实验六report

本实验中，在读入文档的时候，将除了“ ”之外的符号全部按照空格读入，保证文章的处理不被其他多余符号干扰。

对于哈希表结构的设计：

采用链地址的方式处理哈希冲突，链表节点含有两个成员变量，一个记录单词的word，另一个记录其邻接表的下一个哈希结点。

在哈希表中，每次插入均根据哈希值找到对应索引位置，如果当前索引位置的内容不是null，即当前位置已有元素，这时候查找当前位置上的邻接表，若邻接表中的元素的key值都与当前待插入的单词相同，则说明当前单词已经被录入，直接跳过，否则发生冲突，令记录冲突次数的变量加一，并把当前单词插入哈希表该位置的邻接表的表头。

对于两种哈希函数的设计：

对于多项式函数，将读入的所有英文字母映射为其对应的ascii码值，遇到数字时，直接将其映射为对应的数值。随后，按照单词中字母出现顺序，单词abcd的哈希值即为：
a”g^3+“b”g^2+“c”g+“d”g0，其中，g被赋值为131.

对于音节编码，处理每个单词时，如果遇到元音字母aeiou，则将其忽略不计，若遇到辅音字母，则将其按照多项式取值的方法求得哈希值。这里的g值取31

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <cctype>
#include <chrono>

using namespace std;

// 常量定义
const int DEFAULT_TABLE_SIZE = 10007; // 哈希表大小，选择一个质数
const long long MOD = 1000000007;     // 模数，用于防止哈希值溢出
const int G1 = 131;                    // 哈希方法一的基数
const int G2 = 31;                     // 哈希方法二的基数

// 链表节点结构
struct Node {
    string key;
```

```
    Node* next;
    Node(const string& s) : key(s), next(nullptr) {}
};

// 哈希表类
class HashTable {
private:
    Node** table;           // 哈希表数组，每个位置指向链表的头节点
    int size;               // 哈希表大小
    int collision_count;    // 冲突计数

public:
    // 构造函数
    HashTable(int sz = DEFAULT_TABLE_SIZE) : size(sz), collision_count(0) {
        table = new Node*[size];
        for(int i = 0; i < size; ++i) {
            table[i] = nullptr;
        }
    }

    // 析构函数
    ~HashTable() {
        for(int i = 0; i < size; ++i) {
            Node* current = table[i];
            while(current) {
                Node* temp = current;
                current = current->next;
                delete temp;
            }
        }
        delete[] table;
    }

    // 哈希函数一：多项式滚动哈希（编码字母和数字，字母使用ASCII码）
    long long hash1(const string& s) const {
        long long hash = 0;
        for(char c : s) {
            if(isalpha(c)) {
                hash = (hash * G1 + static_cast<int>(tolower(c))) % MOD;
            }
            else if(isdigit(c)) {
                hash = (hash * G1 + (c - '0')) % MOD;
            }
            // 非字母非数字字符已在 cleanWord 中移除，不需处理
        }
        return hash;
    }
```

```
66
67      // 哈希函数二：基于辅音字母和数字编码的哈希（忽略元音字母，字母使用ASCII码）
68      long long hash2(const string& s) const {
69          long long hash = 0;
70          for(char c : s) {
71              char lower = tolower(c);
72              // 检查是否为辅音字母或数字
73              if( (isalpha(lower) && !strchr("aeiou", lower)) || isdigit(c) ) {
74                  if(isalpha(lower)){
75                      hash = (hash * G2 + static_cast<int>(lower)) % MOD;
76                  }
77                  else { // 数字
78                      hash = (hash * G2 + (c - '0')) % MOD;
79                  }
80              }
81              // 元音字母和非字母非数字字符忽略
82          }
83          return hash;
84      }
85
86      // 插入单词，选择哈希方法
87      void insert(const string& key, int method) {
88          long long hash_val = (method == 1) ? hash1(key) : hash2(key);
89          int index = hash_val % size;
90
91          Node* current = table[index];
92          while(current) {
93              if(current->key == key) {
94                  // 单词已存在，跳过插入
95                  return;
96              }
97              current = current->next;
98          }
99
100         // 单词不存在，检查是否需要增加冲突计数
101         if(table[index] != nullptr) {
102             collision_count++; // 该位置已有其他单词，发生冲突
103         }
104
105         // 插入到链表头
106         Node* new_node = new Node(key);
107         new_node->next = table[index];
108         table[index] = new_node;
109     }
110
111     // 查找单词，返回查找步数（从1开始），如果未找到则返回-1
112     int find(const string& key, int method) const {
```

