Homework 4

Due 10/09/19

October 2, 2019

1. Give pseudocode for a divide-and-conquer algorithm that accepts a node in an *uncompressed* prefix tree, and returns the number of words stored in the prefix tree rooted at that node.

   You may assume that every node has two fields: `IsFinal`, a Boolean value indicating whether this node is the end of a word, and `Link`, an array of 26 pointers that link to the children of this node (NIL if a child doesn't exist). You may use letters or numbers as indices into the `Link` array (e.g., *node*.Link[a] or *node*.Link[1]).

Input: node, a node in uncompressed prefix tree

Output: Count, # of words stored in prefix tree starting from that node.

```
CountWords
    Count=0
for i in range(26):
|   if node.Link[i] != NIL:
|   |   Count += CountWords(node.Link[i])
if node.isFinal:
    Count++
return Count
```

2. Consider the problem where you are given an array of n digits $[d_i]$ and a positive integer b, and you need to compute the value of the number in that base.

   In general, you need to compute

$$x = d_0 b^0 + d_1 b^1 + \ldots + d_{n-1} b^{n-1} = \sum_{i=0}^{n-1} d_i b^i.$$

   For example:

$$\begin{aligned}(1011)_2 &= 1(1) + 1(2) + 0(4) + 1(8)\\ &= 11,\\ (1021)_3 &= 1(1) + 2(3) + 0(9) + 1(27)\\ &= 34, \text{ and}\\ (1023)_4 &= 3(1) + 2(4) + 0(16) + 1(64)\\ &= 75.\end{aligned}$$

   In these examples, I give the digits in the order $d_3 d_2 d_1 d_0$, which corresponds to how we would normally write these numbers, though you can assume that d_i is in index i of the array for the questions below. (Yes, the indices will be numbered 0 to $n-1$, not 1 to n.)

   (a) Give pseudocode for a divide-and-conquer algorithm that solves this problem by dividing the digit array into two subarrays of (roughly) the same size.

   For example, $d_5 d_4 d_3 d_2 d_1 d_0$ would be split into $d_5 d_4 d_3$ and $d_2 d_1 d_0$.

   (b) Give pseudocode for a divide-and-conquer algorithm that solves this problem by dividing the digit array into two *interleaved* arrays of (roughly) the same size.

   For example, $d_5 d_4 d_3 d_2 d_1 d_0$ would be split into $d_5 d_3 d_1$ and $d_4 d_2 d_0$.

a) Input arr, array of n digits
Input: b, base to Convert to
Output: arr_b

ConvertToBase:
    let n = arr.size( )
    if n = 1:
        return arr[0]
    if n is odd:
        let subArr1 = arr[0 - ((n-1)/2 - 1)]
        let subArr2 = arr[((n+1)/2 - 1) - (n-1)]
        let exp = ((n-1)/2 + 1)

    else:
        let subArr1 = arr[0 - (n/2 - 1)]
        let subArr2 = arr[(n/2) - (n-1)]
        let exp = n/2

    return (ConvertToBase(SubArr1, b) b^{exp}
        + (ConvertToBase(SubArr2, b))

B)

Input arr, array of n digits

Input: D, base to Convert to
Output. Arr_b

ConvertToBase:

Arr1 = BaseConv(arr, b, 0)
Arr2 = BaseConv(arr, b, 1)
> utility function

return Arr1 + Arr2

Input: arr, the array of n digits
Input: b, base to Convert to
Input: i, starting index
Output: Sum of even or odd Values from arr, in base b.

BaseConv:

```
if (i >= arr.Size()):
    return 0
else
    return arr[i] · b^i + BaseConv(arr, b, i+2)
```