Huffman Coding

CMPUT 414
Winter 2019

Compression

-Overview

-Entropy

-Huffman Coding

Media types

What media types do you know?

Data (Text)

Audio

Image

Video

Graphics

Media types compression

Compression

- Data → Cannot afford to lose information
- Audio / image / video → Can afford to lose information
- Graphics → maybe

Some compression methods

-RLE, Huffman encoding etc. for LOSSLESS

-Sub-band coding, CELP for Audio

-JPEG, Wavelet, fractal for Images

-H.263, MPEG-1, MPEG-2 for video

-MPEG-4, Emerging standard considering all media types, including graphics.

What is the goal of compression?

Goal of compression is to reduce redundancies in information!

Which redundancies?

Coding redundancies

Spatio/ temporal redundancy

Perceptual redundancy (Human)

Redundancies

Coding redundancies

- Reducing coding redundancies is completely lossless

Spatio/ temporal redundancy

- Reducing spatio / temporal redundancies can be lossy

Perceptual redundancy (Human)

- Reducing perceptual redundancies is lossy

What kinds of events convey more information?

1) There was a car accident

2) It snowed in Edmonton on January 1

3) It snowed in Las Vegas on July 10

4) A Jumbo Jet crashed

What kinds of events convey more information?

1) There was a car accident
2) It snowed in Edmonton on January 1
3) It snowed in Las Vegas on July 10
4) A Jumbo Jet crashed

Event 3 contains the most because it's the rarest event

Event 4 eventful, but not as rare

Event 2 not that important, because it happens so often

Event 1 is almost not worth mentioning

Simple Coding Example

How the string "**go go gophers**" is encoded in ASCII, how we might save bits using a simpler coding scheme?

With an ASCII encoding (8 bits per character) the 13 character string "go go gophers" requires 104 bits.

The table below shows how the coding works.

coding a message

ASCII coding

char	ASCII	binary
g	103	1100111
o	111	1101111
p	112	1110000
h	104	1101000
e	101	1100101
r	114	1110010
s	115	1110011
space	32	1000000

3-bit coding

char	code	binary
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101
s	6	110
space	7	111

The string "go go gophers" would be written (coded numerically) as **103 111 32 103 111 32 103 111 112 104 101 114 115.**

coding a message

ASCII coding

char	ASCII	binary
g	103	1100111
o	111	1101111
p	112	1110000
h	104	1101000
e	101	1100101
r	114	1110010
s	115	1110011
space	32	1000000

3-bit coding

char	code	binary
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101
s	6	110
space	7	111

Although not easily readable by humans, this would be written as the following stream of bits (the spaces would not be written, just the 0's and 1's)

1100111 1101111 1100000 1100111 1101111 1000000 1100111 1101111 1110000 1101000 1100101 1110010 1110011

Since there are only eight different characters in "go go gophers", it's possible to use only 3 bits to encode the different characters. We might, for example, use the encoding, though other 3-bit encodings are possible.

Now the string "go go gophers" would be encoded as

0 1 7 0 1 7 0 1 2 3 4 5 6

or, as bits:

000 001 111 000 001 111 000 001 010 011 100 101 110 111

By using three bits per character, the string "go go gophers" uses a total of **39** bits instead of **104** bits.

More bits can be saved if we use fewer than three bits to encode characters like g, o, and space that occur frequently and more than three bits to encode characters like e, p, h, r, and s that occur less frequently in "go go gophers".

This is the basic idea behind **Huffman coding**: to use fewer bits for more frequently occurring characters.

Entropy

Let $\boldsymbol{P(E)} = Probability\ of\ an\ Event\ E\ ocurring$

Info content in $\boldsymbol{E}$ is proportional to $\frac{\mathbf{1}}{\boldsymbol{P(E)}}$

Supposed **b** digits are used to code an Event or Symbol. Then, information content $\boldsymbol{log_b \frac{1}{P(E)}}$

Consider n events $E_1, E_2, E_3, \ldots, E_n$

Entropy

The average information content of a ser of events (or symbol) equal to

$$\sum(Probability\ of\ Event\ E_i\ ocurring)x(Information\ content\ of\ E_i)$$

The entropy ***H*** (in bits) is the weighted sum, across all symbols with non-zero probability $P(E_i)$, of the information content of each symbol:

Where $0 \leq P(E_i) \leq 1\ and\ \sum P(E_i) = 1$

$$\boldsymbol{H = \sum P(E_i)\, log_b P(E_i)}$$ b is base used for coding (e.g., 2 for binary)

Entropy

When $P(E_i) = \frac{1}{N}\ for\ i = 1,2,3, \dots, N$

The $\boldsymbol{H}$ reaches its máximum value

$\boldsymbol{H_{max} = log_b N}$ where b is what base (i.e. 2 for binary).

