

Implementación de Huffman

Dalia Camacho, Gabriela Vargas, Elizabeth Monroy

Implementación

En esta sección utilizamos el código de Huffman para obtener códigos variables para cada caracter dentro de un texto. Para lograr esto se crea un árbol binario que se determina a partir de la frecuencia de caracteres. Además, obtenemos el código de bits para cada caracter recorriendo el árbol en profundidad con pre-orden y asignando un valor “0” a la rama izquierda y un valor “1” a la rama derecha.

Para poder definir árboles en R utilizamos el paquete `data.tree`

```
library(data.tree)
```

La función *Huffman* recibe como entrada el texto para el que se obtendrán los códigos de caracteres y tiene la opción de imprimir cada paso si la opción *Print=TRUE*. El default es *Print=FALSE*. Esta función entrega como salida el árbol que se genera con el algoritmo de Huffman.

```
Huffman <- function(text, Print = FALSE){
  # Se obtienen las frecuencias de los caracteres dentro del texto
  Frequencies <- table(strsplit(text, "")[[1]])
  # Se define un ambiente dentro de la función en el que se
  # guardan los árboles de pasos intermedios
  efun <- environment()
  # n es el número de caracteres en el texto
  n <- dim(Frequencies)
  # Se inicializa Q como la tabla de caracteres y frecuencias
  Q <- Frequencies
  # Se inicia el ciclo para generar árboles con los dos valores menores
  for(i in 1:(n-1)){
    # Se define la raíz de un nuevo árbol
    Newtree <- Node$new(paste0("Tree ", i,
                                "\n Count: ",
                                min(Q)+min(Q[-min(Q)])))
    # Su hijo de la rama izquierda es el objeto con menor frecuencia en Q,
    # ya sea un caracter o un árbol construido previamente
    x.freq <- min(Q)
    if(substr(names(which.min(Q)),3,3+nchar(n))==""){
      Newtree$AddChild(paste0("Char: ", names(which.min(Q))[1], " \n Freq: ", x.freq))
    }else{
      Newtree$AddChildNode(get(
        paste0("Tree_", substr(names(which.min(Q))[1],3,5)), envir = efun))
    }
    # Se elimina el mínimo del arreglo Q
    Q <- Q[-(which.min(Q)[1])]
    # Ahora el hijo de la rama derecha es el mínimo en Q y se repite
    # el proceso realizado en la rama izquierda
    y.freq <- min(Q)
    if(substr(names(which.min(Q)),3,5))==""){
      Newtree$AddChild(paste0("Char: ", names(which.min(Q))[1], " \n Freq: ", y.freq))
    }else{
      Newtree$AddChildNode(get(
        paste0("Tree_", substr(names(which.min(Q))[1],3,5)), envir = efun))
    }
  }
}
```

```

}
Q      <- Q[-(which.min(Q)[1])]
# Se agrega el árbol generado como un nuevo nodo en Q cuya frecuencia
# es la suma del hijo de la rama izquierda y el de la derecha
z.freq <- x.freq + y.freq
Q <- c(Q, z.freq)
names(Q)[length(Q)] <- paste0("T_",i)
if(Print){
  print(Newtree)
}

assign(paste0("Tree_",i), Newtree, envir = efun)
}
# Se regresa el último árbol generado
return(get(paste0("Tree_", i), envir = efun))
}

```

Hacemos un ejemplo pequeño mostrando la construcción del árbol con el texto “Hello World”. Más adelante se hará con un texto más grande.

```
HelloTree <- Huffman("Hello World", Print = TRUE)
```

```

##           levelName
## 1 Tree 1 \n Count: 2
## 2 |--Char:  \n Freq: 1
## 3 °--Char: d \n Freq: 1
##           levelName
## 1 Tree 2 \n Count: 2
## 2 |--Char: e \n Freq: 1
## 3 °--Char: H \n Freq: 1
##           levelName
## 1 Tree 3 \n Count: 2
## 2 |--Char: r \n Freq: 1
## 3 °--Char: W \n Freq: 1
##           levelName
## 1 Tree 4 \n Count: 4
## 2 |--Char: o \n Freq: 2
## 3 °--Tree 1 \n Count: 2
## 4   |--Char:  \n Freq: 1
## 5   °--Char: d \n Freq: 1
##           levelName
## 1 Tree 5 \n Count: 4
## 2 |--Tree 2 \n Count: 2
## 3 |   |--Char: e \n Freq: 1
## 4 |   °--Char: H \n Freq: 1
## 5 °--Tree 3 \n Count: 2
## 6   |--Char: r \n Freq: 1
## 7   °--Char: W \n Freq: 1
##           levelName
## 1 Tree 6 \n Count: 6
## 2 |--Char: l \n Freq: 3
## 3 °--Tree 4 \n Count: 4
## 4   |--Char: o \n Freq: 2
## 5   °--Tree 1 \n Count: 2
## 6     |--Char:  \n Freq: 1

