CS302 – Analysis and Design of Algorithms

Greedy Algorithms and Huffman Coding

Content

Content
Greedy Algorithms
Huffman Coding
Adaptive Huffman Coding
Exercise

Greedy Algorithms

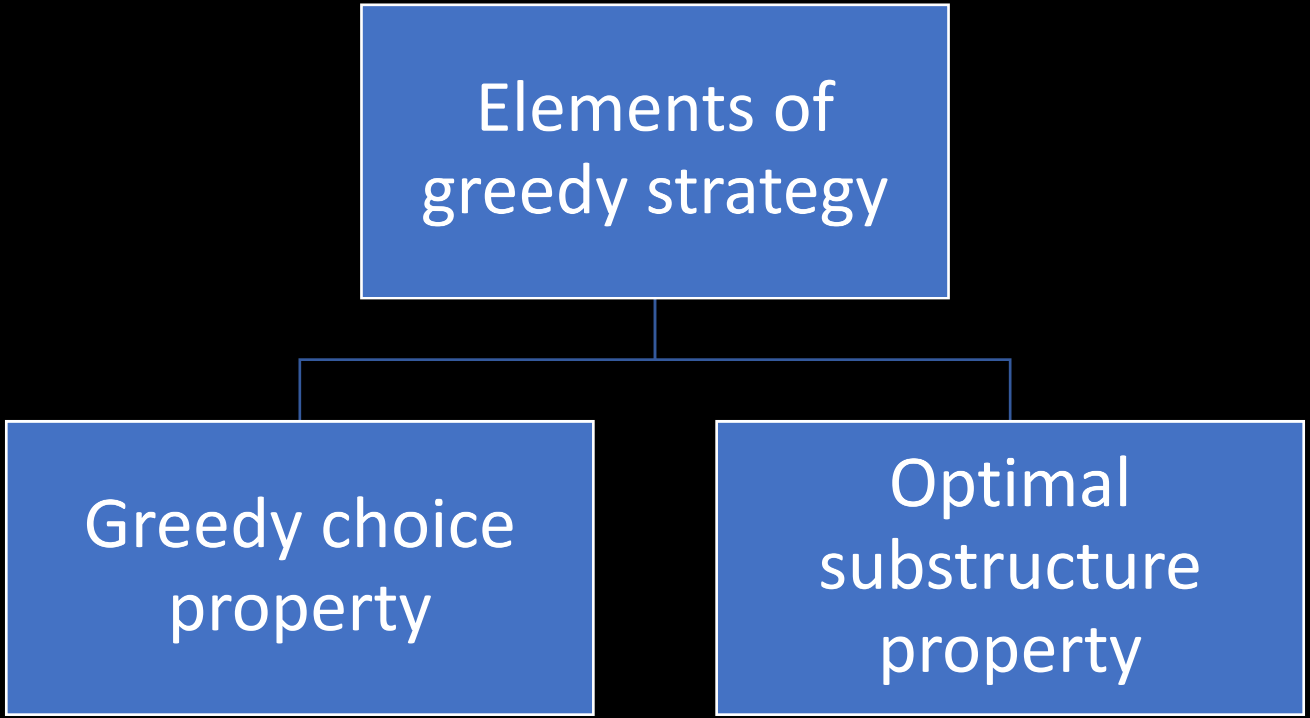
- A greedy algorithm always makes the choice that looks best at the moment.
  - It makes a locally optimal choice in the hope that this choice leads to a globally optimal solution.

- The optimal solution is not always guaranteed.

- How can you tell whether a greedy algorithm will solve a particular optimization problem?

Greedy Algorithms

- If an optimization problem has the following two properties, then it can be solved using a greedy approach.



Greedy Algorithms

- Greedy-choice property:
  - You have many choices to make.
  - Selecting the optimal choice at the current step (selecting local optimum)
  - Achieving the overall optimal solution for the whole problem (achieving global optimum)

- Optimal substructure property:
  - Given a problem that can be divided into subproblems.
  - Each subproblem has an optimal solution.

Greedy Algorithms

- **Activity selection problem:**

A set of activities that need to be completed. Each one has a start and finish time. The algorithm finds the maximum number of activities that can be done in a given time without them overlapping.

Greedy Algorithms

- Steps:
  1. Sort the activities ascending order using the finish time.
  2. Start by picking the first activity. Create a new list to store the selected activity.
  3. To choose the next activity, compare the finish time of the last activity to the start time of the next activity.
  4. If the start time of the next activity is greater than the finish time of the last activity, it can be selected. If not, skip this and check the next one.
  5. This process is repeated until all activities are checked.
  6. The final solution is a list containing the activities that can be done.

Greedy Algorithms

Activity	Start time	Finish time
Read book	2	5
Study	6	10
Eat lunch	4	8
Hangout	7	15
Watch TV	13	14
Shopping	10	12

Greedy Algorithms

- Sort by the finish time

Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15

Greedy Algorithms

- Select the first activity and store it in the selected activity list.

Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15

Greedy Algorithms

- Eat lunch starts at 4, it overlaps with Read book. So, skip it

Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15

Greedy Algorithms

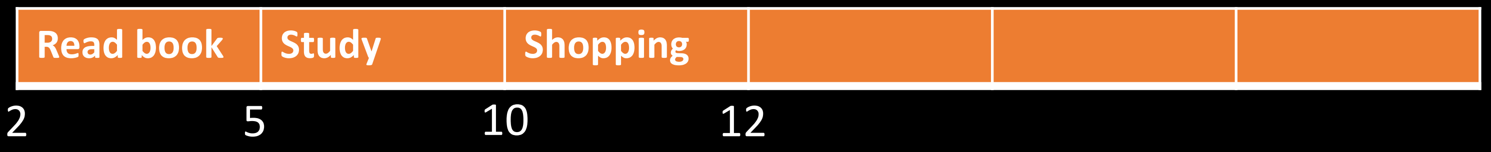
- Study starts at 6, it does not overlap with Read book

Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15

Greedy Algorithms

- Shopping starts at 10, study finished at 10. So, it does not overlap

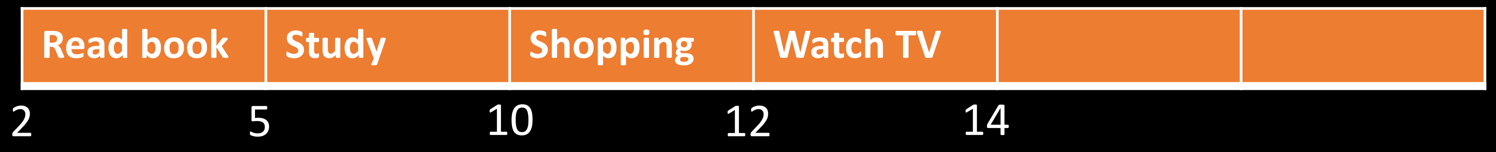
Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15



Greedy Algorithms

- Watch TV does not overlap with Shopping

Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15



Greedy Algorithms

- Hangout starts at 7, it overlaps with previous activities. So, skip it

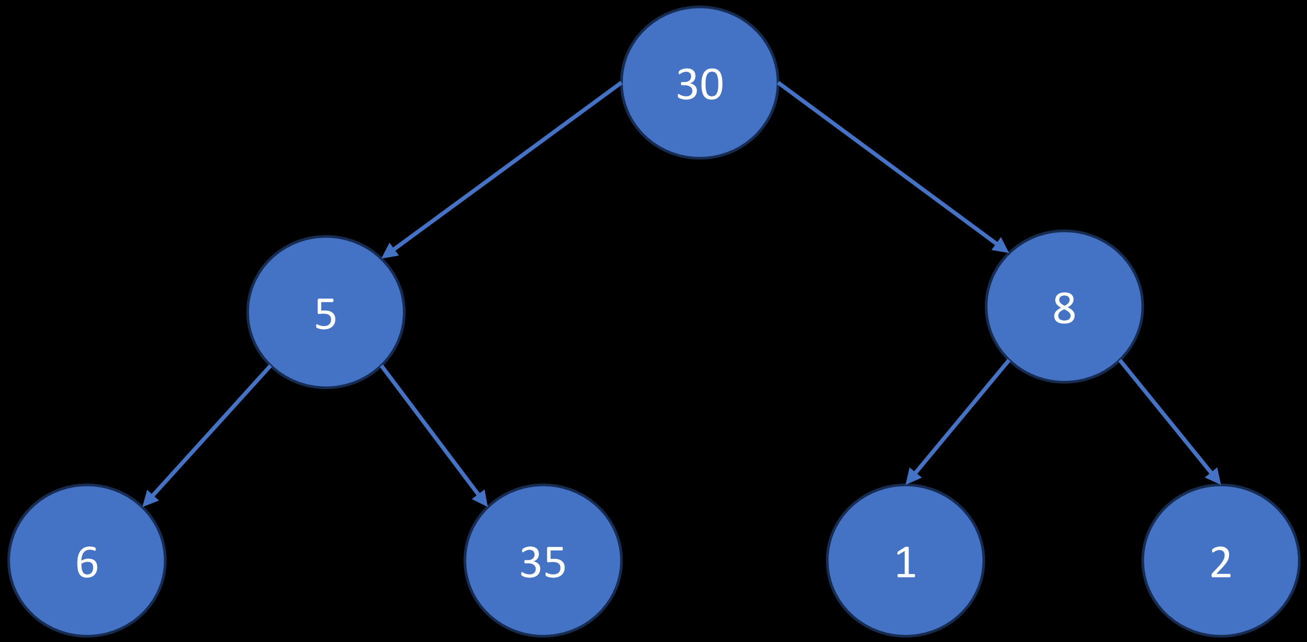
Activity	Start time	Finish time
Read book	2	5
Eat lunch	4	8
Study	6	10
Shopping	10	12
Watch TV	13	14
Hangout	7	15