Example

If base = 2 (i.e., Binary)

$\boldsymbol{H_{max} = log_2 N}$

Huffman Coding

Huffman coding is a lossless data compression algorithm. The procedure is based on two observations regarding optimum prefix codes:

1. In an optimum code, symbols that occur more frequently (have a higher probability of occurrence) will have shorter codewords than symbols that occur less frequently.
2. In an optimum code, the two symbols that occur least frequently will have the same length

Huffman coding

The probability of a symbol (X) is nothing but the proportion of time X appears in the string of symbols to be coded. (P.344 in Image Processing text)

Example
6 symbols: A B C D E F P(A) = 0.4 P(B) = 0.3 P(C) = 0.1 P(D) = 0.1 P(E) = 0.06
P (F) = 0.04

Using fixed length codes, 000 = A, 001 = B, …101 = F (Length = 3)

H= entropy = $\sum \boldsymbol{P(E_i)}\, \boldsymbol{log_b P(E_i)}$ = 2.14

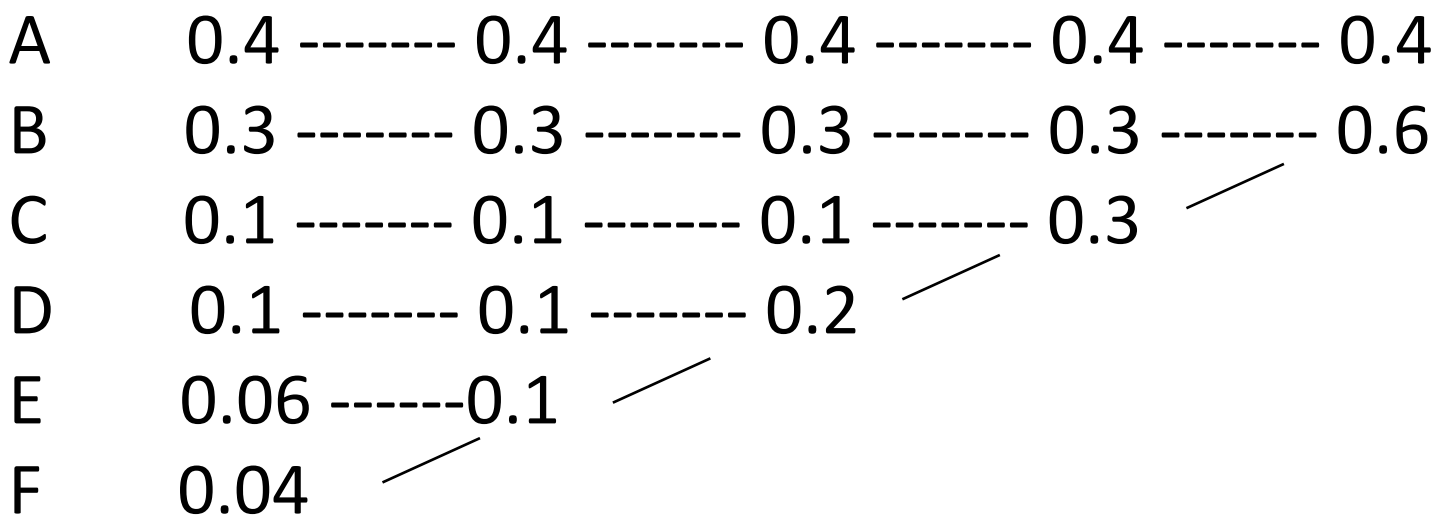
Huffman coding

For variable length codes, we use Huffman method:

