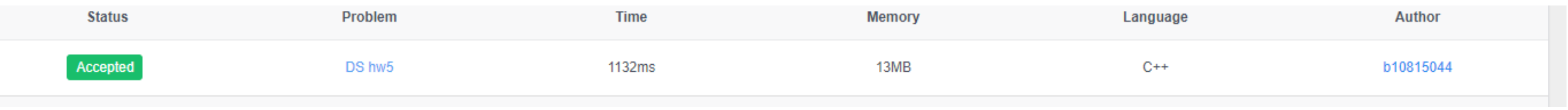
# B10815044 HomeWork5



| Status | Problem | Time | Memory | Language | Author |
|---|---|---|---|---|---|
| Accepted | DS hw5 | 1132ms | 13MB | C++ | b10815044 |

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <time.h>
#include<set>

using namespace std;
struct node
{
    string word;
    node* next_ptr;
    node(string a) {
        word = a;
        next_ptr = NULL;
    }
};

void Insert(string& word, std::set<string>&, bool);
void Delete(string& word, std::set<string>&, bool);
void Substitute(string& word, std::set<string>&, bool);
void Transpose(string& word, std::set<string>&, bool);
bool check_in_dictionary(string& word);

unsigned HashFunction(string word);
vector<node*>dictionary(500000); //Hash Map -> 存字典

int main()
{
    long double START= clock(), END; //計算時間用
    fstream dictionary_file,input_file,output_file;
    //讀檔案
    dictionary_file.open("dictionary.txt", ios::in);
    input_file.open("test.txt", ios::in);
    //寫檔案
    output_file.open("output.csv", ios::out);
    string word;
    vector<string>input; //存 input 的資料
    //建立 Hash Map 利用 djb2 與 chain 生成
    while (getline(dictionary_file, word)) {
        if (word[0] != ';') {
            int index=HashFunction(word);
            node* temp = new node(word);
            if(dictionary[index]!=NULL) {
                node* current = dictionary[index];
                while (current->next_ptr !=NULL) {
                    current = current->next_ptr;
                }
                current->next_ptr = temp;
            }
            else {
                dictionary[index] = temp;
            }
        }
    }
    dictionary_file.close();
    //儲存 word
    while (getline(input_file, word)) {
        word.erase(word.end());
        input.push_back(word);
    }
    input_file.close();
    output_file << "word,answer" << endl;
    //針對每一個 input 去找
    for (string i : input) {
        std::set<string> All_Possible; //用 set 儲存第一次做字串處理
完的資料，避免重複資料同時可以排序。
        int index = HashFunction(i);
        bool check = false;
        //如果第一開始就發現是字典裡的字就寫入 OK
        if (dictionary[index]) {
            if(dictionary[index]->word == i){
                check = true;
            }
            else {
                node* now = dictionary[index];
                while (now)
                {
                    if (now->word == i) {
                        check = true;
                        break;
                    }
                    else {
                        now = now->next_ptr;
                    }
                }
            }
            if (check) {
                output_file << i + ",OK" << endl;
            }
        }
        //第一次判斷之後不在字典裡，所以開始找相似或可能的字
        if (!dictionary[index]|| !check) {
            bool check_none = false; //判斷是否為 none
            //做第一次字串處理
            Insert(i, All_Possible,0);
            Delete(i, All_Possible,0);
            Substitute(i, All_Possible,0);
            Transpose(i, All_Possible,0);

            std::set<string> All_Possible1; //用 set 儲存第二次做字
串處理完的資料，避免重複資料同時可以排序。
            //做第二次字串處理
            for (string k : All_Possible) {
                Insert(k, All_Possible1, 1);
                Delete(k, All_Possible1, 1);
                Substitute(k, All_Possible1, 1);
                Transpose(k, All_Possible1, 1);
            }
            output_file << i + ',';
            int count = 0;
            //開始找相似字
            for (string k : All_Possible1) {
                if (check_in_dictionary(k)) {
                    if(count==0)
                        output_file<< k;
                    else
                        output_file<<' '<< k;
                    check_none = true;
                    count++;
                }
            }
            if (!check_none) {
                output_file <<"NONE";
            }
            output_file << endl;
        }
    }
    cout << (clock() - START)/CLOCKS_PER_SEC << endl;
}

//check_do 判斷是否要 判斷目前這個字要不要加到 set 裡面 主要用來加速避免重複資料產生
void Insert(string& word, std::set<string>& All_Possible, bool check_do) {
    for (int i = 0; i <= word.size(); i++) {
        for (int j = 0; j < 26; j++) {
            string temp = word;
            temp.insert(temp.begin() + i, 'a' + j);
            if (check_do) {
                if (check_in_dictionary(temp)) {
                    All_Possible.insert(temp);
                }
            }
            else {
                All_Possible.insert(temp);
            }
        }
    }
}
void Delete(string& word, std::set<string>& All_Possible, bool
check_do) {
    for (int i = 0; i < word.size(); i++) {
        string temp = word;
        temp.erase(temp.begin() + i);
        if (check_do) {
            if (check_in_dictionary(temp)) {
                All_Possible.insert(temp);
            }
        }
        else {
            All_Possible.insert(temp);
        }
    }
}
void Substitute(string& word, std::set<string>& All_Possible, bool
check_do) {
    for (int i = 0; i < word.size(); i++) {
        for (int j = 0; j < 26; j++) {
            string temp = word;
            if (temp[i] == 'a' + j) {
                continue;
            }
            else
            {
                temp[i] = ('a' + j);
                if (check_do) {
                    if (check_in_dictionary(temp)) {
                        All_Possible.insert(temp);
                    }
                }
                else {
                    All_Possible.insert(temp);
                }
            }

        }
    }
}
void Transpose(string& word, std::set<string>& All_Possible, bool
check_do) {
    for (int i = 0; i < word.size()-1; i++) {
        char store = word[i];
        string temp = word;
        swap(temp[i], temp[i + 1]);
        if (check_do) {
            if (check_in_dictionary(temp)) {
                All_Possible.insert(temp);
            }
        }
        else {
            All_Possible.insert(temp);
        }
    }
}
//判斷字是否在字典裡面的功能
bool check_in_dictionary(string& word) {
    int index = HashFunction(word);
    bool check = false;
    if (dictionary[index]) {
        if (dictionary[index]->word == word) {
            check = true;
        }
        else {
            node* current = dictionary[index];
            while (current != NULL)
            {
                if (current->word == word) {
                    check = true;
                    break;
                }
                else {
                    current = current->next_ptr;
                }
            }
        }
    }
    return check;
}
//djb2 的 HashFunction
unsigned int HashFunction(string word) {
    unsigned hash = 5381;
    for (int i = 0; i < word.size(); i++) {
        hash = ((hash << 5) + hash) + word[i];
    }
    return (hash%499983);
}
```