0511105 李頤

Algorithm Homework 1 Report

A. Design Logic

簡述 ：

函式一找出任意node為起始點的單邊最長鏈的長度（圖1），函式二用函式一找出以 child node 為起始點的最長鍊，把左右兩最長鍊長度相加後（如果parent node也是母音的情況下），就可以找出以某個Node為起始點的最長鍊(圖2)，然後用函式二找出每個node的最長鍊，然後找出max(這步驟好像有點暴力)

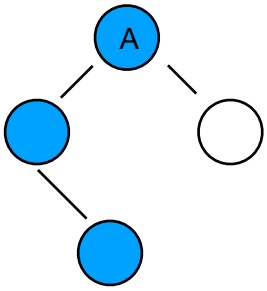


圖1：函式一找出以Node A
為起點的最長鍊（左側）

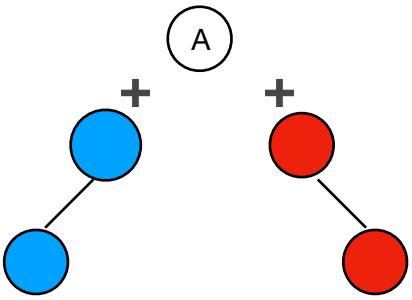


圖1：函式二分別找出以A為
起始點左右的最長鍊並相加

Pseudo Code

函式一 : 用 recursion 找出單邊最長鍊

```
longest_chain(node* curr_root):
    if( curr_root == NULL || !is_vowel(curr_root->content) ) return 0
    left_chain = longest_chain(curr_root->left_node)
    right_chain = longest_chain(curr_root->right_node)

    if(left_chain > right_chain) return 1 + left_chain

    else if (left_chain < right_chain) return 1 + right_chain
    else return 1
```

base case: 碰到不是母音的node，或是走到底了

recursion : 比較左右鍊，回傳比較長的鍊長度

函式二：

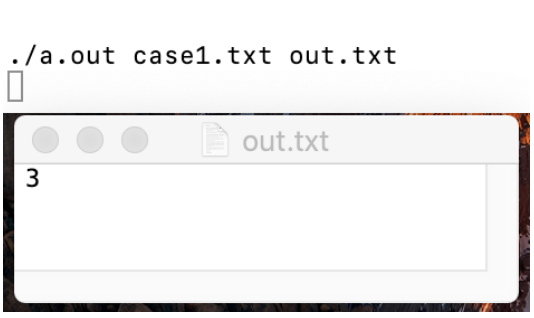
用函式一找出找出左右最長鍊再接起來，用preoder跑過每個node，找出最大長度

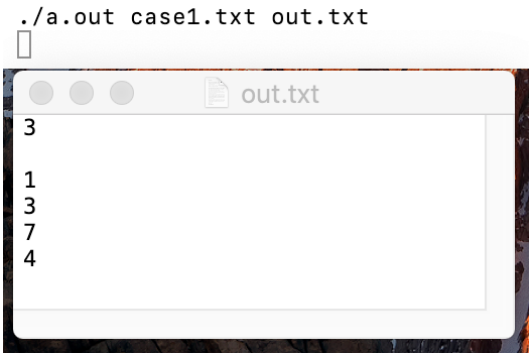
```
longest_chain_utils(node* curr_root)
    if(curr_root)
        if(is_vowel(curr_root->content))
            left_longest = longest_chain(curr_root->left_node);
            right_longest = longest_chain(curr_root->right_node);
        curr_len =  1 + left_longest + right_longest;
        if(curr_len > max_vowel_len)
            this->max_vowel_len = curr_len;

    longest_chain_utils(curr_root->left_node);
    longest_chain_utils(curr_root->right_node);
```

（不太會寫 Pseudo Code）

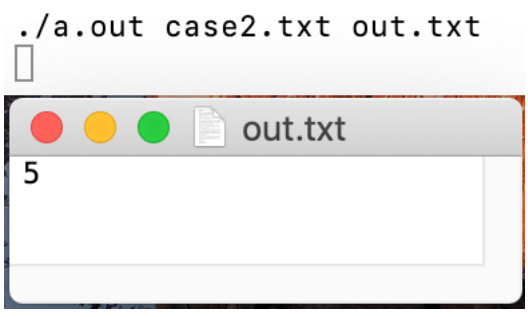
B. Result

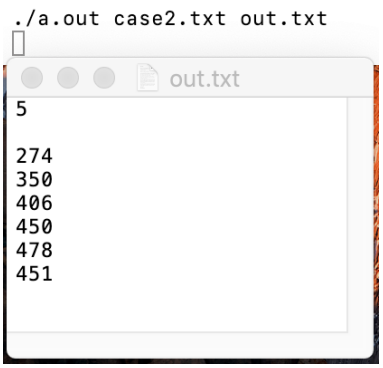
case 1 mode 0 & mode 1:





case 2 mode 0 & mode 1:





C. Program Analysis

函式一複雜度

基本上是node traveral，所以時間複雜度為O(n)

函式二複雜度

函式一每次呼叫輸入node的數量都減為1/2，像是merge sort

根據講義上的分析

$$T(n) = \begin{cases} \Theta(1) & if \quad n = 1 \\ 2T(n/2) + \Theta(n) & if \quad n > 1 \end{cases}$$

$$T(n) = \Theta(n \lg n) \quad \text{by the master theorem}$$

整體複雜度為O(nlgn)

（複雜度還沒吸收完全 寫的很低端請見諒.....