Read book	Study	Shopping	Watch TV		

2 5 10 12 14

Greedy Algorithms

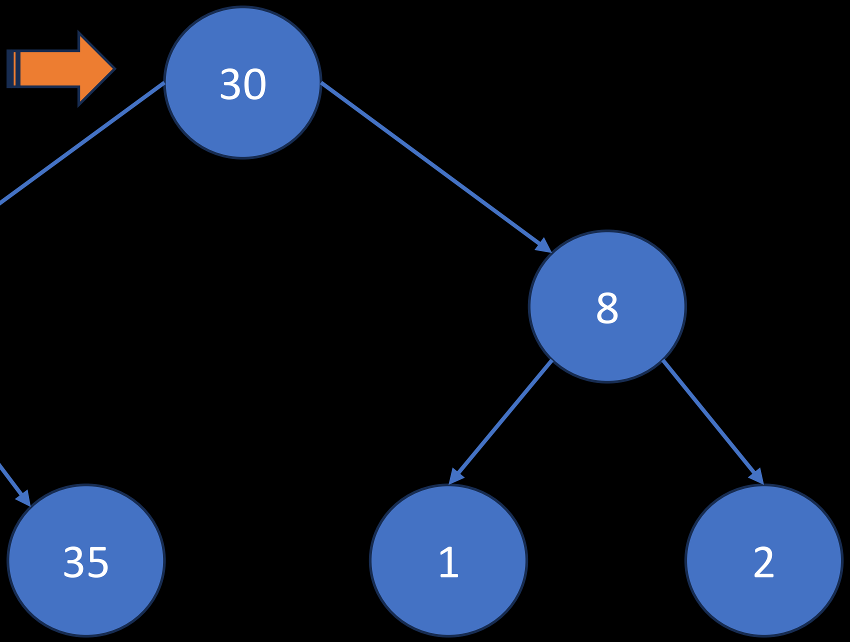
- Example: find the longest path in the graph below from root to leaf.