```
        long long hash_val = (method == 1) ? hash1(key) : hash2(key);
        int index = hash_val % size;
        int steps = 1;
        Node* current = table[index];
        while(current) {
            if(current->key == key) {
                return steps;
            }
            current = current->next;
            steps++;
        }
        return -1; // 未找到
    }

    // 获取总冲突次数
    int getCollisionCount() const {
        return collision_count;
    }
};

// 工具函数：将字符串转换为小写并去除非字母非数字字符
string cleanWord(const string& word) {
    string clean;
    for(char c : word) {
        if(isalpha(c) || isdigit(c)) {
            clean += tolower(c);
        }
    }
    return clean;
}

// 工具函数：从文件中读取单词
int readWords(const string& filename, string* words, int max_words) {
    ifstream infile(filename);
    if(!infile.is_open()) {
        cout << "无法打开文件: " << filename << endl;
        return 0;
    }
    int count = 0;
    string word;
    while(infile >> word && count < max_words) {
        string clean = cleanWord(word);
        if(!clean.empty()) {
            words[count++] = clean;
        }
    }
    infile.close();
```

```
    return count;
}

int main() {
    // 文件路径
    const string file1 = "article1.txt"; // 用于构建哈希表
    const string file2 = "article2.txt"; // 用于查找

    // 读取第一篇文章的单词
    const int MAX_WORDS = 100000;
    string* words1 = new string[MAX_WORDS];
    int count1 = readWords(file1, words1, MAX_WORDS);
    cout << "第一篇文章读取了 " << count1 << " 个单词。" << endl;

    // 创建两个哈希表，分别使用两种哈希方法
    HashTable ht1; // 方法一：多项式滚动哈希
    HashTable ht2; // 方法二：基于辅音字母和数字编码的哈希

    // 插入单词到哈希表，并记录时间
    auto start_insert1 = chrono::high_resolution_clock::now();
    for(int i = 0; i < count1; ++i) {
        ht1.insert(words1[i], 1);
    }
    auto end_insert1 = chrono::high_resolution_clock::now();
    chrono::duration<double> duration_insert1 = end_insert1 - start_insert1;
    cout << "哈希方法一插入完成，耗时: " << duration_insert1.count() << " 秒。" <<
endl;

    auto start_insert2 = chrono::high_resolution_clock::now();
    for(int i = 0; i < count1; ++i) {
        ht2.insert(words1[i], 2);
    }
    auto end_insert2 = chrono::high_resolution_clock::now();
    chrono::duration<double> duration_insert2 = end_insert2 - start_insert2;
    cout << "哈希方法二插入完成，耗时: " << duration_insert2.count() << " 秒。" <<
endl;

    // 读取第二篇文章的单词
    string* words2 = new string[MAX_WORDS];
    int count2 = readWords(file2, words2, MAX_WORDS);
    cout << "第二篇文章读取了 " << count2 << " 个单词。" << endl;

    // 查找单词并统计平均查找长度
    long long total_steps1 = 0;
    long long found1 = 0;
    auto start_find1 = chrono::high_resolution_clock::now();
    for(int i = 0; i < count2; ++i) {
```