```

```
## 7          °--Char: d \n Freq: 1
##                               levelName
## 1 Tree 7 \n Count: 8
## 2 |--Tree 5 \n Count: 4
## 3 |   |--Tree 2 \n Count: 2
## 4 |   |   |--Char: e \n Freq: 1
## 5 |   |   °--Char: H \n Freq: 1
## 6 |   °--Tree 3 \n Count: 2
## 7 |       |--Char: r \n Freq: 1
## 8 |       °--Char: W \n Freq: 1
## 9 °--Tree 6 \n Count: 6
## 10     |--Char: l \n Freq: 3
## 11     °--Tree 4 \n Count: 4
## 12         |--Char: o \n Freq: 2
## 13         °--Tree 1 \n Count: 2
## 14             |--Char:  \n Freq: 1
## 15             °--Char: d \n Freq: 1
```

```
print>HelloTree)
```

```
##                               levelName
## 1 Tree 7 \n Count: 8
## 2 |--Tree 5 \n Count: 4
## 3 |   |--Tree 2 \n Count: 2
## 4 |   |   |--Char: e \n Freq: 1
## 5 |   |   °--Char: H \n Freq: 1
## 6 |   °--Tree 3 \n Count: 2
## 7 |       |--Char: r \n Freq: 1
## 8 |       °--Char: W \n Freq: 1
## 9 °--Tree 6 \n Count: 6
## 10     |--Char: l \n Freq: 3
## 11     °--Tree 4 \n Count: 4
## 12         |--Char: o \n Freq: 2
## 13         °--Tree 1 \n Count: 2
## 14             |--Char:  \n Freq: 1
## 15             °--Char: d \n Freq: 1
```

Ahora definimos la función *getCoding* recibe un árbol binario obtenido con la función *Huffman* y da como salida una tabla que contiene los caracteres, su frecuencia, el código, el número de bits por un caracter y el total de bits usados para ese caracter en todo el texto.

```
getCoding <- function(Tree){
  # Se genera una lista de los árboles después de recorrerlos a profundidad
  # con preorder
  TravList <- Traverse(Tree,"pre-order")
  # N es el número de subárboles
  N <- length(TravList)
  # Se definen arreglos char, freq y coding que van a
  # almacenar los caracteres, la frecuencia el código del caracter
  char <- c()
  freq <- c()
  coding <- c()
  # code se utiliza para guardar el código.
  code <- ""
  # Se recorre la lista de subárboles, si no se está en una hoja
  # se agrega un cero,
```

```

# Si se está en una hoja, se busca el último cero del code y se cambia por un
# uno y se sigue recorriendo la lista.
for (i in 1:N) {
  if(TravList[[i]]$isLeaf){
    char <- c(char, strsplit(TravList[[i]]$name, " ")[[1]][2])
    if(length(char)=="){
      freq <- c(freq, strsplit(TravList[[i]]$name, " ")[[1]][6])
    }else{
      freq <- c(freq, strsplit(TravList[[i]]$name, " ")[[1]][5])
    }
    coding <- c(coding, code)
    if(any(strsplit(code, "")[[1]]=="0")){
      aux <- max(which(strsplit(code, "")[[1]]=="0"))
      code <- paste0(substr(code, 1, aux-1), "1")
    }
  } else{
    code <- paste0(code, "0")
  }
}

# Se acomoda la información en una tabla y se ordena
# por la frecuencia de los caracteres
CodingList <- data.frame(cbind("character"=char, "frequency"=freq,
                                coding),
                          stringsAsFactors=FALSE)
CodingList$frequency <- as.numeric(CodingList$frequency)
CodingList$bits <- nchar(CodingList$coding)
CodingList$Nbits <- CodingList$frequency*CodingList$bits
CodingList <- CodingList[order(CodingList$frequency, decreasing = TRUE),]
return(CodingList)
}