Greedy Algorithms

- The largest weight is 30

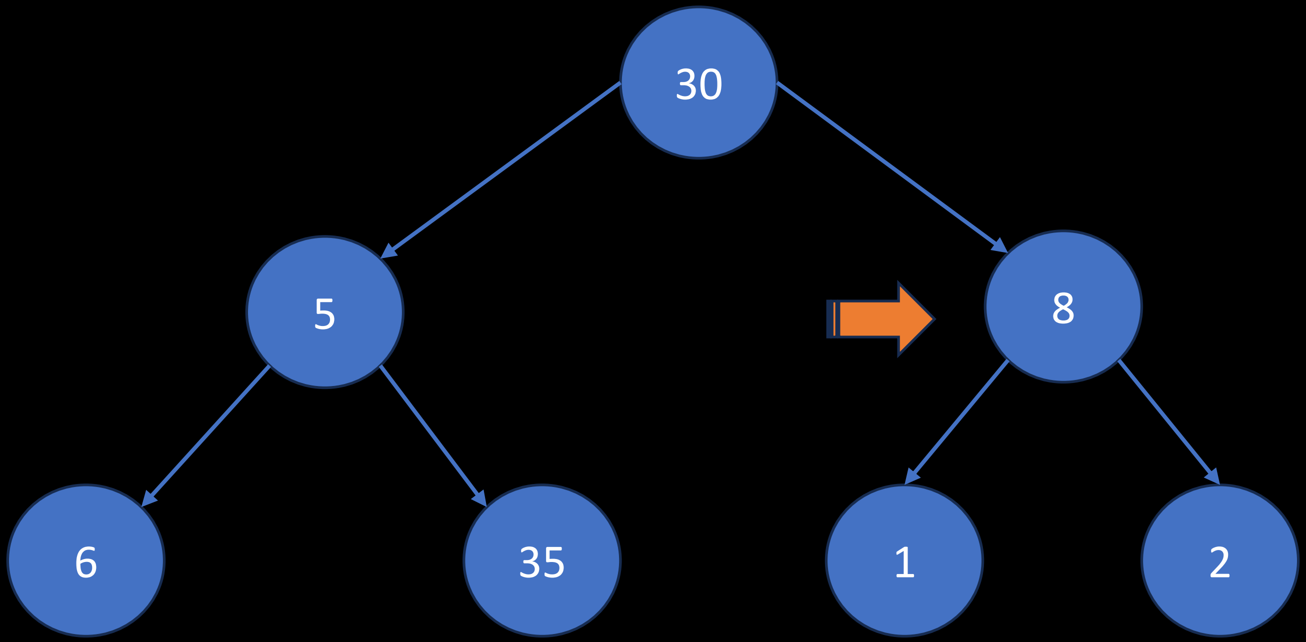
30



Greedy Algorithms

- 8 > 5

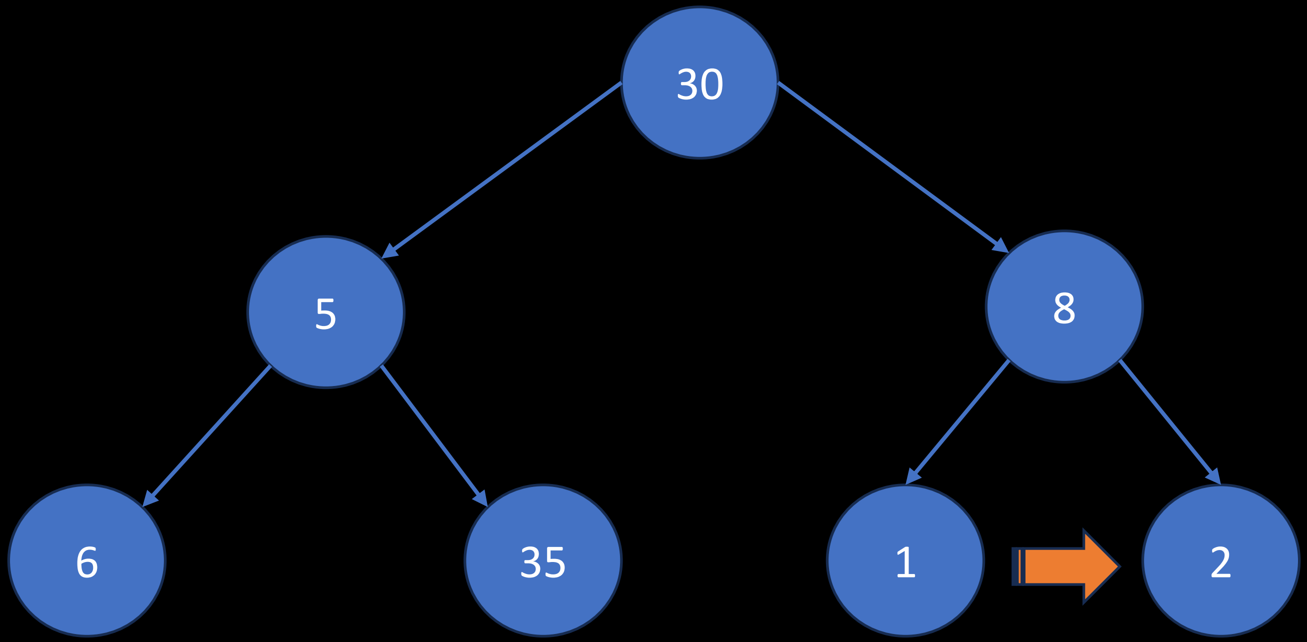
30 + 8



Greedy Algorithms

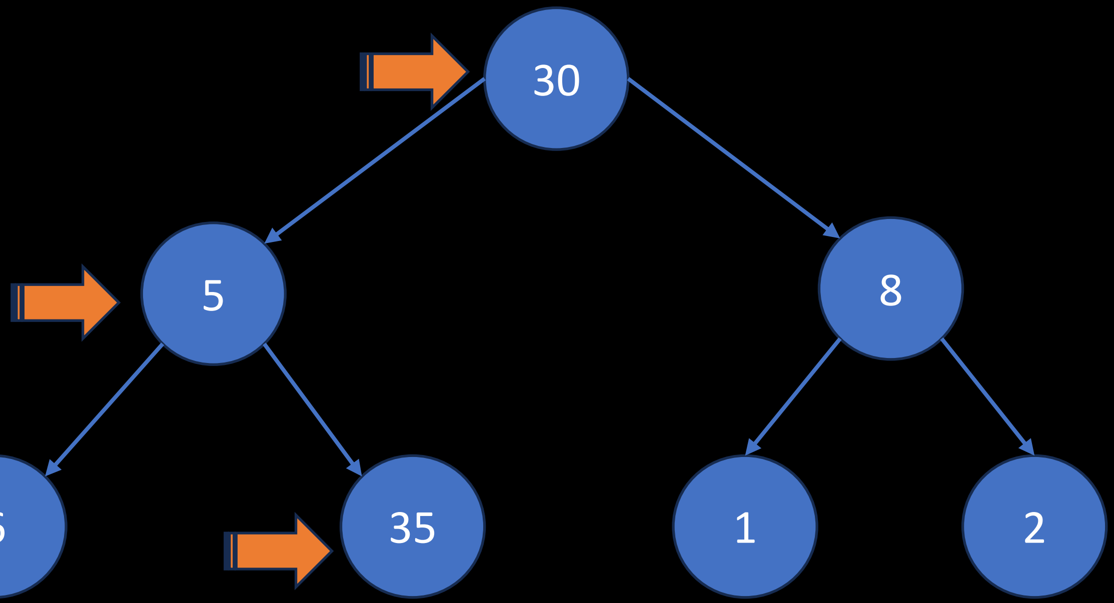
- 2 > 1

30 + 8 + 2 = 40



Greedy Algorithms

- However, this is not optimal!
  - The optimal solution is 30 + 5 + 35 = 70



- The greedy algorithm doesn't always produce the optimal solution.

Content

Content
Greedy Algorithms
Huffman Coding
Adaptive Huffman Coding
Exercise

Huffman Coding

- Why would we want to compress information?
  - To save time and space.

- What is the quality of the compressed information?
  - **Lossless compression:** the decompressed data is identical to the original data.
    - E.g., zip files and PNG pictures

  - **Lossy compression**: the decompressed information differs from the original, but ideally in an insignificant manner.
    - E.g., MP3 and JPEG

Huffman Coding

- Why is it possible to compress information?
    - Because digital data contains redundancy or useless bits.
- Each ASCII character requires 8 bits.
    - Ranges from 0 to 255
- But most used characters have a 0 as the leftmost bit!
    - Ranges from 0 to 127
- Therefore, one-eighth of the bits in ASCII text are useless.
    - Compress most ASCII text by 12.5%.

Huffman Coding

- Text compression must be lossless.

- Because a change in 1 bit in a given text, can have significant differences.

```
Don't forget the pop.  →  A soft drink
Don't forget the pot.  →  The marijuana
```

- Huffman algorithm is a lossless compression algorithm.

Huffman Coding

- DNA representation requires 4 characters: A, C, G, T.

- Representing this strand in ASCII requires 8n bits.
  - Assuming each ASCII value is 8 bits.

- We can do better: we have 4 characters, represent them using 2 bits only.
  - 00, 01, 10, 11 (fixed length codeword)
- We can do better: consider the relative frequency of each character.
  - A = 0, C = 100, G = 101, T = 11 (variable length codeword)

Huffman Coding

- Example: given the 20-char DNA strand TAATTAGAAATTCTATTATA,
  - A = 45%, C = 5%, G = 5%, T = 45%

- In ASCII, it will take $20 \times 8 = 160$ bits.

- In 2-bit encoding, it will take $20 \times 2 = 40$ bits.

- By considering the frequency, it will take $0.45 \times n \times 1 + 0.05 \times n \times 3 + 0.05 \times n \times 3 + 0.45 \times n \times 2 = 1.65n = 1.65 \times 20 =$ 33 bits

Huffman Coding

- The encoding of characters according to their frequency has two properties:
  - E.g., A = 0, C = 100, G = 101, T = 11.

  1. More frequent characters get shorter bit sequence (such as A and T)
  2. It is a prefix-free code. I.e., no code is a prefix of any other code.

- Prefix-free codes → no ambiguity when decompressing.

Huffman Coding

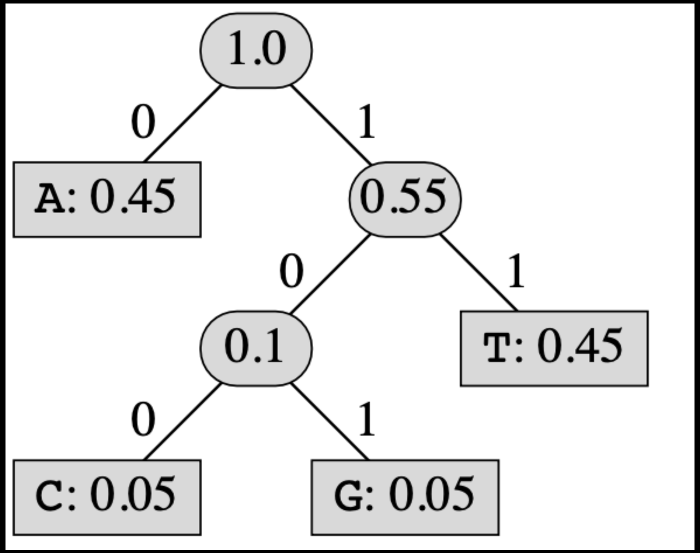
- Example: given the compressed data below using A = 0, C = 100, G = 101, T = 11, decompress it.

11 0 0 11 11 0 101 0 0 0 11 11 100 11 0 11 11 0 11 0

T A A T T A G A A A T T C T A T T A T A

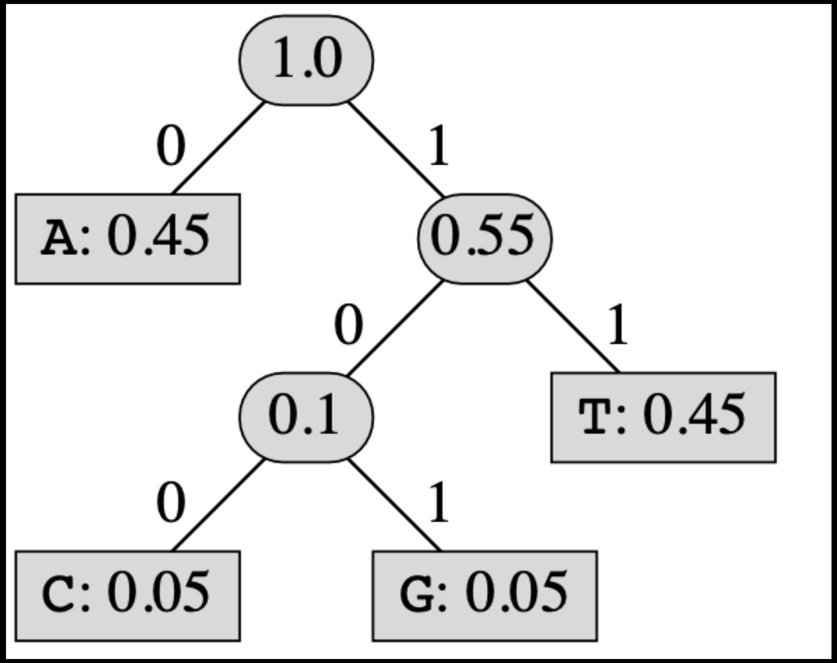
Huffman Coding

- Disadvantage: knowing the frequencies in advance. Thus, it needs 2 passes:
  - One to determine character frequencies,
  - one to map each character to its code.
- After determining the frequencies, Huffman algorithm builds a binary tree