Step 1 Source Reduction

-list symbols in order from largest to smallest probability

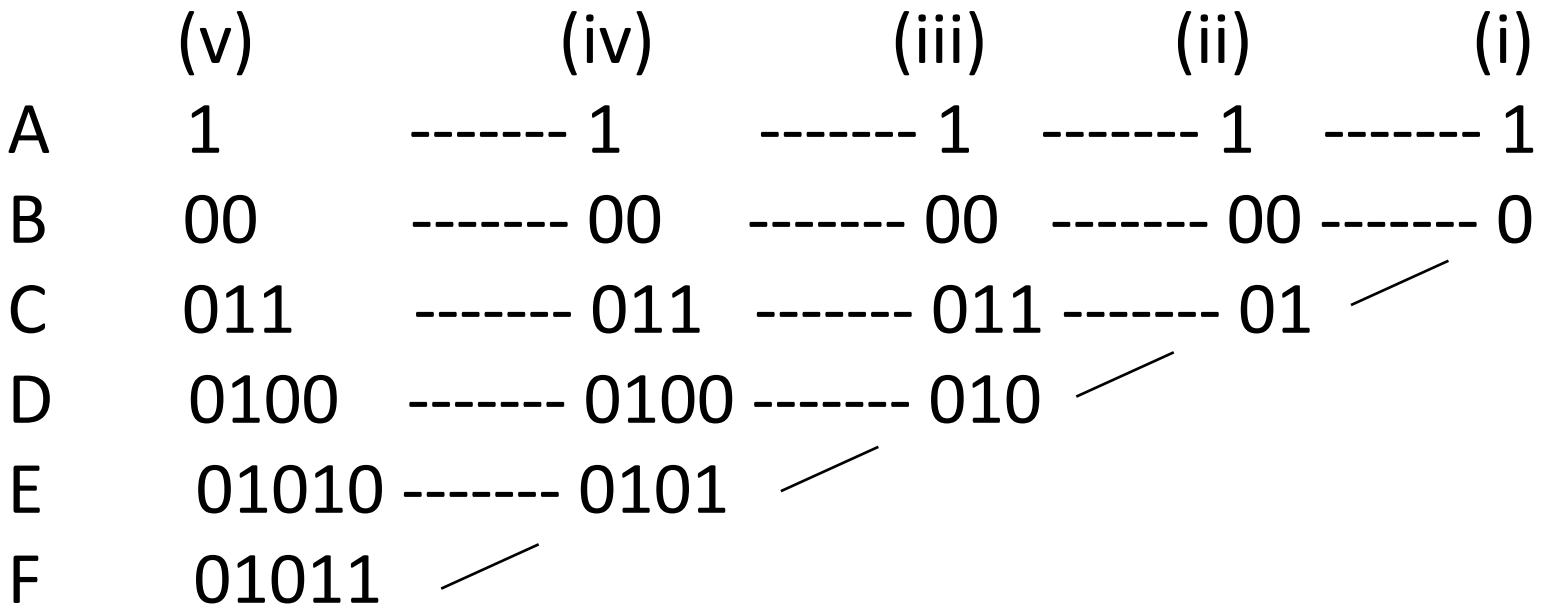
-we then combine successive two small + probability symbols, until 2 left



Huffman coding

Step 2 Code Generation

In this step we work backwards, assigning variable length codes:



Thus final Huffman codes will be:

A = 1

B = 00

C = 011

D = 0100

E = 01010

F = 01011

Average Huffman Code length = 2.2 bits/symbol

$$\overbrace{0.4 \times (1)}^{A} + \overbrace{0.3 \times (2)}^{B} + \overbrace{0.1 \times (3)}^{C} + \overbrace{0.1 \times (4)}^{D} + \overbrace{0.06 \times (5)}^{E} + \overbrace{0.04 \times (5)}^{F}$$

Compared to entropy = 2.12 bits/symbol

$$-\left(\overbrace{0.4 \log_2 (0.4)}^{A} + \overbrace{0.3 \lg (0.3)}^{B} + \overbrace{0.1 \lg (.1)}^{C} + \overbrace{0.1 \lg (.1)}^{D} + \overbrace{0.06 \lg (.06)}^{E} + \overbrace{0.04 \lg (.04)}^{F} \right)$$

Huffman Coding

Note: codes are unique and identifiable without delimiters as individuals.

