Home work #5 B10815057 廖聖郝

1. Word Checker

   Hash function，參考自:

   https://stackoverflow.com/questions/7666509/hash-function-for-string

```
#define MAX_ARRAY_SIZE 299903//list 陣列大小，用質數可以減少碰撞

//參考自:https://stackoverflow.com/questions/7666509/hash-function-for-string
unsigned int djb2(string& word) {//hash function
    unsigned hash = 5381;
    for (int i = 0; i < word.size(); i++) {
        hash = ((hash << 5) + hash) + word[i];
    }
    return (hash % MAX_ARRAY_SIZE);
}
```

   用 linked list 儲存發生碰撞的資料

```
template<typename T>
struct Node {
    Node(T& s) :data(s) {}
    T data;//節點內的資料
    Node* next = nullptr;//指到下一個節點
};

template<typename T>
struct List {
    Node<T>* head = nullptr;//linked list 起始點
    void add(T n) {//加入新 node，直接放在開頭的地方
        Node<T>* new_node = new Node<T>(n);//配置記憶體尺產生新節點
        new_node->next = head;//新節點的下一個節點改為現在的起始點
        head = new_node;//起始點改為新節點
    }
    bool find(T target) {//在 List 裡尋找該資料，找到回傳 true
        Node<T>* now = head;//先設當前節點為起始點
        while (now) {//跑過每一個節點
            if (now->data == target) {//找到目標資料
                return true;//回傳 true
            }
            now = now->next;//往下一節點走
        }
        return false;//找不到回傳 false
    }
};
```

字典結構，傳入 hash function 的函式指標，新增或查找資料都用 hash 找到 list，然後再呼叫 list 的新增或查找函式

```
template<typename T>
struct Dictionary {//字典
    Dictionary(unsigned int (*f)(T&) ):hash_funciton(f){
    //初始化 hash function，用函式指標傳入
        all_list = new List<T>[MAX_ARRAY_SIZE];//配置 list 陣列
    }
    void add(T new_data) {
        all_list[hash_funciton(new_data)].add(new_data);
        //用 hash 找到 list，然後對該 list 新增節點
    }
    bool find(T target) {
        return all_list[hash_funciton(target)].find(target);
        //用 hash 找到 list，然後對該 list 做搜尋
    }
    ~Dictionary() {
        delete[] all_list;//釋放記憶體
    }
    List<T>* all_list;//所有 list
    unsigned int (*hash_funciton)(T&);//hash function pointer，函式指標
};
```

創建字典、開啟檔案、儲存時間，並把字典檔的所有單字加到字典裡

```
    auto start = std::chrono::steady_clock::now();//儲存開始時間
    Dictionary<string>  dictionary(djb2);
    //創建字典，使用 djb2 hash function
    ifstream dictionary_file;//字典檔案
    string word;//暫存單字
    dictionary_file.open("dictionary.txt");//開啟字典檔案
    while (getline(dictionary_file, word))
    //讀入字典檔裡的每個單字然後存到字典裡
    {
        dictionary.add(word);//存到字典裡
    }
    dictionary_file.close();//關閉字典檔

    ifstream input_file;//輸入檔案
    input_file.open("input_500.txt");//開啟輸入檔案

    ofstream output_csv;//輸出檔案
    output_csv.open("answer.csv");//開啟輸出檔案
    output_csv << "word,answer\n";//輸出第一行的固定格式
```