Huffman Coding

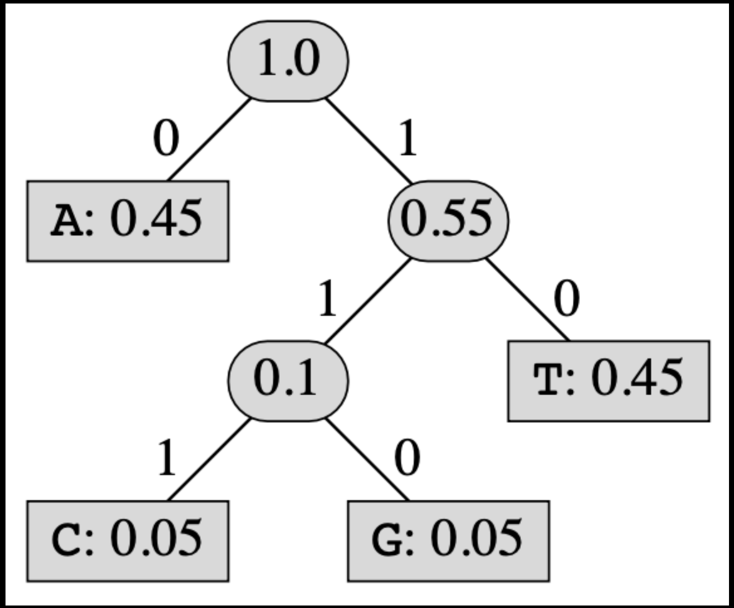
- Leaves represent characters with their frequencies.
- Internal nodes contain the sum of the frequencies in the leaves below it.
- Edge labeled with either a 0 or a 1.
- To determine the code for a character, follow the path from the root down to the leaf.
  - E.g., 100 → C



Huffman Coding

- The encoding of the labels does not matter as long as it's prefix-free.
- The next binary tree represents a valid prefix-free encoding.
    - The number of bits is the same as before.
- The number of bits for a character = the depth of the character's leaf

A = 0, C = 111, G = 110, T = 10



Huffman Coding

- Steps to build binary tree for Huffman coding:
    1. Compute the frequencies of each character.
    2. Create leaf nodes that represent each character.
    3. Find the two nodes with the lowest frequency.
    4. Assign a parent to those nodes with its value the sum of the frequencies of the children.
    5. Mark the left edge with 0 and the right edge with 1.
    6. Repeat steps 3-5 until we reach the root node with value 1.

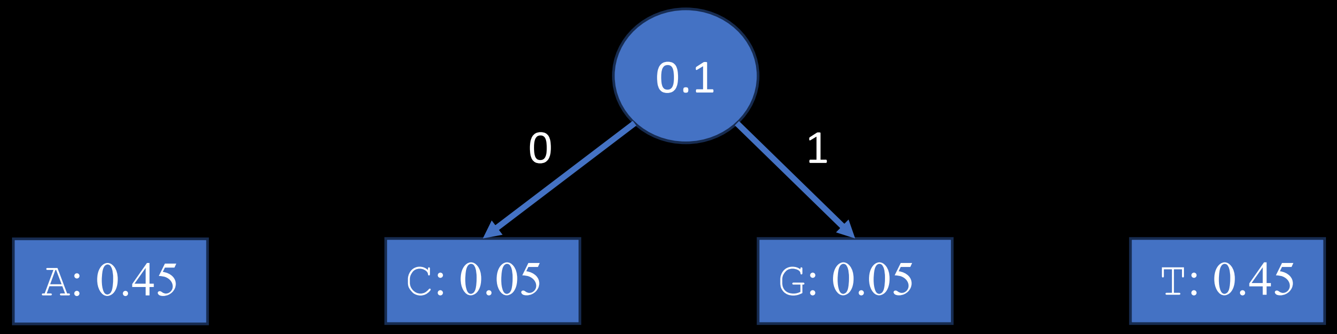
Huffman Coding

- DNA example:

A: 0.45 C: 0.05 G: 0.05 T: 0.45

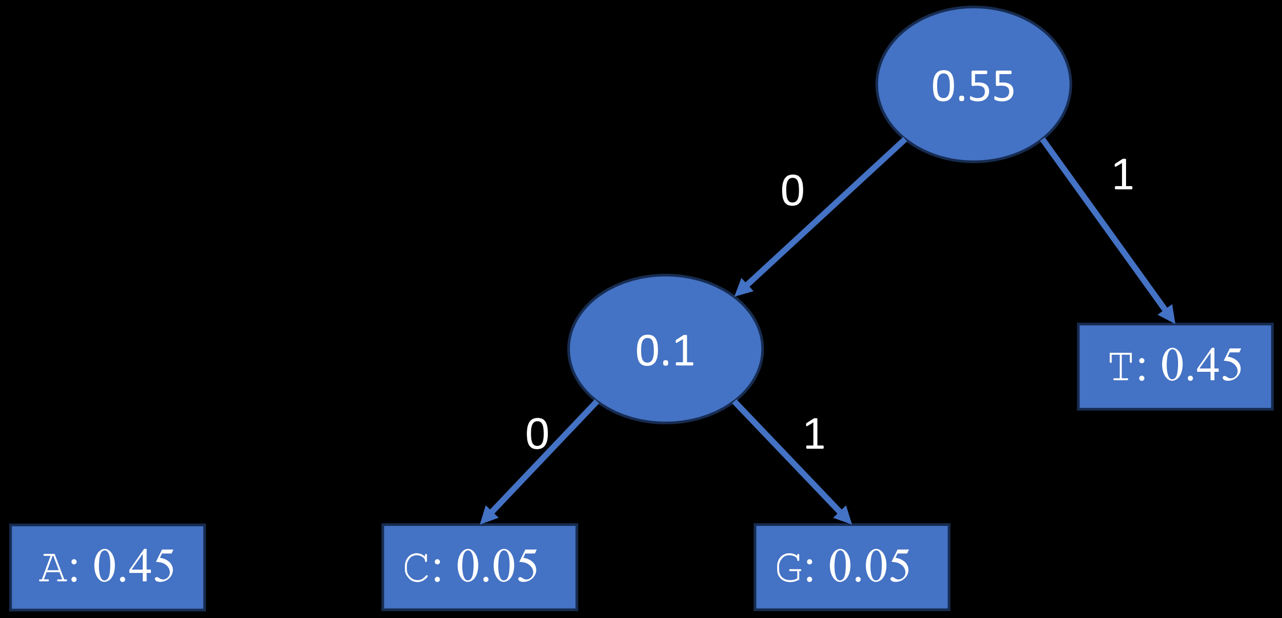
Huffman Coding

- The minimum frequencies are C and G



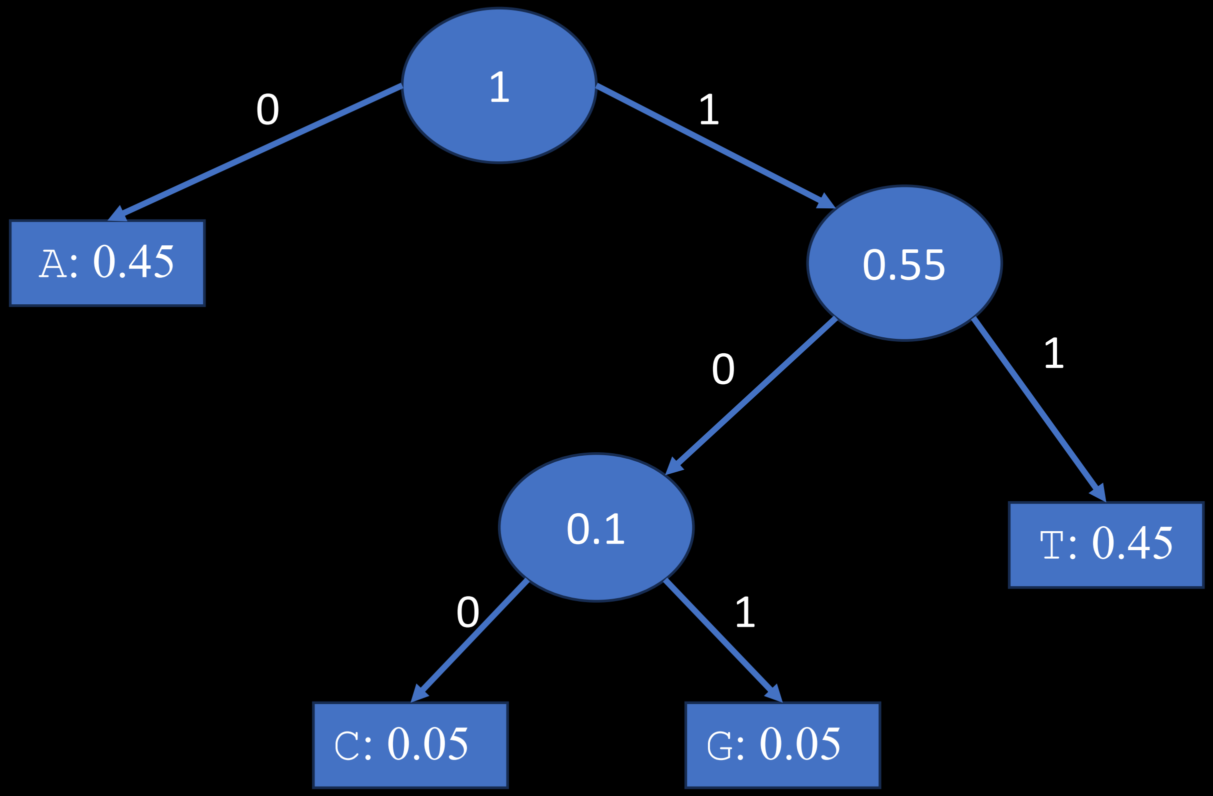
Huffman Coding

- The minimum frequencies are T and 0.1



Huffman Coding

- The minimum frequencies are A and 0.55



Huffman Coding

Procedure BUILD-HUFFMAN-TREE(*char*, *freq*, n)

Inputs:

- *char*: an array of n uncompressed characters.
- *freq*: an array of n character frequencies.
- n: the sizes of the *char* and *freq* arrays.

