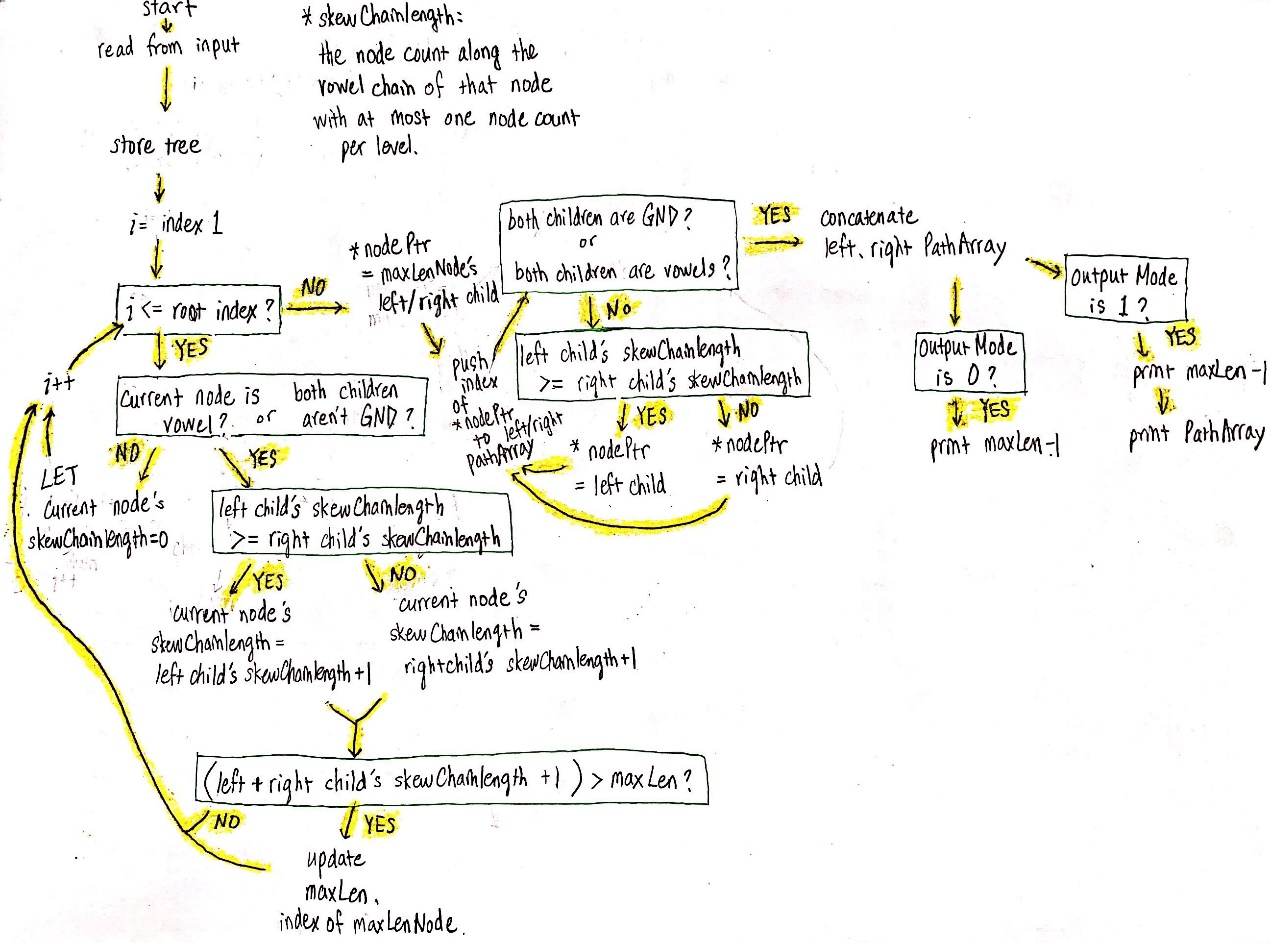
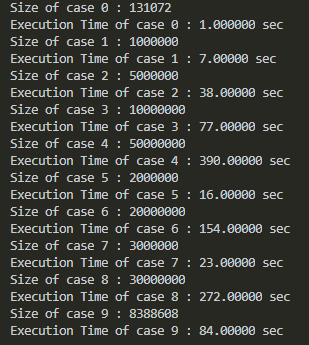
**Flow Chart of Program:**



**Experimental Results & Analysis:**

1. **設計架構分析和複雜度估計**  
   在這次的作業程式裡面，我所使用的是iterative的手法，按照輸入節點的index，每到一個節點就:  
   a. 檢查節點是否為母音，不是就繼續到下一個節點  
   b. 分別看左右小孩的母音鍊長度（在此的母音鍊算的是節點數，而且不能有任何轉折，也就是每一level至多算一個node而已），如果左、右小孩的長度再加上自己大於現有（另外整數變數存取，在此名為maxNodeChain）的最大長度就更新那個最大長度，然後把此節點的index存到另一個變數（在此名為maxIndex）  
   c. 將自己的長度更新為左或右小孩較長者的長度加1，繼續往上做。  
     
   按照這樣的規律，做到每個節點的時候其小孩的母音鍊都已經算好了，若是在output mode為0的情況下，在做完所有節點就可以輸出最大長度了（剛才所提到的節點數maxNodeChain減1，因為要輸出的是edge數），這樣的複雜度是O(n)  
     
   若output mode為1，則從先前記下的maxIndex的節點往下做跟一上步驟類似的事，分別從左右小孩開始:  
    a. 檢查其母音鍊長度是否大於0（i.e. 不是GND也不是非母音字母）  
    b. 將index存起來  
    c. 往下看他的左右小孩，比較誰的母鍊長度較長  
    d. 選擇較長者，依序往下做一樣的事直到兩小孩皆為非母音或GND  
   最後就將原先maxIndex兩小孩往下做以上步驟所經的節點index接在一起進行輸出  
     
   對此演算法的複查度分析，應該為O(h)，若再進一步以n表示樹高h，那應該就是O(lgn) （假設最壞case重從root做起，然後是一個balanced的binary tree, h = lg(n)）
2. **實際程式執行時間實驗**以上做完複雜度分析後我依然還是對執行時間十分好奇，所以除了本次作業給的範例檔外，也利用了一些額外的測資來測自己的程式，兩個範例case的execution時間太快出來都是0.000000 sec我就沒有附上了，以下<Fig.1>十個case是額外跑的測資實驗結果，其中size of case就是字母樹的節點總數。



<Fig 1.> 實際實驗數據

<Fig 2.> 數據散佈圖

將上面<Fig1.>的圖形製成趨勢圖<Fig.2>可以看到，將數據以散布圖呈現做迴歸預測，大致呈線性的關係，做完線性迴歸決定係數R2也十分接近1，證實前面所做的複雜度分析，此程式演算法的確為O(n)。