處理所有的輸入，首先確認輸入的單字是否存在字典中，如果有就輸出OK，如果沒有，就遞迴尋找子字串然後輸出，如果沒有子字串，就輸出NONE

```
    while (getline(input_file, word))//讀入輸入檔裡的每個單字然後處理
    {
        output_csv << word << ',';//輸出要處理的單字，固定格式
        if (dictionary.find(word)) {//如果該單字存在於字典裡
            output_csv << "OK\n";//輸出 OK
            continue;//不處理子字串，直接處理下個單字
        }

        set<string> result;//存放所有存在於字典裡的子字串
        generate_string_subset(result, word, dictionary, true);
        //呼叫遞迴函式，產生子字串並與字典比對，產生最終結果
        if (result.size()) {//如果有一個以上的結果，全部輸出
            output_csv << *result.begin();
            //第一個輸出的前面沒有空白，特別處理它
            for (auto i = next(result.begin());i != result.end();i++) {
             //從第二個跑到最後一個
                output_csv << ' ' << *i;//前面要加空白，以區隔單字
            }
        }
        else {//沒有任何結果
            output_csv << "NONE";//輸出 NONE
        }
        output_csv << '\n';//換行
    }
```

計算花費時間並輸出，然後關閉檔案

```
    auto end = std::chrono::steady_clock::now();//儲存結束時間
    auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds
>(end - start);//總花費時間(毫秒)
    std::cout << "time: " << elapsed.count() / 1000 << " seconds " << e
lapsed.count() % 1000 << " milliseconds";//輸出幾秒幾毫秒
    input_file.close();//關閉輸入檔
    output_csv.close();//關閉輸出檔
    return 0;
```

產生子字串的遞迴函式，result 儲存結果，使用 set，因此附帶排序好的資料，且不用處理重複新增的問題，target 是要產生子字串的母字串，dictionary 是字典，first 是紀錄第幾次遞迴，如果 first 為 false 就不再遞迴下去

```
void generate_string_subset(set<string>& result,const string& target, Dicti
onary<string>& dictionary, bool first) {//產生子字串並與字典比對，產生最終結果
```

該函式共分為 4 個部分:insert、delete、substitute、transpose

Insert:每個可插入位置放入 a~z，可插入位置產生 26 個子字串

```
    string insert(target.length() + 1, 0);
    //insert 的大小會是母字串大小+1
    for (int i = 0;i <= target.length();i++) {
    //跑過每個可插入的位置
        for (int j = 0, index = 0;j < target.length() + 1; j++) {
         //跑過新字串的每個位置，index 儲存原字串的位置
            if (j != i) {//如果跑到要插入的位置，就跳過
                insert[j] = target[index++];
                 //放入原字串的字，原字串 index + 1
            }
        }
        for (char c = 'a';c <= 'z';c++) {
         //把 a~z 放到要插入的位置，產生 26 個子字串
            insert[i] = c;
            if (dictionary.find(insert)) {//該子字串是否存在於字典裡
                result.insert(insert);//如果存在就放到結果集合裡

            }
            if (first) generate_string_subset(result, insert, dictionar
y,false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
    }
```

Delete:每個位置刪除，產生母字串長度個子字串

```
    string delete_s(target.length() - 1, 0);
    //delete 的大小會是母字串大小-1
    for (int i = 0;i < target.length();i++) {
        for (int j = 0,index = 0;j < target.length(); j++) {
         //跑過原字串的每個位置，index 儲存新字串的位置
            if (j != i) {//如果跑到要刪除的位置，就跳過
                delete_s[index++] = target[j];
                 //放入原字串的字，新字串 index + 1
            }
        }

        if (dictionary.find(delete_s)) {//該子字串是否存在於字典裡
            result.insert(delete_s);//如果存在就放到結果集合裡
        }

        if(first) generate_string_subset(result, delete_s, dictionary,f
alse);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
    }
```