e.g.,

A = 1, and nothing else starts with 1

B = 00, and nothing else starts with 00

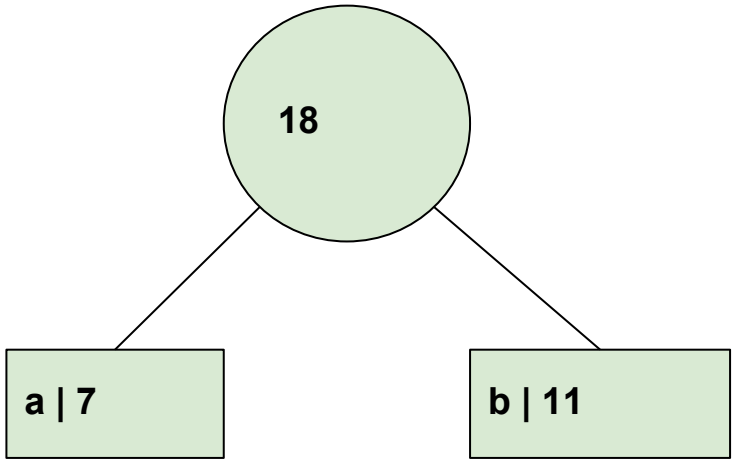
Steps to build Huffman Tree

First the input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

Character	Frequency
a	7
b	11
c	14
d	15
e	19
f	47

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

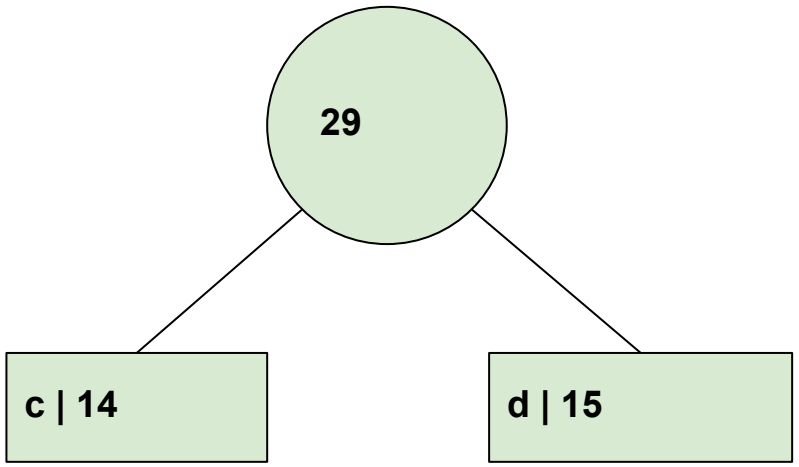
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 7 + 11 = 18.



