作业 PA7 实验报告

姓名：何正潇 学号：1950095 日期：2022 年 1 月 7 日

1. 涉及数据结构和相关背景

涉及的数据结构主要是关于树，题目的背景在于根据前缀匹配进行字典树的建立与搜索。

2. 实验内容

2.1 统计文章中单词出现次数

2.1.1 问题描述

给一篇超过100000字的英文文章，你能统计出里面每一个单词出现的次数吗？

相信聪明的你，一定不屑于使用std::map这种走捷径的手段，并想到使用字典树可以很好地解决这个问题

2.1.2 基本要求

要求：

输入：

1. 一个纯ASCII字符组成的文本文件，统计里面每个单词出现的次数
2. 单词仅由大、小写英文字母组成，处理时将所有大写字母转换为小写字母
3. 当出现连接符'-'连接两个单词时，如果'-'后跟的是换行符，可以视为'-'前后是一个因换行而被拆开的单词（即删除'-'并将'-'前后连接起来成为一个单词），否则视为两个单词。
4. 使用字典树作为基本数据结构

输出：

1. 输出为若干形式为<key,value>的键值对，其中键为文本中出现的单词，值为这个单词出现的次数
2. 每一行输出一个键值对，输出按键的字典序升序排序

2.1.3 数据结构设计

```
using namespace std;
struct TrieNode
{
    TrieNode* arrNext[26] = { NULL };
    bool bTerminate = false;
    int count = 0;
};
class Dictionary tree
```

就是一个简单的多叉树结构

2.1.4 功能说明（函数、类）

```
class Dictionary_tree
{
public:
    Dictionary_tree(const char* p); //字典树建成
    Dictionary_tree();
    Dictionary_tree& add(const char* p);//字典树添加元素
    void trie_travel(TrieNode* cur);//字典树遍历
    TrieNode* root = NULL;
};
Dictionary_tree::Dictionary_tree(const char* p)
{
    const char* temp = p;
    root = new TrieNode;
    TrieNode* temp1 = root;
    root->arrNext[temp[0] - 'a'] = new TrieNode;
    temp1 = root->arrNext[temp[0] - 'a'];
    temp++;
    while (*temp != '\0')
    {
        if (temp1->arrNext[*temp - 'a'] == NULL)
        {
            temp1->arrNext[*temp - 'a'] = new TrieNode;
            temp1 = temp1->arrNext[*temp - 'a'];
        }
        else
        {
            temp1 = temp1->arrNext[*temp - 'a'];
        }
        temp++;
    }
    if (temp1->bTerminate == false)
    {
        temp1->bTerminate = true;
        temp1->count++;
    }
    else
        temp1->count++;

}
Dictionary_tree::Dictionary_tree()
{
    root = NULL;
```

```
}
Dictionary_tree& Dictionary_tree::add(const char* p)
{
    const char* temp = p;
    TrieNode* temp1 = root;
    while (*temp != '\0')
    {
        if (temp1->arrNext[*temp - 'a'] == NULL)
        {
            temp1->arrNext[*temp - 'a'] = new TrieNode;
            temp1 = temp1->arrNext[*temp - 'a'];
        }
        else
        {
            temp1 = temp1->arrNext[*temp - 'a'];
        }
        temp++;
    }
    if (temp1->bTerminate == false)
    {
        temp1->bTerminate = true;
        temp1->count++;
    }
    else
        temp1->count++;
    return *this;
}
void Dictionary_tree::trie_travel(TrieNode* cur)
{
    static string temp;
    static int pos = 0;
    int i;
    if (cur->count != 0 || cur->bTerminate == true)
    {
        cout << "<";
        cout << temp << "," << cur->count << ">" << endl;
    }
    for (i = 0; i < 26; i++)
    {
        if (cur->arrNext[i] != NULL)
        {
            temp += i + 'a';
            trie_travel(cur->arrNext[i]);
            if (strlen(temp.c_str()) >= 1)
```

```
                temp.erase(strlen(temp.c_str()) - 1, 1);
            else
                ;
        }
    }
}
```

2.1.5 调试分析（遇到的问题和解决方法）

利用彭博社的自我介绍进行简单测试。

We power global finance. Bloomberg is a forward looking company focused on building products and solutions that are needed for centurys. As a global information and technology company, we connect decision makers to a dynamic network of data, people and ideas. accurately delivering business and financial information, news and insights to customers around the world.

结果输出

result.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

<a,3>
<accurately,1>
<and,5>
<are,1>
<around,1>
<as,1>
<bloomberg,1>
<building,1>
<business,1>
<centurys,1>
<company,2>
<connect,1>
<customers,1>
<data,1>
<decision,1>
<delivering,1>
<dynamic,1>
<finance,1>
<financial,1>
<focused,1>
<for,1>
<forward,1>
<global,2>
<ideas,1>
<information,2>
<insights,1>
<is,1>
<looking,1>
<makers,1>
<needed,1>
<network,1>
<news,1>
<of,1>
<on,1>
<people,1>
<power,1>
<products,1>
<solutions,1>
<technology,1>
<that,1>
<the,1>
<to,2>
<we,2>
<world,1>

2.1.6 总结和体会

我感觉这类题目最难的在于文件处理，除了这个以外，别的建立字典树以及字典树遍历倒是难度尚可。这题深化了我对于搜索概念的理解，同时我对于查找的实际应用也有了一定的了解和体会，对于接下来的进一步学习帮助很大。而且我在github 上也发现很多关于字典树的趣味应用，包括敏感字段屏蔽等应用，可以说是很有趣而且实用的。