```

Para el ejemplo de “Hello World” utilizamos el código anterior para obtener la codificación de los caracteres.

```

tableHello <- getCoding(HelloTree)
tableHello

```

##	character	frequency	coding	bits	Nbits
## 5	l	3	10	2	6
## 6	o	2	110	3	6
## 1	e	1	000	3	3
## 2	H	1	001	3	3
## 3	r	1	010	3	3
## 4	W	1	011	3	3
## 7		1	1110	4	4
## 8	d	1	1111	4	4

Para este ejemplo el número de bits requeridos es 32. Con una codificación fija se ocuparían 4 bits por caracter, por lo que necesitaríamos 44 bits.

Ejemplo

Como ejemplo utilizamos el discurso de Obama de 2016, usamos este ejemplo ya que se ha utilizado en ejemplos de procesamiento de texto <https://programminghistorian.org/en/lessons/basic-text-processing-in-r#>

a-small-example.

Obtenemos el texto

```
base_url <- "https://programminghistorian.org/assets/basic-text-processing-in-r"
url <- sprintf("%s/sotu_text/236.txt", base_url)
text <- paste(readLines(url), collapse = "\n")
```

Generamos el árbol

```
ObamaTree <- Huffman(text)
print(ObamaTree)
```

```
## levelName
## 1 Tree 74 \n Count: 28170
## 2 |--Tree 72 \n Count: 13692
## 3 |   |--Tree 68 \n Count: 6796
## 4 |     |--Char: e \n Freq: 3398
## 5 |     °--Tree 62 \n Count: 3376
## 6 |       |--Tree 57 \n Count: 1584
## 7 |       |   |--Char: c \n Freq: 792
## 8 |       |   °--Char: d \n Freq: 896
## 9 |       °--Char: s \n Freq: 1760
## 10 | °--Tree 69 \n Count: 7060
## 11 |   |--Tree 63 \n Count: 3522
## 12 |     |--Char: i \n Freq: 1761
## 13 |     °--Char: r \n Freq: 1769
## 14 |   °--Tree 64 \n Count: 3678
## 15 |     |--Char: n \n Freq: 1839
## 16 |     °--Tree 58 \n Count: 1806
## 17 |       |--Tree 51 \n Count: 840
## 18 |         |--Tree 46 \n Count: 398
## 19 |           |--Tree 41 \n Count: 190
## 20 |             |--Tree 35 \n Count: 92
## 21 |               |--Tree 27 \n Count: 44
## 22 |                 |--Tree 17 \n Count: 22
## 23 |                   |--Char: [ \n Freq: 11
## 24 |                     °--Char: ] \n Freq: 11
## 25 |                       °--Tree 18 \n Count: 24
## 26 |                         |--Char: 1 \n Freq: 12
## 27 |                         °--Tree 9 \n Count: 12
## 28 |                           |--Char: D \n Freq: 6
## 29 |                           °--Tree 4 \n Count: 6
## 30 |                             |--Char: 4 \n Freq: 3
## 31 |                             °--Char: 6 \n Freq: 3
## 32 |                               °--Char: ; \n Freq: 49
## 33 |                               °--Tree 36 \n Count: 102
## 34 |                                 |--Tree 28 \n Count: 50
## 35 |                                   |--Char: 0 \n Freq: 25
## 36 |                                   °--Tree 19 \n Count: 26
## 37 |                                     |--Char: M \n Freq: 13
## 38 |                                     °--Char: 0 \n Freq: 13
```

```

