s

Instituto Politécnico Nacional
Escuela Superior de Computo

Análisis de algoritmos

Sandra Díaz Santiago

Practica 4: Algoritmos ávidos

Alumno:

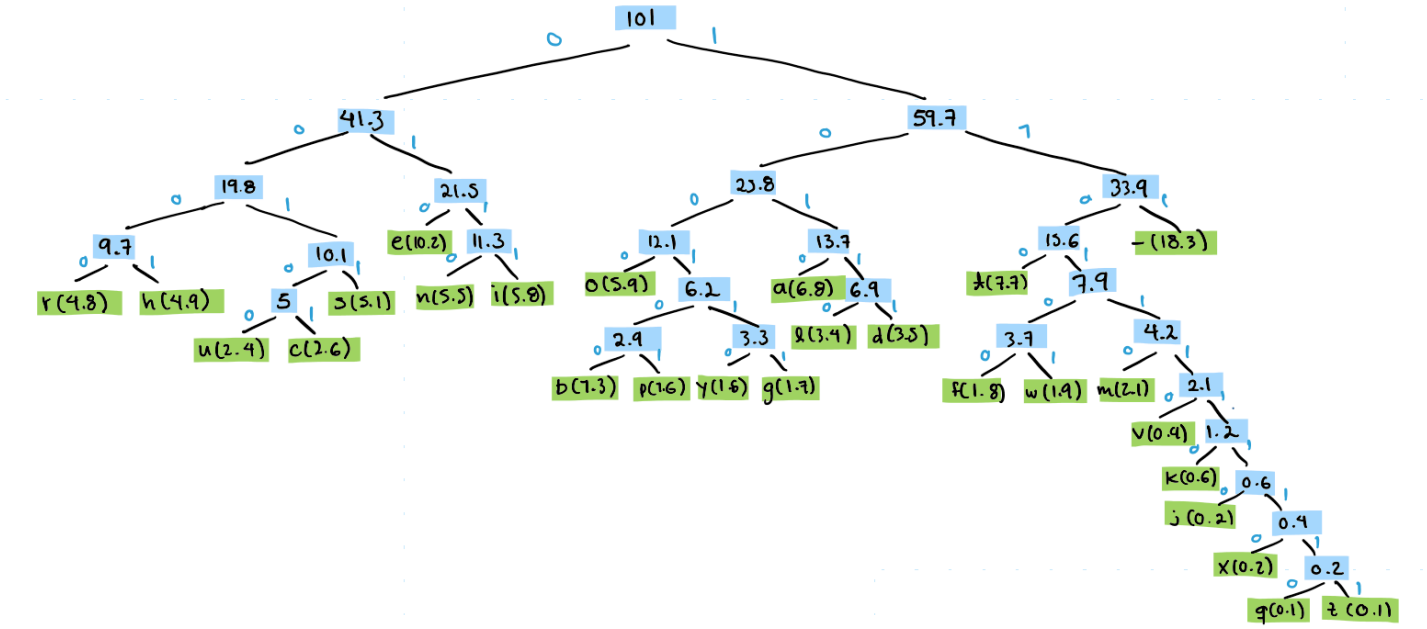
- Vallejo serrano Ehecatzin.
- García González Aarón Antonio

Octubre, 2019

1. La siguiente tabla muestra las frecuencias de las letras del alfabeto inglés, incluyendo el espacio en blanco para separar las palabras, de cierto conjunto de textos.

espacio	18.3 %	r	4.8 %	y	1.6 %
e	10.2 %	d	3.5 %	p	1.6 %
t	7.7 %	l	3.4 %	b	1.3 %
a	6.8 %	c	2.6 %	v	0.9 %
o	5.9 %	u	2.4 %	k	0.6 %
i	5.8 %	m	2.1 %	j	0.2 %
n	5.5 %	w	1.9 %	x	0.2 %
s	5.1 %	f	1.8 %	q	0.1 %
h	4.9 %	g	1.7 %	z	0.1 %

¿Cuál es la codificación de Huffman, para estos datos? Proporciona el árbol correspondiente.