Substitute:每個位置替代 a~z，每個位置產生 26 個子字串

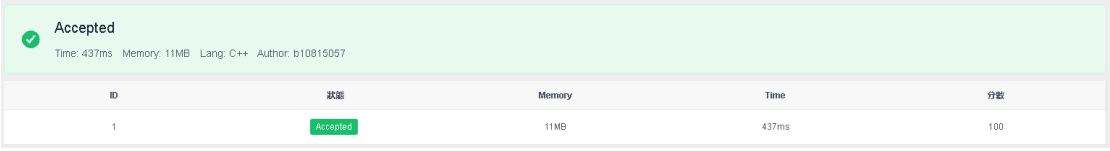
```
    string substitute = target;//substitute 的大小與母字串大小相同
    for (int i = 0;i < target.length();i++) {//跑過字串的每個位置
        for (char c = 'a';c <= 'z';c++) {
        //每個位置都替代 a~z，產生 26 個子字串
            substitute[i] = c;

            if (dictionary.find(substitute)) {//該子字串是否存在於字典裡
                result.insert(substitute);//如果存在就放到結果集合裡
            }

            if (first) generate_string_subset(result, substitute, dicti
onary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
        substitute[i] = target[i];//恢復成原本的字，然後處理下個位置
    }
```

Transpose:除了最後一個位置，每個位置與下個位置交換，產生母字串長度-1 個子字串

```
    string transpose = target;//transpose 的大小與母字串大小相同
    for (int i = 0;i < target.length() - 1;i++) {
    //鄰居交換，所以只有母字串大小-1個子字串

        swap(transpose[i], transpose[i + 1]);//現在位置與下個位置交換

        if (dictionary.find(transpose)) {//該子字串是否存在於字典裡
            result.insert(transpose);//如果存在就放到結果集合裡
        }

        if (first) generate_string_subset(result, transpose, dictionary
,false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串

        swap(transpose[i], transpose[i + 1]);//換回來，然後處理下個位置
    }
```

