INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

Análisis de Algoritmos

Práctica Sesión 7
Algoritmos ávidos

Edgar Adrián Nava Romo

Maestra: Sandra Díaz Santiago
Grupo: 3CM3

Código Fuente

```
1.  def assign_code(nodes, label, result, prefix = ''):
2.      childs = nodes[label]
3.      tree = {}
4.      if len(childs) == 2:
5.          tree['0'] = assign_code(nodes, childs[0], result, prefix + '0')
6.          tree['1'] = assign_code(nodes, childs[1], result, prefix + '1')
7.          return tree
8.      else:
9.          result[label] = prefix
10.         return label
11.
12. def Huffman_code(_vals):
13.     vals = _vals.copy()
14.     nodes = {}
15.     for n in vals.keys(): # leafs initialization
16.         nodes[n] = []
17.
18.     while len(vals) > 1: # binary tree creation
19.         s_vals = sorted(vals.items(), key=lambda x:x[1])
20.         a1 = s_vals[0][0]
21.         a2 = s_vals[1][0]
22.         vals[a1 + a2] = vals.pop(a1) + vals.pop(a2)
23.         nodes[a1 + a2] = [a1, a2]
24.     code = {}
25.     root = a1 + a2
26.     tree = {}
27.     tree = assign_code(nodes, root, code)   # assignment of the code fo
    r the given binary tree
28.     return code, tree
29.
30. freq = [
31. (6.8, 'a'), (1.3, 'b'), (2.6, 'c'), (3.5, 'd'),
32. (10.2, 'e'),(1.8, 'f'), (1.7, 'g'), (4.9, 'h'),
33. (5.8, 'i'), (0.2, 'j'), (0.6, 'k'), (3.4, 'l'),
34. (2.1, 'm'), (5.5, 'n'), (5.9, 'o'), (1.6, 'p'),
35. (0.1, 'q'), (4.8, 'r'), (5.1, 's'), (7.7, 't'),
36. (2.4, 'u'), (0.9, 'v'), (1.9, 'w'), (0.2, 'x'),
37. (1.6, 'y'), (0.1, 'z'), (18.3, ' ') ]
38. vals = {l:v for (v,l) in freq}
39. code, tree = Huffman_code(vals)
40.
41. text = 'hola' # text to encode
42. encoded = ''.join([code[t] for t in text])
43. print('Encoded text:',encoded)
```

*Código propuesto para imprimir el código de Hoffman en forma de árbol:

```
1.  import subprocess
2.  with open('graph.dot','w') as f:
3.      f.write('digraph G {\n')
4.      f.write(draw_tree(tree))
5.      f.write('}')
6.  subprocess.call('dot -Tpng graph.dot -o graph.png', shell=True)
```

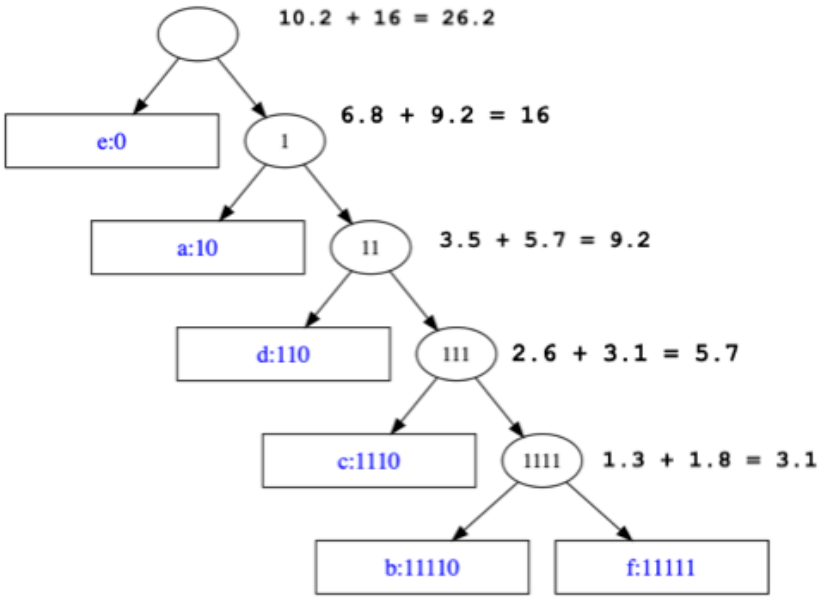
Pseudocódigo

```
1. Procedure Huffman(C):            # C is the set of n characters and
   related information
2. n = C.size
3. Q = priority_queue()
4. for i = 1 to n                   # leafs initialization
5.     n = node(C[i])
6.     Q.push(n)
7. end for
8. while Q.size() is not equal to 1 # binary tree creation
9.     Z = new node()
10.    Z.left = x = Q.pop
11.    Z.right = y = Q.pop
12.    Z.frequency = x.frequency + y.frequency
13.    Q.push(Z)        # assignment of the code for the given binary tree
14.end while
15.Return Q
```

La Complejidad de este algoritmo es de O(n) aunque en la mayoría de los casos suele dar O(n log n) ya que los árboles binarios suelen tener este tiempo en la práctica.

Ejemplo

(6.8, 'a'), (1.3, 'b'), (2.6, 'c'), (3.5, 'd'), (10.2, 'e'),(1.8, 'f')



Empezamos de abajo hacía arriba a armar el árbol desde los que tienen menos valor hasta los que tienen más valor, se van sumando los valores y se van comparando con el valor que sigue, al final se convierte el valor a binario donde los de la izquierda son 0's y los de la izquierda son 1's

Comparación a la Tarea Códigos de Huffman

De acuerdo a la tarea anterior donde se tenía que hacer un código de Huffman para el alfabeto español, los valores dieron igual a la siguiente imagen generada por el código encontrado para la implementación del código de Huffman usando la librería graphviz en python 3