Character	Frequency
a	7
b	11
c	14
d	15
e	19
f	47

Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 14 + 15 = 29

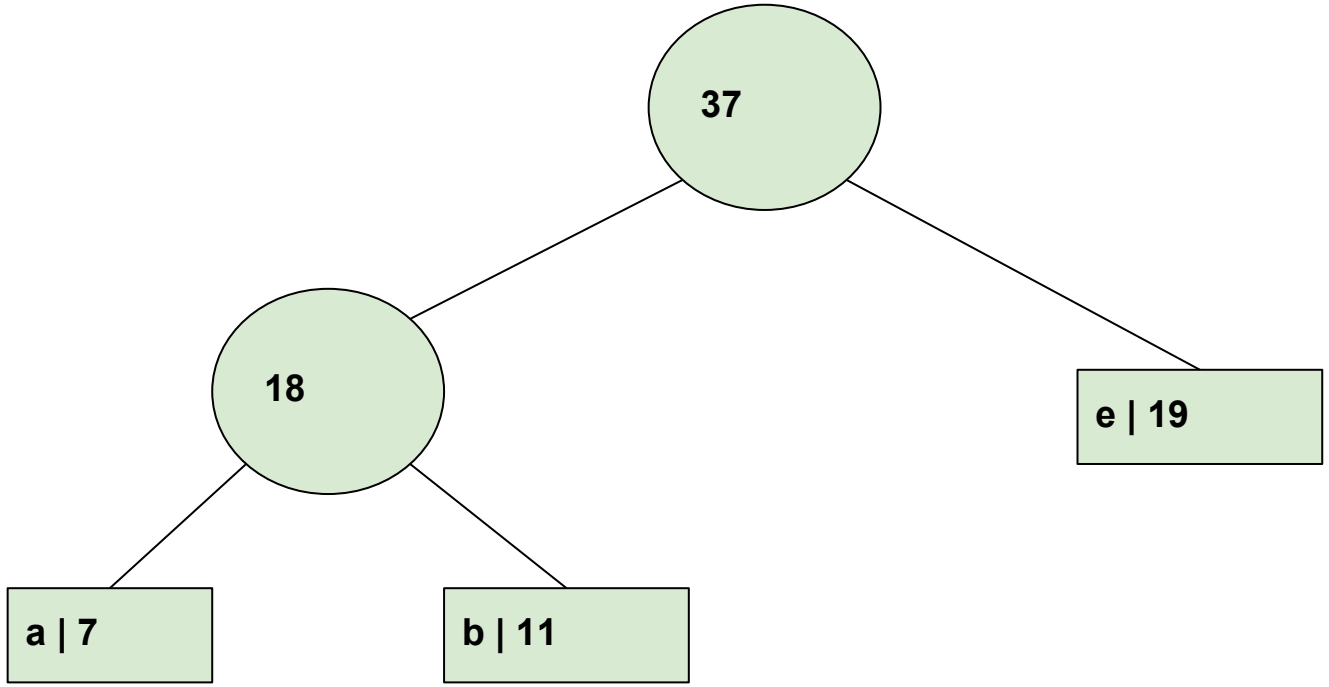


Character	Frequency
c	14
d	15
Internal Node	18
e	19
f	47

Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency

18 + 19 = 37

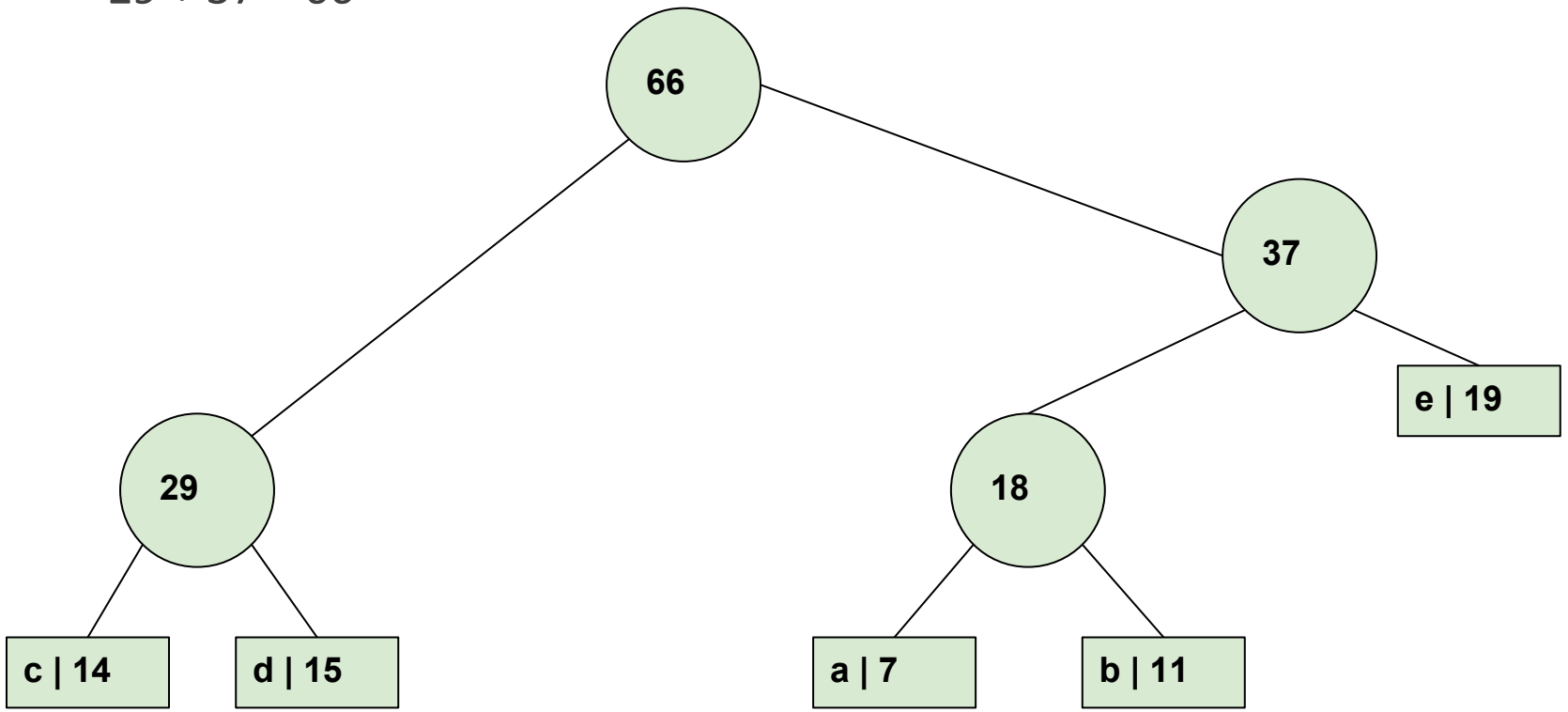


Character	Frequency
Internal Node	18
e	19
Internal Node	29
f	47

Now min heap contains 3 nodes.