如果子字串存在於字典中，就放入 result，如果是第一次遞迴，每個子字串會再遞迴一次。

2. Source code
   GIST　　IMGUR 圖片
   OJ version:
   GIST　　IMGUR 圖片
3. OJ Submission



```cpp
#include <fstream>
#include <string>
#include <set>
#include <iostream>
#include <numeric>
#include <unordered_map>
#include <iterator>
#include <chrono>
#include <sstream>
using namespace std;

#define MAX_ARRAY_SIZE 299903//list陣列大小，用質數可以減少碰撞

//參考自:https://stackoverflow.com/questions/7666509/hash-function-for-string
unsigned int djb2(string& word) {//hash function
    unsigned hash = 5381;
    for (int i = 0; i < word.size(); i++) {
        hash = ((hash << 5) + hash) + word[i];
    }
    return (hash % MAX_ARRAY_SIZE);
}

template<typename T>
struct Node {
    Node(T& s) :data(s) {}
    T data;//節點內的資料
    Node* next = nullptr;//指到下一個節點
};
```

```cpp
template<typename T>  <T> 提供適用於 IntelliSense 的樣本範本引數
struct List {
    Node<T>* head = nullptr;//linked list起始點
    void add(T n) {//加入新node，直接放在開頭的地方
        Node<T>* new_node = new Node<T>(n);//配置記憶體尺產生新節點
        new_node->next = head;//新節點的下一個節點改為現在的起始點
        head = new_node;//起始點改為新節點
    }
    bool find(T target) {//在List裡尋找該資料，找到回傳true
        Node<T>* now = head;//先設當前節點為起始點
        while (now) {//跑過每一個節點
            if (now->data == target) {//找到目標資料
                return true;//回傳true
            }
            now = now->next;//往下一節點走
        }
        return false;//找不到回傳false
    }
};

template<typename T>
struct Dictionary {//字典
    Dictionary(unsigned int (*f)(T&) ):hash_funciton(f){//初始化hash function，用函式指標傳入
        all_list = new List<T>[MAX_ARRAY_SIZE];//配置list陣列
    }
    void add(T new_data) {
        all_list[hash_funciton(new_data)].add(new_data);//用hash找到list，然後對該list新增節點
    }
    bool find(T target) {
        return all_list[hash_funciton(target)].find(target);//用hash找到list，然後對該list做搜尋
    }
    ~Dictionary() {
        delete[] all_list;//釋放記憶體
    }
    List<T>* all_list;//所有list
    unsigned int (*hash_funciton)(T&);//hash function pointer，函式指標
};

void generate_string_subset(set<string>& result,const string& target, Dictionary<string>& dictionary,bool);//產生子字串並與字典比對，產生最終結果
```

```cpp
auto start = std::chrono::steady_clock::now();//儲存開始時間
Dictionary<string>  dictionary(djb2);//創建字典，使用djb2 hash function
ifstream dictionary_file;//字典檔案
string word;//暫存單字
dictionary_file.open("dictionary.txt");//開啟字典檔案
while (getline(dictionary_file, word))//讀入字典檔裡的每個單字然後存到字典裡
{
    dictionary.add(word);//存到字典裡
}
dictionary_file.close();//關閉字典檔

ifstream input_file;//輸入檔案
input_file.open("input_500.txt");//開啟輸入檔案

ofstream output_csv;//輸出檔案
output_csv.open("answer.csv");//開啟輸出檔案
output_csv << "word,answer\n";//輸出第一行的固定格式
while (getline(input_file, word))//讀入輸入檔裡的每個單字然後處理
{
    output_csv << word << ',';//輸出要處理的單字，固定格式
    if (dictionary.find(word)) {//如果該單字存在於字典裡
        output_csv << "OK\n";//輸出OK
        continue;//不處理子字串，直接處理下個單字
    }

    set<string> result;//存放所有存在於字典裡的子字串
    generate_string_subset(result, word, dictionary, true);//呼叫遞迴函式，產生子字串並與字典比對，產生最終結果
    if (result.size()) {//如果有一個以上的結果，全部輸出
        output_csv << *result.begin();//第一個輸出的前面沒有空白，特別處理它
