texts and insert the words into trie. Then sort the words occurrence times and write the information into a file. And get the common words from the file and insert these common words into trie.

**trie\_sentence**

class trie\_sentence

{

private:

sentence\_trie\_node\* T\_root;

void delNode(sentence\_trie\_node\* node); //Delete trie node

sentence\_trie\_node\* createNode(std::string character); //Create a new node

int calIndex(char letter); //Calculate the index of the key

void output\_tr(sentence\_trie\_node\* node, max\_heap\* cmax\_heap, std::string output\_word); //Output trie into max-heap for sorting

void output\_d(sentence\_trie\_node\* node, std::string output\_word); //Display trie

public:

trie\_sentence();

~trie\_sentence();

public:

void insert(std::string key);

unsigned int search(std::string key); //Search a key and return the occurrence time

void traverse(max\_heap\* cmax\_heap); ////Output trie into max-heap for sorting

void display(); //Display trie

};

Trie class for storing word patterns.

**sentence\_freq**

class sentence\_freq

{

private:

std::vector<std::string> vfname; //Texts names that will be read

unsigned int sentences\_num; //The total number of sentences has read

bool isEndOfSen(char\* con); //Judge if the character is the end of the sentence

bool isVaild(char\* con); //Judge if the character is a vaild English letter

int wordsInPattern(std::string pattern); //Calculate the number of words in the pattern

public:

sentence\_freq();

~sentence\_freq();

//The function that should be called for getting the word patterns information. out\_fname: the name of the result file

void cal\_sentences\_freq( trie\_single \*const ctrie\_single, std::string out\_fname = "sentence\_freq.txt");

//ctrie\_sentence: the trie where pattern stored; ctrie\_single: the trie where common words stored

void insert\_sentences(std::string fname, trie\_sentence \*const ctrie\_sentence, trie\_single \*const ctrie\_single);

void write\_to\_file(std::string out\_fname, max\_heap \*const cmax\_heap); //Write the word patterns information into the file

int getFileNum()

{

return vfname.size();

}

unsigned int get\_sentences\_num()

{

return sentences\_num;

}

};

Read texts and insert the word patterns into trie. Then sort the word patterns occurrence times and write the information into a file.

# **Chinese**

**Reference**

Goodrich, Michael T., & Tamassia, Roberto, Algorithm Design and Applications, John Wiley Publishing 2015