程式作業(四)

107502533

資工二A

張文耀

演算法

基本上就和老師上課講得一樣，因為要建樹，所以我先用一個structure裡面放有

1. key: 儲存這是哪一個字元
2. frequency: 這個字元出現的頻率
3. code: 這個字元最終的霍夫曼code
4. left: 指向左子樹的指標
5. right: 指向右子數的指標

接著用quick sort 將輸入進來的頻率由小到大排序，時間複雜度為

$O(n\log n)$

演算法

而我們會由兩個陣列，第一個是存輸入資料的Node，第二個是存指向這些Node的指標。

接著用quick sort 將指標陣列指向Node的頻率由小到大排序，時間複雜度為$O(n \log n)$

所以我們的

Node 陣列是依照字母排序

Node* 陣列是依照頻率排序

演算法

接著把排序好的Node* 陣列push進一個queue，老師上課的作法是push進一個priority queue，然後每次pop兩個，把這兩個建成樹後再insert進priority queue裡，因此最壞的情形是每次都要insert到最前面，這樣時間複雜度有可能會來到$O(n^2)$。

因此我的作法是除了原本排序好的queue，再多創第二個空的queue，接著每次從兩個queue裡選最小的兩個pop出來，建成樹後再push進第二個queue，這樣就能保證兩個queue都是priority queue的形式，而且只要做n-1次就能夠完成，所以這個做法的時間複雜度為$O(n)$。

最後第二個queue會只剩下一個指標(root的指標)，回傳它。

演算法

再用得到的root指標去遞迴，產生code，

左子樹給現有的code + ‘0’

右子樹給現有的code + ‘1’

所有的Node都會訪問到，而所有的Node總數為2n-1個，因此此步驟的時間複雜度為$O(n)$。

所以最終的時間複雜度為$O(n\log n)$

Pseudo code

```
Structrue Node{key, frequency, code, left, right}
nodeArr: Node 陣列
nodeptr: Node* 陣列
quickSort(nodeptr) //讓陣列一小到大排序
buildTree(nodeptr, num: 陣列長度)
    q1.push(nodeptr) //陣列push進queue
    for I = 0 to n-1
        Node* temp = new Node //創新Node
        pop(small1, small2) from(q1, q2)//從q1,q2中pop出最小的兩個指標
        temp->left = small1, temp->right = small2
        q2.push(temp) //把建好的Node push進q2
    return q2.front()  //回傳root
```

Pseudo code

```
generateCode(root, tempCode: 當前的code)
    if(root == NULL)
        return
    root->code = tempCode
    generateCode(root->left, tempCode + ‘0’)
    generateCode(root->right, tempCode + ‘1’)

For I = 0 to n-1
    print(nodeArr[i].key + “ “ + nodeArr[i].code)
```