```
205         int steps = ht1.find(words2[i], 1);
206         if(steps != -1) {
207             total_steps1 += steps;
208             found1 += 1;
209         }
210     }
211     auto end_find1 = chrono::high_resolution_clock::now();
212     chrono::duration<double> duration_find1 = end_find1 - start_find1;
213     double avg_steps1 = (found1 == 0) ? 0.0 : ((double)total_steps1 / found1);
214     cout << "哈希方法一查找完成，耗时: " << duration_find1.count() << " 秒。" <<
    endl;
215     cout << "哈希方法一平均查找长度: " << avg_steps1 << endl;
216 
217     long long total_steps2 = 0;
218     long long found2 = 0;
219     auto start_find2 = chrono::high_resolution_clock::now();
220     for(int i = 0; i < count2; ++i) {
221         int steps = ht2.find(words2[i], 2);
222         if(steps != -1) {
223             total_steps2 += steps;
224             found2 += 1;
225         }
226     }
227     auto end_find2 = chrono::high_resolution_clock::now();
228     chrono::duration<double> duration_find2 = end_find2 - start_find2;
229     double avg_steps2 = (found2 == 0) ? 0.0 : ((double)total_steps2 / found2);
230     cout << "哈希方法二查找完成，耗时: " << duration_find2.count() << " 秒。" <<
    endl;
231     cout << "哈希方法二平均查找长度: " << avg_steps2 << endl;
232 
233     // 输出冲突率
234     cout << "哈希方法一总冲突数: " << ht1.getCollisionCount() << endl;
235     cout << "哈希方法二总冲突数: " << ht2.getCollisionCount() << endl;
236 
237     // 计算并输出冲突率
238     double collision_rate1 = (count1 > 0) ? ((double)ht1.getCollisionCount() /
    count1) : 0.0;
239     double collision_rate2 = (count1 > 0) ? ((double)ht2.getCollisionCount() /
    count1) : 0.0;
240     cout << "哈希方法一冲突率: " << collision_rate1 * 100 << " %" << endl;
241     cout << "哈希方法二冲突率: " << collision_rate2 * 100 << " %" << endl;
242 
243     // 清理内存
244     delete[] words1;
245     delete[] words2;
246 
247     return 0;
```

```
248 }
249
```

在处理以下两篇文章时，该代码运行效果如下

⚽ article1.txt

The exploration of space is one of humanity’ s most ambitious endeavors. It has been driven by our curiosity to understand the universe and our desire to expand our presence beyond Earth. The history of space exploration began with the launch of the first artificial satellite, Sputnik 1, by the Soviet Union in 1957. This event marked the beginning of the Space Race between the Soviet Union and the United States, a period of intense competition in space exploration.

In 1961, Soviet cosmonaut Yuri Gagarin became the first human to journey into space. His flight aboard Vostok 1 was a significant milestone in the history of space exploration. The United States quickly followed suit with the Mercury and Gemini space programs, which laid the groundwork for the Apollo missions. In 1969, Apollo 11 successfully landed astronauts Neil Armstrong and Buzz Aldrin on the Moon, marking one of the most significant achievements in human history.

Since then, space exploration has continued to advance. The launch of the Hubble Space Telescope in 1990 provided us with breathtaking images of distant galaxies, while the Mars rovers have sent back invaluable data about the Red Planet. In recent years, private companies like SpaceX and Blue Origin have entered the space race, promising to make space travel more affordable and accessible.

❤️ article2.txt

Climate change refers to long-term changes in temperature, precipitation, and other atmospheric conditions on Earth. It is largely driven by human activities, particularly the burning of fossil fuels, which release greenhouse gases into the atmosphere. These gases trap heat, causing the planet to warm and leading to a phenomenon known as global warming.

The impacts of climate change are widespread and severe. Rising global temperatures are causing polar ice caps to melt, leading to rising sea levels that threaten coastal communities. Extreme weather events, such as hurricanes, droughts, and floods, are

becoming more frequent and intense. These changes are not only damaging ecosystems but are also having profound effects on agriculture, water supply, and human health.

One of the most concerning aspects of climate change is its effect on biodiversity. Many species are unable to adapt to the changing conditions and are facing extinction. The loss of biodiversity can have cascading effects on ecosystems, affecting everything from food chains to pollination.

To address climate change, world leaders have called for global cooperation to reduce carbon emissions and invest in renewable energy sources. The Paris Agreement, signed in 2015, aims to limit global temperature rise to below 2°C above pre-industrial levels. However, much work remains to be done to mitigate the effects of climate change and protect the planet for future generations.

🎨 输出结果

第一篇文章读取了 215 个单词。

哈希方法一插入完成，耗时: 0 秒。

哈希方法二插入完成，耗时: 0 秒。

第二篇文章读取了 221 个单词。

哈希方法一查找完成，耗时: 0 秒。

哈希方法一平均查找长度: 1

哈希方法二查找完成，耗时: 0 秒。

哈希方法二平均查找长度: 1.1194

哈希方法一总冲突数: 1

哈希方法二总冲突数: 6

哈希方法一冲突率: 0.465116 %

哈希方法二冲突率: 2.7907 %

其中，方法二的平均查找长度和冲突次数均大于方法一，由于方法二创建哈希函数的时候其实是将方法一的内容去除了其中的元音音节来进行计算的，所以该结果符合预期。