Codificación

–	1 1 1	r	0 0 0 0	y	1 0 0 1 1 0
e	0 1 0	d	1 0 1 1 1	p	1 0 0 1 0 1
t	1 1 0 0	l	1 0 1 1 0	b	1 0 0 1 0 0
a	1 0 1 0	c	0 0 1 0 1	v	1 1 0 1 1 1
o	1 0 0 0	u	0 0 1 0 0	k	1 1 0 1 1 1 1 0
i	0 1 1 1	m	1 1 0 1 1 0	j	1 1 0 1 1 1 1 1 0
n	0 1 1 0	w	1 1 0 1 0 1	x	1 1 0 1 1 1 1 1 1 0
s	0 0 1 1	f	1 1 0 1 0 0	q	1 1 0 1 1 1 1 1 1 1 0
h	0 0 0 1	g	1 0 0 1 1 1	z	1 1 0 1 1 1 1 1 1 1 1

Realiza una implementación que encuentre la codificación de Huffman, utilizando una cola de prioridad y compara con el resultado que obtuviste previamente.

```
from heapq import heappush, heappop, heapify
from collections import defaultdict

def encode(symb2freq):
    """Huffman encode the given dict mapping symbols to weights"""
    heap = [[wt, [sym, ""]] for sym, wt in symb2freq.items()] # frecuencia, [simbolo,codificacion]
    heapify(heap) # Transforma la lista x en un heap en su lugar, en tiempo lineal.
    while len(heap) > 1:
        # Devolver el elemento más pequeño del heap, manteniendo el heap invariante.
        lo = heappop(heap) # izquierdo
        hi = heappop(heap) # derecho
        for pair in lo[1:]: # tomamos solo [simbolo,codificacion] en izquierdo
            pair[1] = '0' + pair[1] # pair[1] representa solo las codificaciones
        for pair in hi[1:]: # tomamos solo [simbolo,codificacion] en derecho
            pair[1] = '1' + pair[1]
        #print([round(lo[0] + hi[0],2)] + lo[1:] + hi[1:])
        heappush(heap, [round(lo[0] + hi[0],2)] + lo[1:] + hi[1:]) # mete la suma de los dos lados,
 asi como el arbol/nodo de cada lado
    return sorted(heappop(heap)[1:], key=lambda p: (len(p[-
1]), p)) # ordenamos una copia con base en sus codigos


frecuencias = defaultdict(float)
frecuencias['_']=18.3
frecuencias['e']=10.2
frecuencias['t']=7.7
frecuencias['a']=6.8
frecuencias['o']=5.9
frecuencias['i']=5.8
frecuencias['n']=5.5
frecuencias['s']=5.1
frecuencias['h']=4.9
frecuencias['r']=4.8
frecuencias['d']=3.5
frecuencias['l']=3.4
frecuencias['c']=2.6
frecuencias['u']=2.4
frecuencias['m']=2.1
frecuencias['w']=1.9
frecuencias['f']=1.8
frecuencias['g']=1.7
frecuencias['y']=1.6
frecuencias['p']=1.6
frecuencias['b']=1.3
frecuencias['v']=0.9
frecuencias['k']=0.6
```

```
frecuencias['j']=0.2
frecuencias['x']=0.2
frecuencias['q']=0.1
frecuencias['z']=0.1


huff = encode(frecuencias)
print ("Symbolo\tFrecuencia\tCódigo Huffman")
for p in huff:
    print ("%s\t%s\t\t%s" % (p[0], frecuencias[p[0]], p[1]))
```

```
MacBook-Pro-de-Aaron:h4 aarongarcia$ python3 huffman1.py
Symbolo Frecuencia      Código Huffman
_       18.3            111
e       10.2            010
a       6.8             1010
h       4.9             0001
i       5.8             0111
n       5.5             0110
o       5.9             1000
r       4.8             0000
s       5.1             0011
t       7.7             1100
c       2.6             00101
d       3.5             10111
l       3.4             10110
u       2.4             00100
b       1.3             100100
f       1.8             110100
g       1.7             100111
m       2.1             110110
p       1.6             100101
w       1.9             110101
y       1.6             100110
v       0.9             1101110
k       0.6             11011110
x       0.2             110111110
j       0.2             1101111110
q       0.1             11011111110
z       0.1             11011111111
```