Output: The root of the binary tree constructed for Huffman codes.

1. Let Q be an empty priority queue.
2. For $i = 1$ to n:
   - A. Construct a new node z containing *char*$[i]$ and whose frequency is *freq*$[i]$.
   - B. Call INSERT(Q, z).
3. For $i = 1$ to $n - 1$:
   - A. Call EXTRACT-MIN(Q), and set x to the node extracted.
   - B. Call EXTRACT-MIN(Q), and set y to the node extracted.
   - C. Construct a new node z whose frequency is the sum of x's frequency and y's frequency.
   - D. Set z's left child to be x and z's right child to be y.
   - E. Call INSERT(Q, z).
4. Call EXTRACT-MIN(Q), and return the node extracted.

Huffman Coding

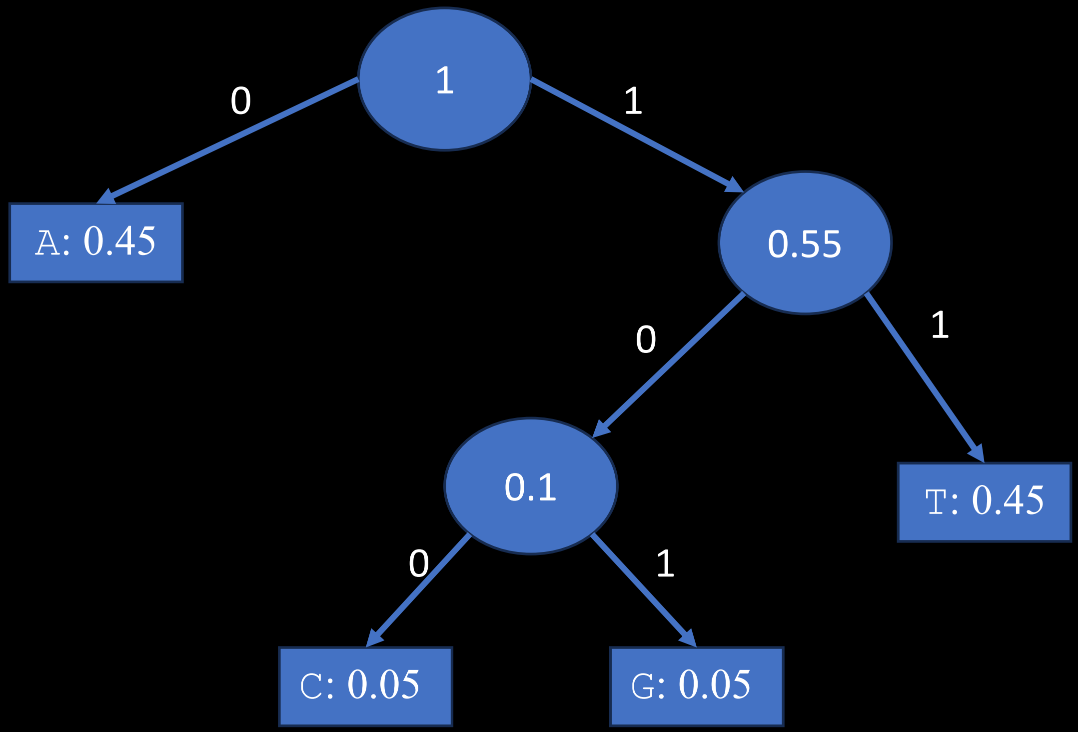
- The design of the algorithm uses priority queue, which is based on the heap.

1. Let Q be an empty priority queue.
2. For $i = 1$ to n: O(n)
   A. Construct a new node z containing $char[i]$ and whose frequency is $freq[i]$.
   B. Call INSERT(Q, z). O(lg n)
3. For $i = 1$ to $n - 1$: O(n)
   A. Call EXTRACT-MIN(Q), and set x to the node extracted. O(lg n)
   B. Call EXTRACT-MIN(Q), and set y to the node extracted. O(lg n)
   C. Construct a new node z whose frequency is the sum of x's frequency and y's frequency.
   D. Set z's left child to be x and z's right child to be y.
   E. Call INSERT(Q, z). O(lg n)
4. Call EXTRACT-MIN(Q), and return the node extracted. O(lg n)

$$T(n) = O(n \lg n)$$

Huffman Coding

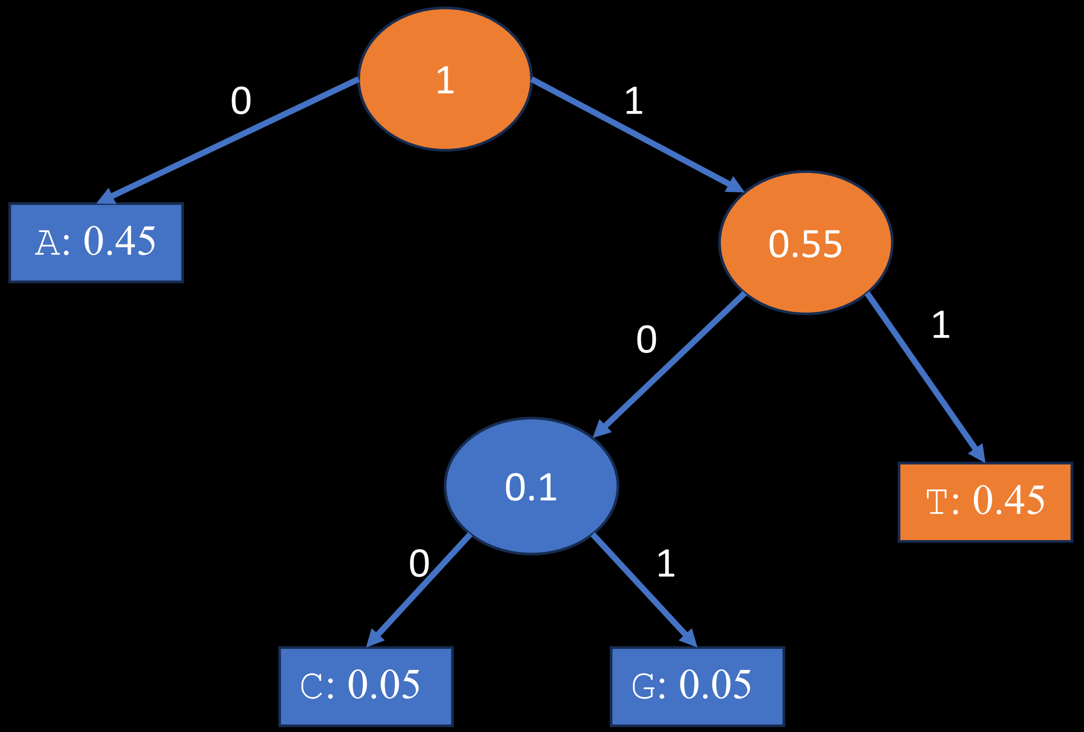
- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 1100111101010001111100110111011