        for (auto i = next(result.begin());i != result.end();i++) {//從第二個跑到最後一個
            output_csv << ' ' << *i;//前面要加空白，以區隔單字
        }
    }
    else {//沒有任何結果
        output_csv << "NONE";//輸出NONE
    }
    output_csv << '\n';//換行
}

auto end = std::chrono::steady_clock::now();//儲存結束時間
auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);//總花費時間(毫秒)
std::cout << "time: " << elapsed.count() / 1000 << " seconds " << elapsed.count() % 1000 << " milliseconds";//輸出幾秒幾毫秒
input_file.close();//關閉輸入檔
output_csv.close();//關閉輸出檔
return 0;
void generate_string_subset(set<string>& result,const string& target, Dictionary<string>& dictionary, bool first) {//產生子字串並與字典比對，產生最終結果
    //result儲存結果，使用set，因此附帶排序好的資料，且不用處理重複新增的問題，target是要產生子字串的母字串，dictionary是字典，first是紀錄第幾次遞迴，如果first為false
    string insert(target.length() + 1, 0);//insert的大小會是母字串大小+1
    for (int i = 0;i <= target.length();i++) {//跑過每個可插入的位置
        for (int j = 0, index = 0;j < target.length() + 1; j++) {//跑過新字串的每個位置，index儲存原字串的位置
            if (j != i) {//如果跑到要插入的位置，就跳過
                insert[j] = target[index++];//放入原字串的字，原字串index + 1
            }
        }
        for (char c = 'a';c <= 'z';c++) {//把a~z放到要插入的位置，產生26個子字串
            insert[i] = c;
            if (dictionary.find(insert)) {//該子字串是否存在於字典裡
                result.insert(insert);//如果存在就放到結果集合裡

            }
            if (first) generate_string_subset(result, insert, dictionary,false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
    }
    string delete_s(target.length() - 1, 0);//delete的大小會是母字串大小-1
    for (int i = 0;i < target.length();i++) {
        for (int j = 0,index = 0;j < target.length(); j++) {//跑過原字串的每個位置，index儲存新字串的位置
            if (j != i) {//如果跑到要刪除的位置，就跳過
                delete_s[index++] = target[j];//放入原字串的字，新字串index + 1
            }
        }

        if (dictionary.find(delete_s)) {//該子字串是否存在於字典裡
            result.insert(delete_s);//如果存在就放到結果集合裡
        }

        if(first) generate_string_subset(result, delete_s, dictionary,false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
    }
```

```
    string substitute = target;//substitute的大小與母字串大小相同
    for (int i = 0;i < target.length();i++) {//跑過字串的每個位置
        for (char c = 'a';c <= 'z';c++) {//每個位置都替代a~z，產生26個子字串
            substitute[i] = c;

            if (dictionary.find(substitute)) {//該子字串是否存在於字典裡
                result.insert(substitute);//如果存在就放到結果集合裡
            }

            if (first) generate_string_subset(result, substitute, dictionary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
        substitute[i] = target[i];//恢復成原本的字，然後處理下個位置
    }

    string transpose = target;//transpose的大小與母字串大小相同
    for (int i = 0;i < target.length() - 1;i++) {//鄰居交換，所以只有母字串大小-1個子字串

        swap(transpose[i], transpose[i + 1]);//現在位置與下個位置交換

        if (dictionary.find(transpose)) {//該子字串是否存在於字典裡
            result.insert(transpose);//如果存在就放到結果集合裡
        }

        if (first) generate_string_subset(result, transpose, dictionary,false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串

        swap(transpose[i], transpose[i + 1]);//換回來，然後處理下個位置
    }
}
```