2. Cierta compañía ha desarrollado discos duros ternarios. Cada celda en un disco puede almacenar valores 0,1,2 (no solo 0 o 1). Para aprovechar esta nueva tecnología, construye un código de Huffman modificado para comprimir secuencias de caracteres de un alfabeto de tamaño n, cada uno de los cuales tiene una frecuencia f1, f2, . . . , fn. Tu algoritmo debería codificar cada carácter usando una palabra ternaria (usando los valores 0,1,2), de tal manera que ningún código sea prefijo de otro y así obtener una compresión máxima. Calcula la complejidad de tu algoritmo. Justifica por qué tu algoritmo tiene una compresión máxima. Implementa tu algoritmo en algún lenguaje de programación.

Alicia quiere hacer una fiesta y está haciendo la lista de invitados. Ella tiene una lista con n personas y también una segunda lista de pares de personas que se conocen. Alicia quiere invitar a tantas personas como sea posible, con dos condiciones: en la fiesta cada persona debe conocer al menos a otras 5 personas y debe haber otras 5 personas a las cuales no conoce. Diseña un algoritmo que tome como entrada la lista de n invitados y las listas de pares de personas y que como salida de la lista de invitados. Calcula la complejidad de tu algoritmo. Implementa tu algoritmo en algún lenguaje de programación.

Solución:

Éste problema fue resuelto con un grafo , por conveniencia éste grafo está representado en forma de lista de Adyacencia.
Explicaré una parte de la solución del problema a forma de introducción, con la siguiente entrada:

personas = ["Karla", "Catzin","Hueman","Ernesto","Jaen","Alfredo"]

conocidos = [("Karla","Catzin"),("Karla","Ernesto"),("Karla","Jaen"),("Karla","Alfredo"),("Karla","Hueman"),
("Jose","Lucia"),("Laura","Samuel"),("Samuel","Catzin"),("Angel","Israel")]

Lo primero es construir la matriz de adyacencia , la cual estará conformada por listas , una lista de personas conocidas por cada persona en la lista “personas” . Un caso a considerar a demás es cuando alguna persona , no conoce a alguien en la lista de “conocidos” , entones crearé una lista con los no conocidos.
Considerando las posiciones [] de personas, podemos construir la siguiente lista de adyacencia:

*nota : cada posición de la lista de adyacencia es una lista de los conocidos por cada persona en “personas”

Karla catzin Hueman Ern Jaen Alfredo

[['Catzin', 'Ernesto', 'Jaen', 'Alfredo', 'Hueman'] , ['Karla', 'Samuel'] , [Karla], [Karla], [Karla], [Karla]]

La lista de no conocidos:

['Jose', 'Lucia', 'Laura', 'Samuel', 'Angel', 'Israel']

Una vez que tenemos idea de esto , lo siguiente sería cumplir las condiciones:

- En la fiesta cada persona debe conocer al menos a 5

Esto quiere decir que solamente meteremos a nuestra lista final de invitados, a aquella lista en la lista de adyacencia que tenga 5 o más elementos , no menos.

- Debe haber 5 personas no conocidas

  Una vez que tengamos nuestra lista de no conocidos , solamente deberemos agregar 5 de estos
  A la lista final de invitados.

  Diseño del algoritmo

  ListarConocidos(personas [] , conocidos []){

  ListaAdyacencia = [[] , [] . . .] //lista de listas

  Otros = [ ] //lista de pares (x,y) -- (persona,conocido)

  For i = 0 to len (personas[])
  For j to len (conocidos[])
  -> busca a persona[i] en conocido[j]
  Si está -> agrega a "y" de conocido[j] en listaAdyacencia[i]

  Si no -> agrega a conocido[j] en Otros[ ]

  Return listaAdyacencia && Otros

  }

  //Esta función retorna la lista Oficial de invitados

  ListarInvitados(personas[], listaAdyacencia[] , Otros [ ]){

  InvitadosConocidos = [ ]

  For i = 0 to len(listaAdyacencia)

  If (longitud(listaAdyacencia[i] >= 5)
  //agrega al primer invitado de la lista Personas
  InvitadosConocidos.add(personas[i])