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency

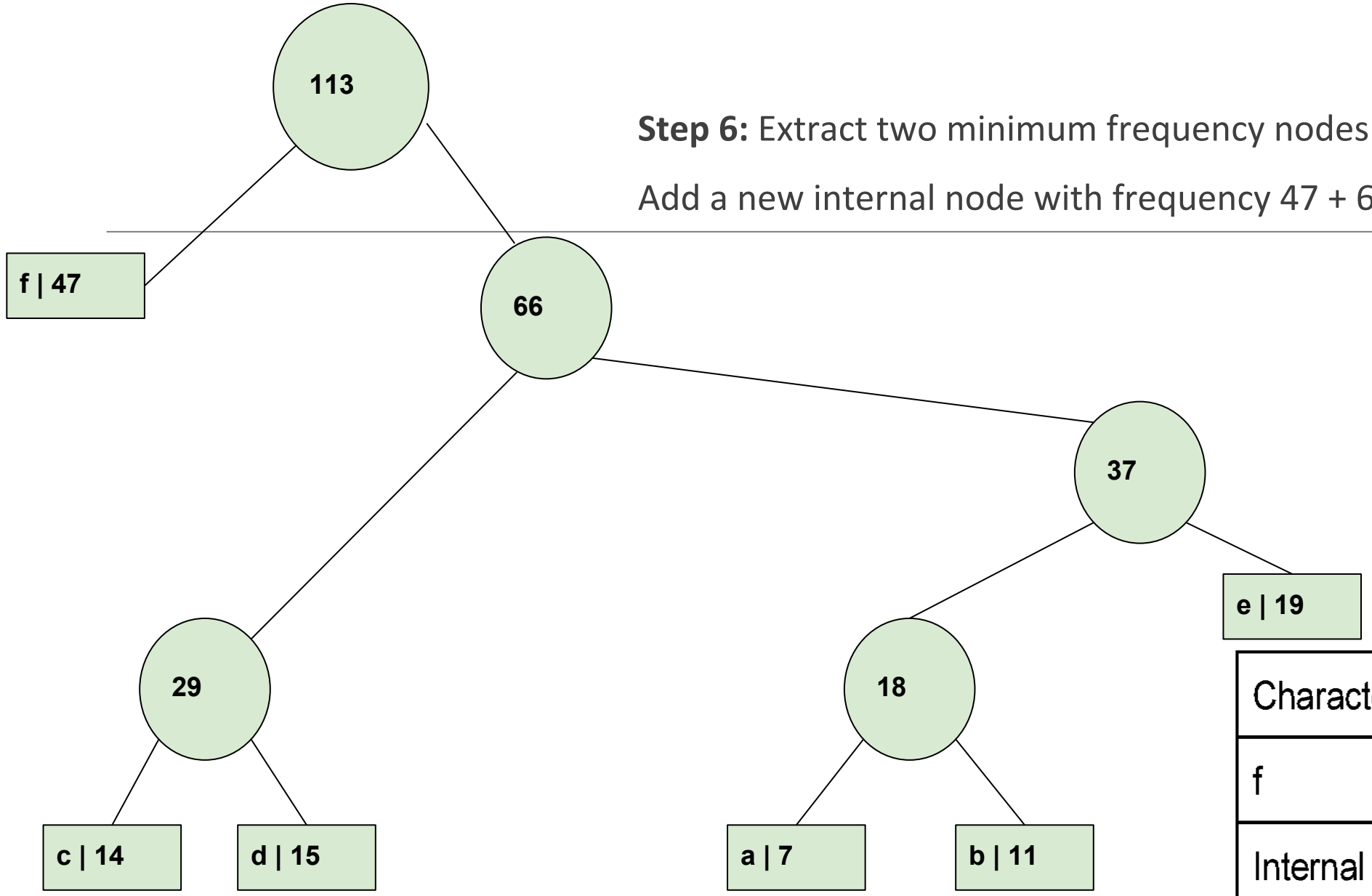
29 + 37 = 66



Character	Frequency
Internal Node	29
Internal Node	37
f	47

Step 6: Extract two minimum frequency nodes.