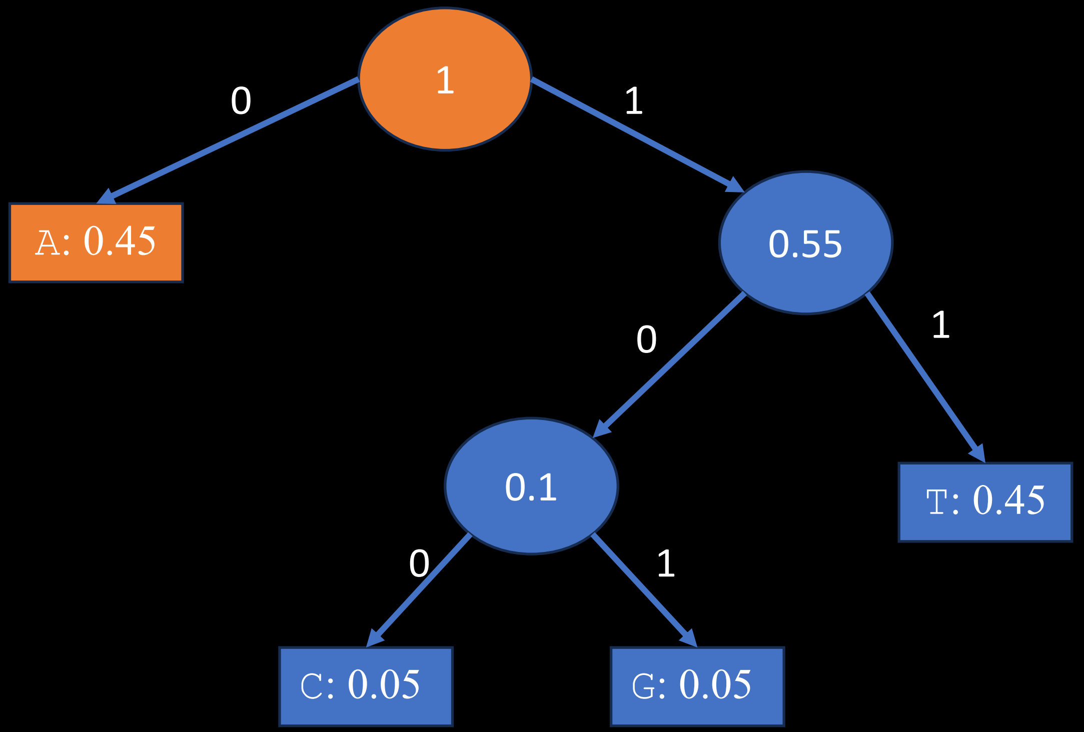
T



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

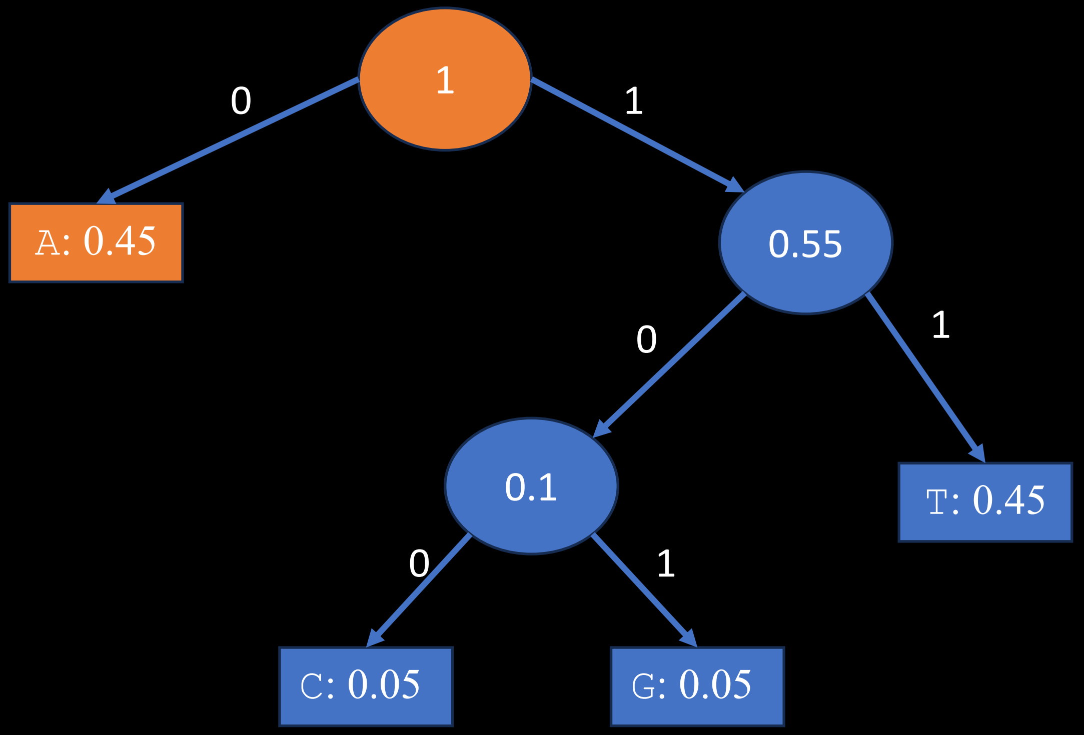
TA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

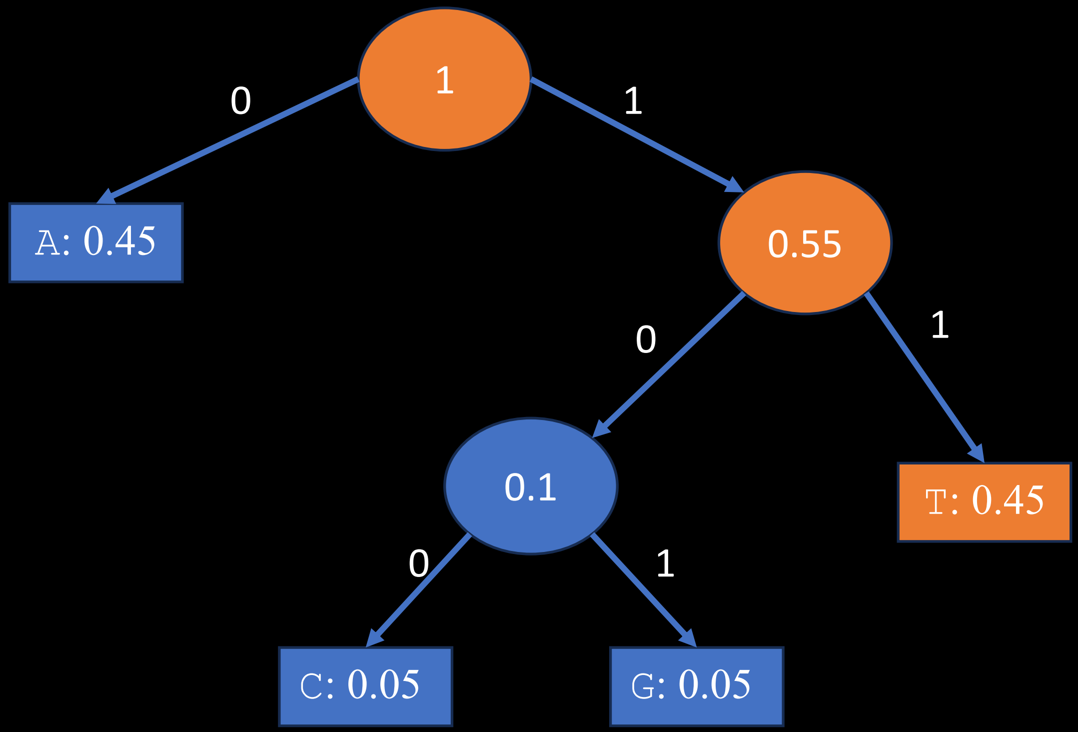
TAA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

