FIT1008 Introduction to Computer Science
Practical Session 10

Semester 2, 2014

Objectives of this practical session

To understand recursive sorting algorithms and binary trees.

Testing

For this prac, you are required to write:

(1) a *function to test* each part of the functionality of the programs you implement (*at least two test cases*). The cases need to show that your functions can handle both valid and invalid inputs.

(2) programs with a sensible level of modularity.

Built in Functions

Important: Before doing this prac, you should have a look at the buitin functions in Python. A good reference is:

- `https://docs.python.org/3.3/library/functions.html` from *The Python Tutorial*

Background

This prac is about working with text files, binary trees, and text compression.
All text files consists characters represented by a string of bits, usually all of the same length. However, if you encode the characters by strings of variable lengths, where the length depends on how the frequent the character occurs, you can compress the size of the file.
The first task in this prac considers counting characters in files and constructing simple text files that only consist of lowercase letters.
The second task explores the connection between positions in a binary tree and binary strings, i.e., strings that only consists of '0's and '1's. The third task uses binary trees to encode and decode text files into a binary string. The Advanced Question investigates other ways of encoding characters, using Morse Code. Finally, in the Hall of Fame, we use the frequency of the characters in a text file and binary trees to construct a Huffman code. These codes are very useful for compressing files.

Task 1 [3 marks]

Many problems in computer science involving reading files one character at a time.

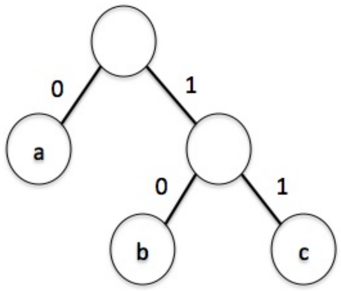
(i) Write a Python program that reads a file name and prints out how many characters are in the file.

(ii) Write a Python program that reads an input file name and output file name. This program should read each character from the input file. If the character is a letter, then the letter should be made lowercase and then written to the output file. Otherwise do not write the character to the output file.

(iii) Write a Python program that reads an input file name. Assume that the input file only consists of lowercase alphabetical letters. Keep count of how many times each character is read in. Then using your own implementation of *merge sort*, sort the letters into decreasing order of frequency. Finally, print out all the characters and their corresponding frequency, in decreasing order of frequency.

Task 2 [3 marks]

The position of every node in a binary tree can be represented by a binary string, i.e., a string that consists of only '0's and '1's. The position is determined by starting at the root of the tree and moving down the tree by going *left* when the character is '0', and going *right* when the character is '1'. For example in the following tree the positions of 'a', 'b' and 'c' can be represented by '0', '10', and '11', respectively.



Using the following code:

```python
class TreeNode:
    def __init__(self, item, left, right):
        self.item = item
        self.right = right
        self.left = left

class BinaryTree:
    def __init__(self):
        self.root = None

    def add(self, item, binary_str_itr):
        self.root = self.add_aux(self.root, item, binary_str_itr)

    def add_aux(self, current, item, binary_str_itr):
        if current is None:
            current = TreeNode(None, None, None)

        try:
            bit = next(binary_str_itr)
            if bit == '0':
                current.left = self.add_aux(current.left, item, binary_str_itr)
            elif bit == '1':
                current.right = self.add_aux(current.right, item, binary_str_itr)
        except StopIteration:
            current.item = item

        return current
```

write a Python program that implements a Binary Tree and allows a user to perform the following commands on the Binary Tree using a menu:

add item binary_str: using the *add* method insert an *item* in the Binary tree in the position represented by the binary string, *binary_str*.

get binary_str: prints the item stored in the Binary tree in the position represented by the binary string, *binary_str*.

print: using inorder traversal prints all the items in the Binary tree.

quit: which quits the program.

Task 3 [4 marks]

(i) Write a Python program that reads in an input and output file name, reads in the characters from the input file (only consisting of lowercase letters), and writes out the ASCII code of each character (as a binary string) to the output file. (So, the output consists of a single binary string)

(ii) Using your implementation of a Binary Tree, construct a binary tree that contains each lowercase letter in the position specified by their ASCII code.

(iii) Write a Python program that reads in an input and output file name and uses the Binary Tree you constructed in part (ii). This program should read in the binary string in the input file[1], convert the string back into the corresponding sequence of characters, and write the characters into the output file.[2]

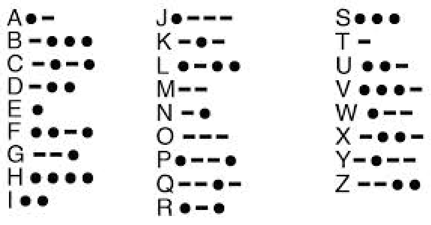
[1] This file should have been generated by the program in part (i)

[2] This file should be the same as the input file that generated, using the program in part (i), the binary string you read in.

Advanced Question [Bonus 2 marks]

Consider the International Morse Code. [3]

[3] `http://en.wikipedia.org/wiki/Morse_code`



A	•–	J	•–––	S	•••
B	–•••	K	–•–	T	–
C	–•–•	L	•–••	U	••–
D	–••	M	––	V	•••–
E	•	N	–•	W	•––
F	••–•	O	–––	X	–••–
G	––•	P	•––•	Y	–•––
H	••••	Q	––•–	Z	––••
I	••	R	•–•		

You will notice that some of the codes (e.g, the code for 'E') are the prefix of another code (e.g. the code for 'S'). This makes it difficult to decode messages unambiguously without adding additional gaps between letters, and/or between the dots and dashes within the code for a letter.

(i) Using the following representation for the International Morse Code, write a Python program to encode and decode files.

(a) a dot by '1'.

(b) a dash by '111'.

(c) the gap between dots and dashes within a code for a character by '0'

(d) the end of a character by '000'

For example, *'sos'* would be represented by *'101010001110111011100010101000'*.

(ii) Take a few of the texts in Australian Project Gutenberg repository `http://gutenberg.net.au`, remove any characters in the texts which are lower or uppercase letters, and convert every uppercase letter to a lowercase letter. Then for each text compare the length of the binary string produced using the ASCII code to that produced by the Morse code.

ASCII is shorter

(iii) Find another way to represent the International Morse Code, so that:

(a) in the Binary Tree representing the encoding every letter corresponds to a leaf.

(b) for every text you took from the repository, the binary string generated for your encoding is shorter than using the above representation of the Morse Code.

Change 111 to 11, and 000 to 00

Hall of Fame

Write a Python program which, for the texts you took from the repository, finds a Huffman code[4] for all the lowercase letters.

To find the Binary Tree representing the Huffman code you need to do the following:

[4] `http://en.wikipedia.org/wiki/Huffman_coding`

1. Find the frequency of each lowercase letter in the texts you took from the repository.
2. Construct a list of Binary Trees, where each Binary Tree corresponds to a lowercase letter and has two instance variables:
   - the root which points to a node whose item is the letter, and
   - the weight of the tree which is the frequency of the letter.
3. Repeat the following until you have one Binary Tree in the list of Binary Trees.
   (a) Pick the two Binary Trees, *tree1* and *tree2*, with the smallest weights.
   (b) Construct a new Binary Tree so that the left subtree of the root is *tree1*, the right subtree of the root is *tree2*, and its weight is the sum of the weights of *tree1* and *tree2*.
   (c) Add the new Binary tree to the list of Binary trees, and remove *tree1* and *tree2* from the list.