## 39 | | | | °--Char: j \n Freq: 53
## 40 | | | | °--Char: k \n Freq: 221
## 41 | | | | °--Char: y \n Freq: 483
## 42 | | | | °--Char: l \n Freq: 967
## 43 | °--Tree 73 \n Count: 17854
## 44 | | °--Tree 70 \n Count: 8346
## 45 | | | °--Tree 65 \n Count: 4056
## 46 | | | | °--Char: a \n Freq: 2028
## 47 | | | | °--Tree 59 \n Count: 2042
## 48 | | | | | °--Tree 52 \n Count: 980
## 49 | | | | | | °--Char: f \n Freq: 490
## 50 | | | | | | °--Char: p \n Freq: 531
## 51 | | | | | °--Tree 53 \n Count: 1080
## 52 | | | | | | °--Tree 47 \n Count: 504
## 53 | | | | | | | °--Tree 42 \n Count: 226
## 54 | | | | | | | | °--Tree 37 \n Count: 106
## 55 | | | | | | | | | °--Char: W \n Freq: 53
## 56 | | | | | | | | | °--Tree 29 \n Count: 56
## 57 | | | | | | | | | | °--Tree 20 \n Count: 28
## 58 | | | | | | | | | | | °--Char: : \n Freq: 14
## 59 | | | | | | | | | | | °--Tree 10 \n Count: 12
## 60 | | | | | | | | | | | | °--Tree 5 \n Count: 6
## 61 | | | | | | | | | | | | | °--Char: 9 \n Freq: 3
## 62 | | | | | | | | | | | | | °--Char: K \n Freq: 3
## 63 | | | | | | | | | | | | | °--Char: " \n Freq: 8
## 64 | | | | | | | | | | | | | °--Tree 21 \n Count: 32
## 65 | | | | | | | | | | | | | | °--Tree 11 \n Count: 16
## 66 | | | | | | | | | | | | | | | °--Char: G \n Freq: 8
## 67 | | | | | | | | | | | | | | | °--Char: H \n Freq: 8
## 68 | | | | | | | | | | | | | | | °--Tree 12 \n Count: 16
## 69 | | | | | | | | | | | | | | | | °--Char: R \n Freq: 8
## 70 | | | | | | | | | | | | | | | | °--Char: V \n Freq: 8
## 71 | | | | | | | | | | | | | | | | °--Char: \n \n Freq: 139
## 72 | | | | | | | | | | | | | | | | °--Tree 43 \n Count: 280
## 73 | | | | | | | | | | | | | | | | | °--Tree 38 \n Count: 138
## 74 | | | | | | | | | | | | | | | | | | °--Tree 30 \n Count: 68
## 75 | | | | | | | | | | | | | | | | | | | °--Tree 22 \n Count: 32
## 76 | | | | | | | | | | | | | | | | | | | | °--Tree 13 \n Count: 16
## 77 | | | | | | | | | | | | | | | | | | | | | °--Tree 6 \n Count: 8
## 78 | | | | | | | | | | | | | | | | | | | | | | °--Char: J \n Freq: 4
## 79 | | | | | | | | | | | | | | | | | | | | | | °--Char: Q \n Freq: 4
## 80 | | | | | | | | | | | | | | | | | | | | | | °--Tree 7 \n Count: 8
## 81 | | | | | | | | | | | | | | | | | | | | | | | °--Tree 2 \n Count: 4
## 82 | | | | | | | | | | | | | | | | | | | | | | | | °--Char: ! \n Freq: 2
## 83 | | | | | | | | | | | | | | | | | | | | | | | | °--Char: 5 \n Freq: 2
## 84 | | | | | | | | | | | | | | | | | | | | | | | | °--Tree 3 \n Count: 4
## 85 | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: 8 \n Freq: 2
## 86 | | | | | | | | | | | | | | | | | | | | | | | | | °--Tree 1 \n Count: 2
## 87 | | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: / \n Freq: 1
## 88 | | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: 3 \n Freq: 1
## 89 | | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: N \n Freq: 18
## 90 | | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: & \n Freq: 35
## 91 | | | | | | | | | | | | | | | | | | | | | | | | | | °--Tree 31 \n Count: 70
## 92 | | | | | | | | | | | | | | | | | | | | | | | | | | | °--Char: B \n Freq: 35