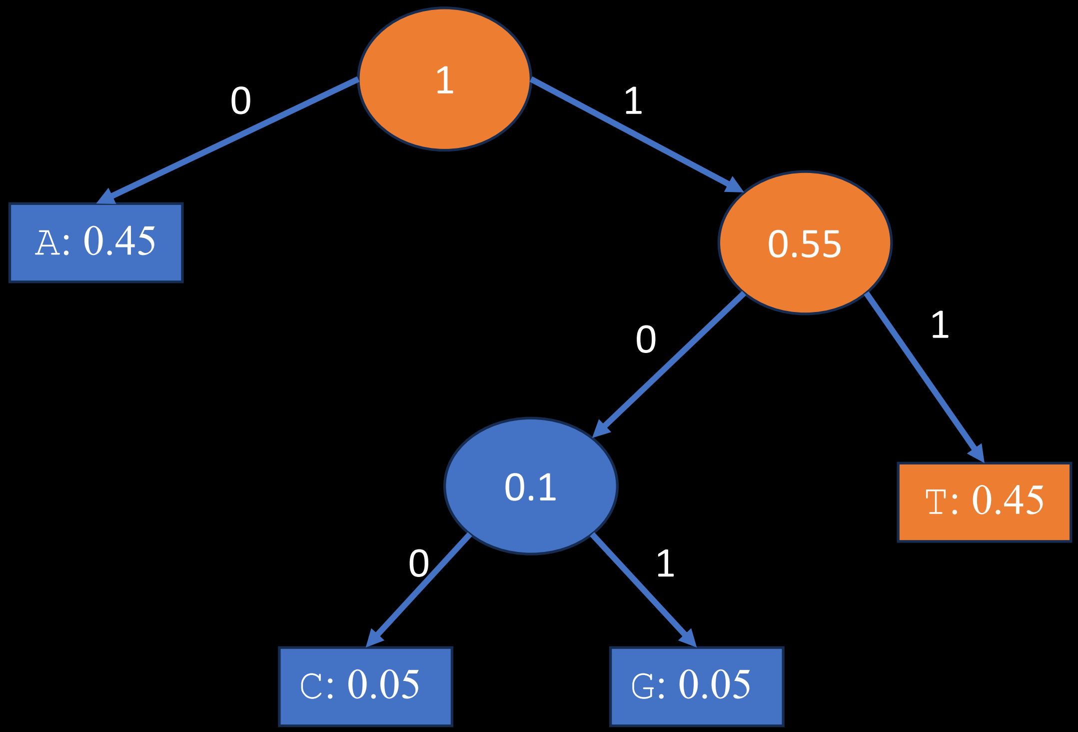
TAAT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

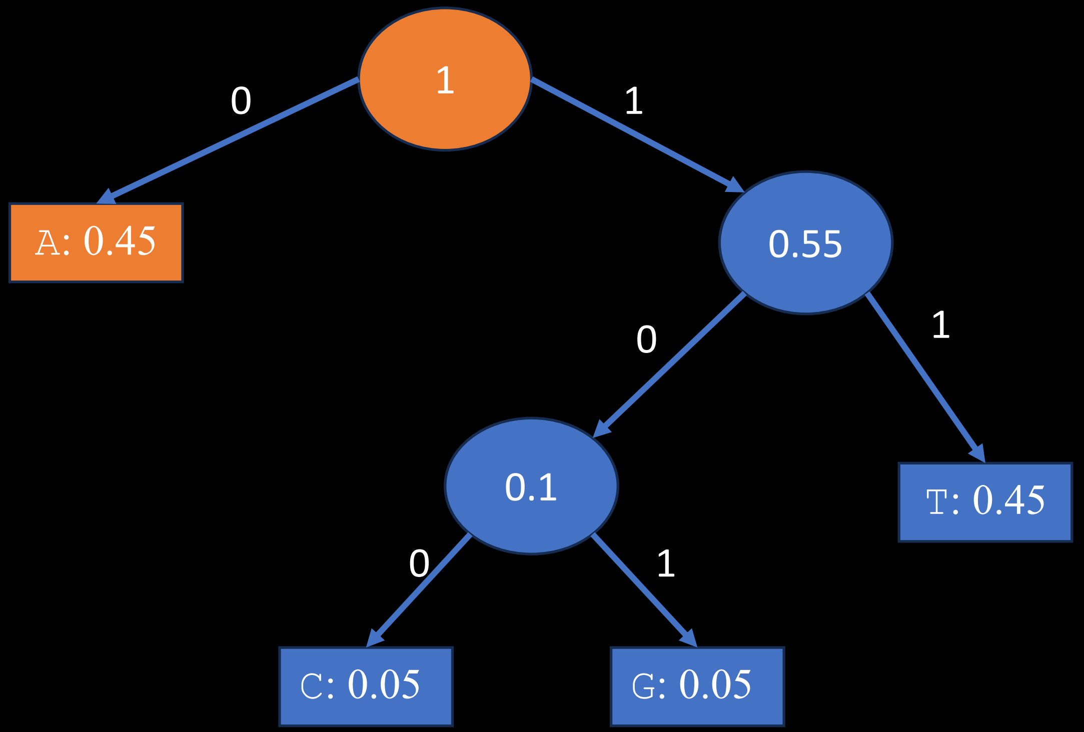
TAATT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

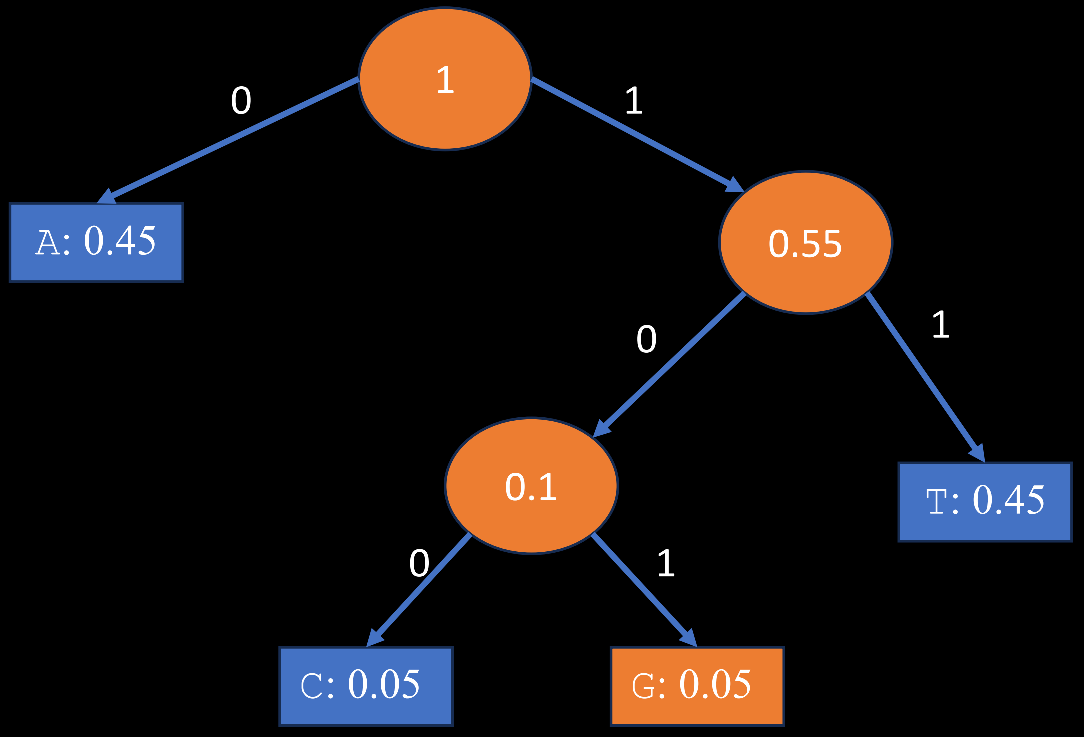
TAATTA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

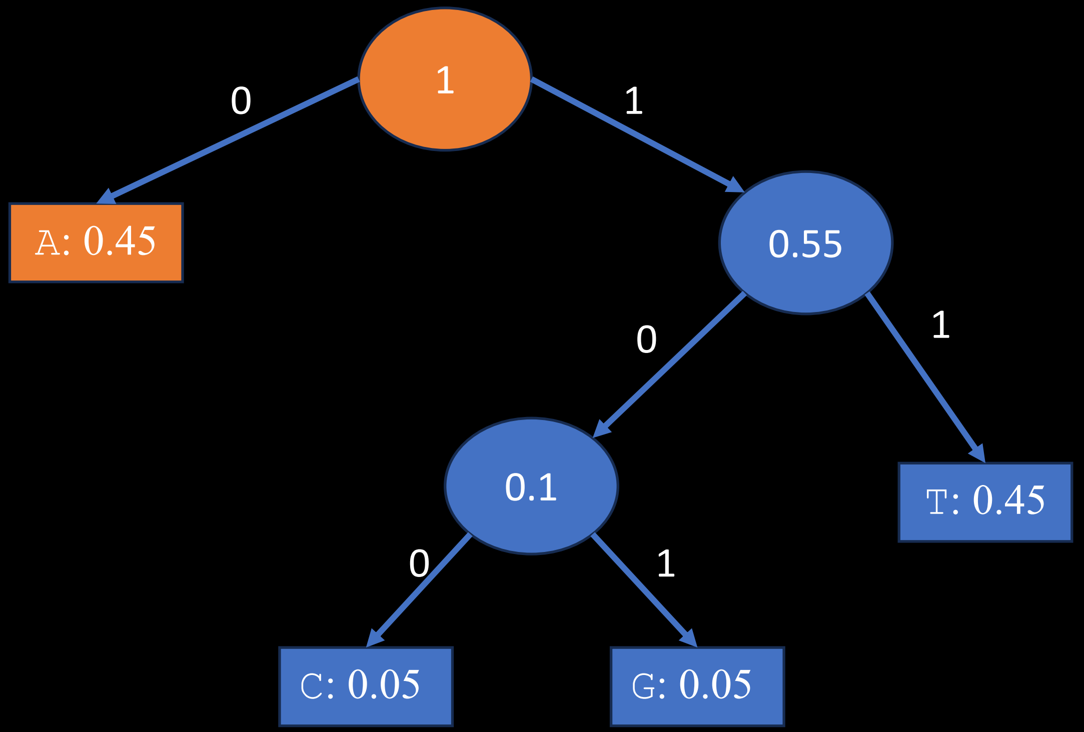
TAATTAG



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 1100111101010001111100110111101 1