OJ version:

```
#include <string>
#include <set>
#include <iostream>
#include <iterator>
using namespace std;

#define MAX_ARRAY_SIZE 299983//list陣列大小，用質數可以減少碰撞

//參考自:https://stackoverflow.com/questions/7666509/hash-function-for-string
unsigned int djb2(string& word) {//hash function
    unsigned hash = 5381;
    for (int i = 0; i < word.size(); i++) {
        hash = ((hash << 5) + hash) + word[i];
    }
    return (hash % MAX_ARRAY_SIZE);
}

template<typename T>
struct Node {
    Node(T& s) :data(s) {}
    T data;//節點內的資料
    Node* next = nullptr;//指到下一個節點
};

template<typename T>
struct List {
    Node<T>* head = nullptr;//linked list起始點
    void add(T& n) {//加入新node，直接放在開頭的地方
        Node<T>* new_node = new Node<T>(n);//配置記憶體尺產生新節點
        new_node->next = head;//新節點的下一個節點改為現在的起始點
        head = new_node;//起始點改為新節點
    }
    bool find(T& target) {//在List裡尋找該資料，找到回傳true
        Node<T>* now = head;//先設當前節點為起始點
        while (now) {//跑過每一個節點
            if (now->data == target) {//找到目標資料
                return true;//回傳true
            }
            now = now->next;//往下一節點走
        }
        return false;//找不到回傳false
    }
};
```

```cpp
template<typename T>
struct Dictionary {//字典
    Dictionary(unsigned int (*f)(T&)) :hash_funciton(f) {//初始化hash function，用函式指標傳入
        all_list = new List<T>[MAX_ARRAY_SIZE];//配置list陣列
    }
    void add(T& new_data) {
        all_list[hash_funciton(new_data)].add(new_data);//用hash找到list，然後對該list新增節點
    }
    bool find(T& target) {
        return all_list[hash_funciton(target)].find(target);//用hash找到list，然後對該list做搜尋
    }
    ~Dictionary() {
        delete[] all_list;//釋放記憶體
    }
    List<T>* all_list;//所有list
    unsigned int (*hash_funciton)(T&);//hash function pointer，函式指標
};

void generate_string_subset(set<string>& result, const string& target, Dictionary<string>& dictionary, bool);//產生子字串並與字典比對，產生最終結果

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    Dictionary<string>  dictionary(djb2);//創建字典，使用djb2 hash function
    string word;//暫存單字
    while (getline(cin, word))//讀入字典檔裡的每個單字然後存到字典裡
    {
        if (word[0] == ';') continue;
        else if (word[0] == '-')break;
        dictionary.add(word);//存到字典裡
    }
    cout << "word,answer\n";//輸出第一行的固定格式
    while (getline(cin, word))//讀入輸入檔裡的每個單字然後處理
    {
        if (word[0] == '-')break;
        cout << word << ',';//輸出要處理的單字，固定格式
        if (dictionary.find(word)) {//如果該單字存在於字典裡
            cout << "OK\n";//輸出OK
            continue;//不處理子字串，直接處理下個單字
        }

        set<string> result;//存放所有存在於字典裡的子字串
        generate_string_subset(result, word, dictionary, true);//呼叫遞迴函式，產生子字串並與字典比對，產生最終結果
        if (result.size()) {//如果有一個以上的結果，全部輸出
            cout << *result.begin();//第一個輸出的前面沒有空白，特別處理它
            for (auto i = next(result.begin());i != result.end();i++) {//從第二個跑到最後一個
                cout << ' ' << *i;//前面要加空白，以區隔單字
            }
        }
        else {//沒有任何結果
            cout << "NONE";//輸出NONE
        }
        cout << '\n';//換行
    }
    return 0;
}
void generate_string_subset(set<string>& result, const string& target, Dictionary<string>& dictionary, bool first) {//產生子字串並與字典比對，產生最終結果
    //result儲存結果，使用set，因此附帶排序好的資料，且不用處理重複新增的問題，target是要產生子字串的母字串，dictionary是字典，first是紀錄第幾次遞迴，如果first為fa
    string insert(target.length() + 1, 0);//insert的大小會是母字串大小+1
    for (int i = 0;i <= target.length();i++) {//跑過每個可插入的位置
        for (int j = 0, index = 0;j < target.length() + 1; j++) {//跑過新字串的每個位置，index儲存原字串的位置
            if (j != i) {//如果跑到要插入的位置，就跳過
                insert[j] = target[index++];//放入原字串的字，原字串index + 1
            }
        }
        for (char c = 'a';c <= 'z';c++) {//把a~z放到要插入的位置，產生26個子字串
            insert[i] = c;
            if (dictionary.find(insert)) {//該子字串是否存在於字典裡
                result.insert(insert);//如果存在就放到結果集合裡

            }
            if (first) generate_string_subset(result, insert, dictionary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
    }
    string delete_s(target.length() - 1, 0);//delete的大小會是母字串大小-1
    for (int i = 0;i < target.length();i++) {
        for (int j = 0, index = 0;j < target.length(); j++) {//跑過原字串的每個位置，index儲存新字串的位置
            if (j != i) {//如果跑到要刪除的位置，就跳過
                delete_s[index++] = target[j];//放入原字串的字，新字串index + 1
            }
        }

        if (dictionary.find(delete_s)) {//該子字串是否存在於字典裡
            result.insert(delete_s);//如果存在就放到結果集合裡
        }

        if (first) generate_string_subset(result, delete_s, dictionary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
    }
```

```
    string substitute = target;//substitute的大小與母字串大小相同
    for (int i = 0;i < target.length();i++) {//跑過字串的每個位置
        for (char c = 'a';c <= 'z';c++) {//每個位置都替代a~z，產生26個子字串
            substitute[i] = c;

            if (dictionary.find(substitute)) {//該子字串是否存在於字典裡
                result.insert(substitute);//如果存在就放到結果集合裡
            }

            if (first) generate_string_subset(result, substitute, dictionary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串
        }
        substitute[i] = target[i];//恢復成原本的字，然後處理下個位置
    }

    string transpose = target;//transpose的大小與母字串大小相同
    for (int i = 0;i < target.length() - 1;i++) {//鄰居交換，所以只有母字串大小-1個子字串

        swap(transpose[i], transpose[i + 1]);//現在位置與下個位置交換

        if (dictionary.find(transpose)) {//該子字串是否存在於字典裡
            result.insert(transpose);//如果存在就放到結果集合裡
        }

        if (first) generate_string_subset(result, transpose, dictionary, false);//如果是第一次遞迴就拿子字串遞迴第二次，產生子字串的子字串

        swap(transpose[i], transpose[i + 1]);//換回來，然後處理下個位置
    }
}
```