```

```

## 93      |      |      |      |      °--Char: L \n Freq: 36
## 94      |      |      |      |      °--Char: A \n Freq: 148
## 95      |      |      |      |      °--Char: g \n Freq: 584
## 96      |      |      |      |      °--Tree 66 \n Count: 4446
## 97      |      |      |      |      |--Char: o \n Freq: 2223
## 98      |      |      |      |      °--Tree 60 \n Count: 2444
## 99      |      |      |      |      |--Tree 54 \n Count: 1216
## 100     |      |      |      |      |      °--... 2 nodes w/ 12 sub
## 101     |      |      |      |      |      °--... 1 nodes w/ 18 sub
## 102     |      |      |      |      °--... 1 nodes w/ 49 sub

```

Obtenemos los códigos correspondientes

```

tableObama <- getCoding(ObamaTree)
tableObama

```

##	character	frequency	coding bits	Nbits
## 75		6024	111	3 18072
## 1	e	3398	000	3 10194
## 60	t	2666	1100	4 10664
## 48	o	2223	1010	4 8892
## 22	a	2028	1000	4 8112
## 7	n	1839	0110	4 7356
## 6	r	1769	0101	4 7076
## 5	i	1761	0100	4 7044
## 4	s	1760	0011	4 7040
## 61	h	1336	11010	5 6680
## 21	l	967	01111	5 4835
## 3	d	896	00101	5 4480
## 2	c	792	00100	5 3960
## 74	u	778	110111	6 4668
## 57	m	641	101110	6 3846
## 56	w	614	101101	6 3684
## 47	g	584	100111	6 3504
## 24	p	531	100101	6 3186
## 23	f	490	100100	6 2940
## 20	y	483	011101	6 2898
## 62	.	350	1101100	7 2450
## 59	b	343	1011111	7 2401
## 58	,	325	1011110	7 2275
## 55	v	310	1011001	7 2170
## 19	k	221	0111001	7 1547
## 73	'	193	11011011	8 1544
## 54	I	150	10110001	8 1200
## 46	A	148	10011011	8 1184
## 34	\n	139	10011001	8 1112
## 49	T	72	101100000	9 648
## 18	j	53	011100011	9 477
## 25	W	53	100110000	9 477
## 14	;	49	011100001	9 441
## 68	S	42	1101101001	10 420
## 50	x	38	1011000010	10 380
## 45	L	36	1001101011	10 360

## 43	&	35	1001101001	10	350
## 44	B	35	1001101010	10	350
## 15	0	25	0111000100	10	250
## 69	-	22	11011010100	11	242
## 70	C	22	11011010101	11	242
## 71	q	22	11011010110	11	242
## 72	z	22	11011010111	11	242
## 51	P	19	10110000110	11	209
## 42	N	18	10011010001	11	198
## 26	:	14	10011000100	11	154
## 16	M	13	01110001010	11	143
## 17	O	13	01110001011	11	143
## 10	1	12	01110000010	11	132
## 8	[11	01110000000	11	121
## 9]	11	01110000001	11	121
## 67	?	11	110110100011	12	132
## 53	2	10	101100001111	12	120
## 63	F	10	110110100000	12	120
## 64	U	10	110110100001	12	120
## 52	E	9	101100001110	12	108
## 29	"	8	100110001011	12	96
## 30	G	8	100110001100	12	96
## 31	H	8	100110001101	12	96
## 32	R	8	100110001110	12	96
## 33	V	8	100110001111	12	96
## 11	D	6	011100000110	12	72
## 65	7	5	1101101000100	13	65
## 66	Y	5	1101101000101	13	65
## 35	J	4	1001101000000	13	52
## 36	Q	4	1001101000001	13	52
## 12	4	3	0111000001110	13	39
## 13	6	3	0111000001111	13	39
## 27	9	3	1001100010100	13	39
## 28	K	3	1001100010101	13	39
## 37	!	2	10011010000100	14	28
## 38	5	2	10011010000101	14	28
## 39	8	2	10011010000110	14	28
## 40	/	1	100110100001110	15	15
## 41	3	1	100110100001111	15	15

El número de bits requeridos es $1.5301e+05$. Con una codificación fija se ocuparían 8 bits por caracter, por lo que necesitaríamos $2.4961e+05$ bits. Por lo que el tamaño disminuye en un 38.70%.