Add a new internal node with frequency 47 + 66 = 113



Character	Frequency
f	47
Internal Node	66

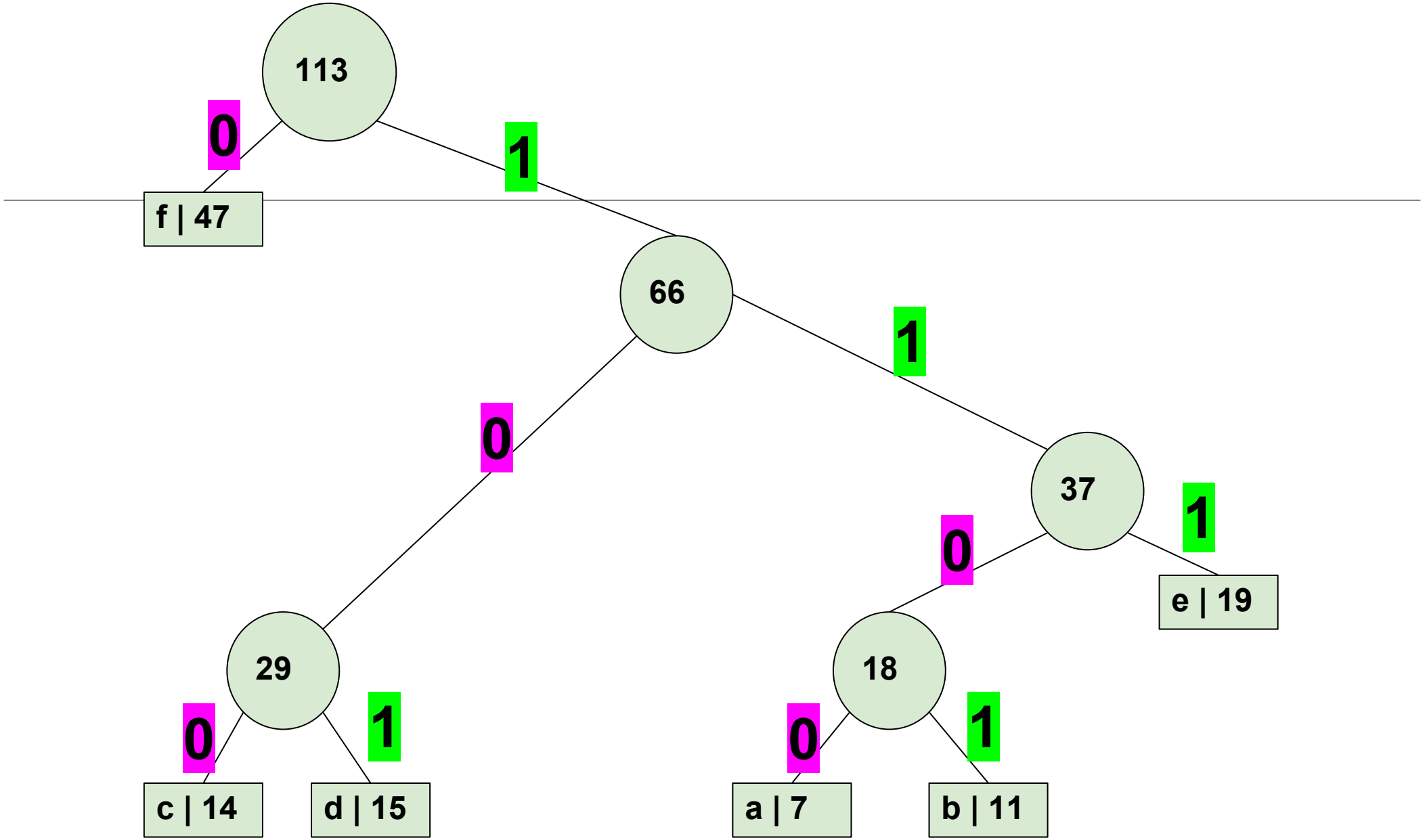
Now min heap contains only one node.

Since the heap contains only one node, the algorithm stops here.

Character	Frequency
Internal Node	113

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.


113
0
1
f | 47
66
1
0
37
0
1
e | 19
29
18
0
1
0
1
c | 14
d | 15
a | 7
b | 11

The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111