  //Ahora agrega a todos los que esta persona conoce
  For k = 0 to len (listaAdyacencia[i])

InvitadosConocidos.add(Otros[k])

Return invitadosConocidos

}

Implementación

```
def listarConocidos(personas,conocidos):
    listaAdyacencia = iniciarListaAdyacencia(len(personas))
    otros =  [ ]

    for i in range(len(personas)):

        for conocido in conocidos:
            if(conocido[0] is personas[i]):

                listaAdyacencia[i].append(conocido[1])

            elif(conocido[1] is personas[i]):

                listaAdyacencia[i].append(conocido[0])
            else:
                if(not otros):
                    otros.append(conocido[0])
                    otros.append(conocido[1])
                else:
                    if(conocido[0] not in otros):
                        otros.append(conocido[0])
                    elif(conocido[1] not in otros):
                            otros.append(conocido[1])

    return listaAdyacencia,otros
```

```
35
36  def listarInvitados(personas,listaAdyacencia , noConocidos):
37      invitadosConocidos = [ ] #personas conocidas por los que están en la primera lista de personas
38
39      #eliminamos de no conocidos a las personas iniciales (personas)
40      for persona in personas:
41          if(persona in noConocidos):
42              noConocidos.pop(noConocidos.index(persona))
43
44      for i in range(len(listaAdyacencia)):
45
46          if(len(listaAdyacencia[i]) >= 5):
47              invitadosConocidos.append(personas[i])
48
49              for k in listaAdyacencia[i]:
50
51                  invitadosConocidos.append(k)
52
53
54      return invitadosConocidos+noConocidos
55
56
57
```

Entrada y pruebas:

personas = ["Karla", "Catzin","Hueman","Ernesto","Jaen","Alfredo"]

conocidos = [("Karla","Catzin"),("Karla","Ernesto"),("Karla","Jaen"),("Karla","Alfredo"),("Karla","Hueman"),
("Jose","Lucia"),("Laura","Samuel"),("Samuel","Catzin"),("Angel","Israel")]

```
personas = [ "Karla", "Catzin","Hueman","Ernesto","Jaen","Alfredo"]

conocidos = [("Karla","Catzin"),("Karla","Ernesto"),("Karla","Jaen"),("Karla","Alfredo"),("Karla","Hueman"),
    ("Jose","Lucia"),("Laura","Samuel"),("Samuel","Catzin"),("Angel","Israel")]


listaAdyacencia,noConocidos = listarConocidos(personas,conocidos)

invitacionFormal = listarInvitados(personas,listaAdyacencia,noConocidos)

print("lista de personas:\n")
```

Salida:

```
catzin@catzin-MacBookAir:~/Desktop$ python3 alicia.py
Lista de personas:

['Karla', 'Catzin', 'Hueman', 'Ernesto', 'Jaen', 'Alfredo']


Lista de pares de conocidos:

[('Karla', 'Catzin'), ('Karla', 'Ernesto'), ('Karla', 'Jaen'), ('Karla', 'Alfredo'), ('Karla', 'Hueman'), ('Jose', 'Lucia'), ('Laura',
'Samuel'), ('Samuel', 'Catzin'), ('Angel', 'Israel')]


Karla conoce a:

['Catzin', 'Ernesto', 'Jaen', 'Alfredo', 'Hueman']


Catzin conoce a:

['Karla', 'Samuel']


Hueman conoce a:

['Karla']


Ernesto conoce a:

['Karla']


Jaen conoce a:

['Karla']
```

```
catzin@catzin-MacBookAir: ~/Desktop
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

Catzin conoce a:

['Karla', 'Samuel']


Hueman conoce a:

['Karla']


Ernesto conoce a:

['Karla']


Jaen conoce a:

['Karla']


Alfredo conoce a:

['Karla']


lista de no conocidos:

['Jose', 'Lucia', 'Laura', 'Samuel', 'Angel', 'Israel']


invitacion formal:

['Karla', 'Catzin', 'Ernesto', 'Jaen', 'Alfredo', 'Hueman', 'Jose', 'Lucia', 'Laura', 'Samuel', 'Angel', 'Israel']
catzin@catzin-MacBookAir:~/Desktop$
```