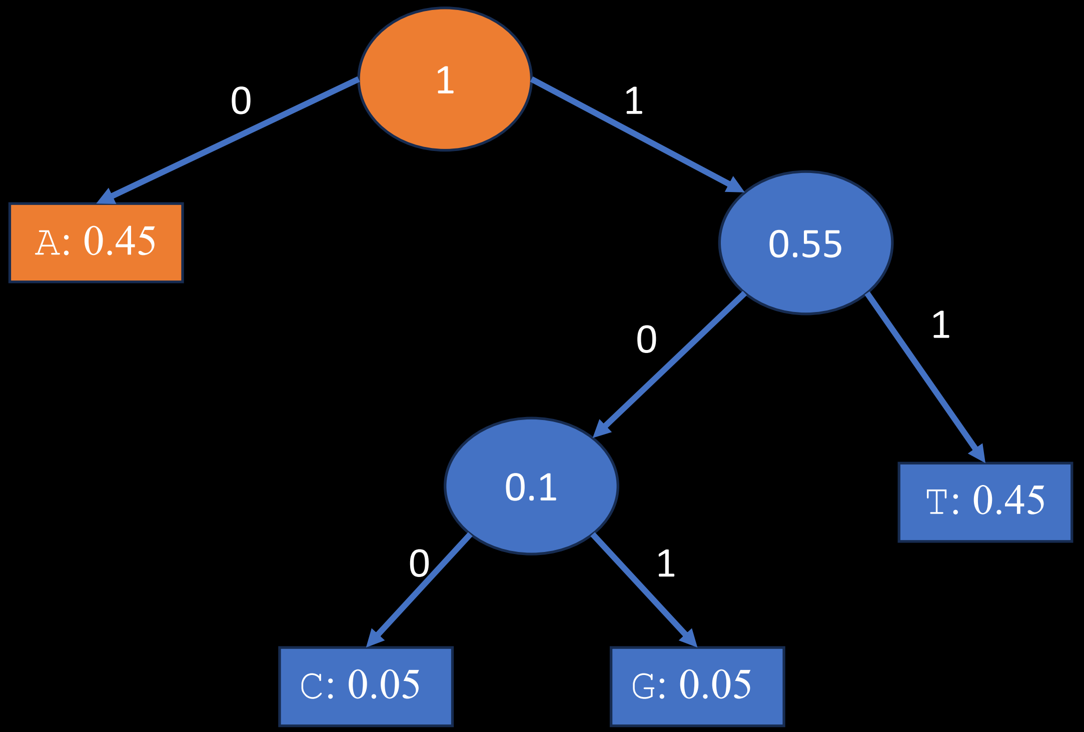
TAATTAGA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

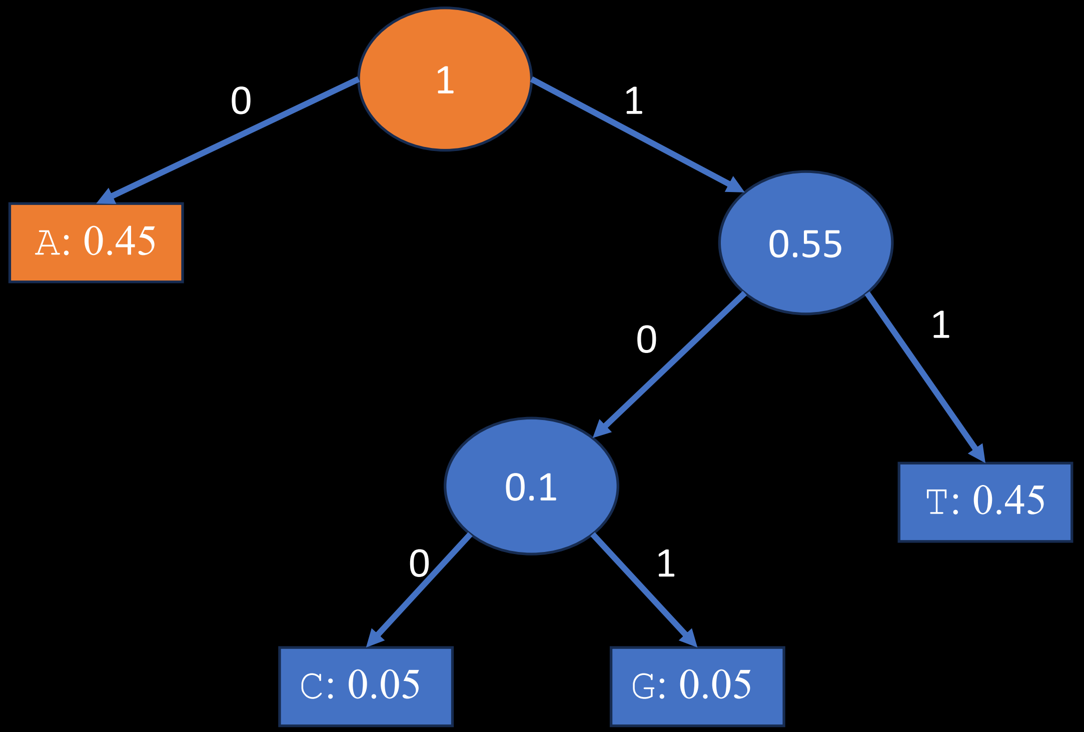
TAATTAGAA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

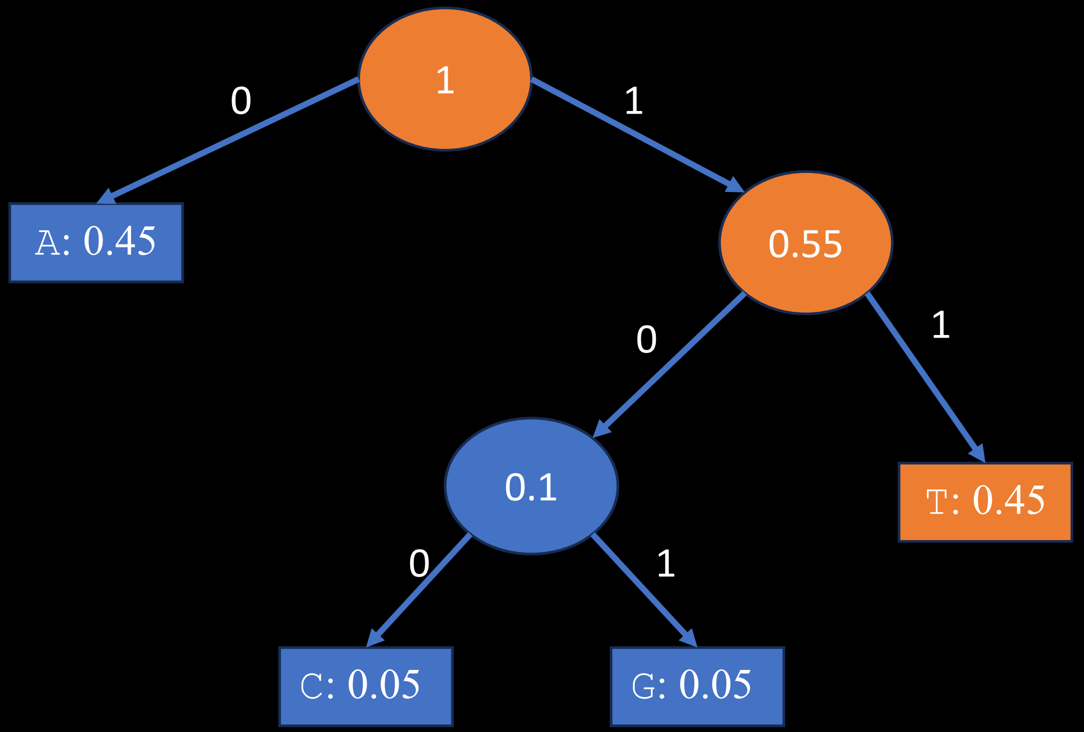
TAATTAGAAA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

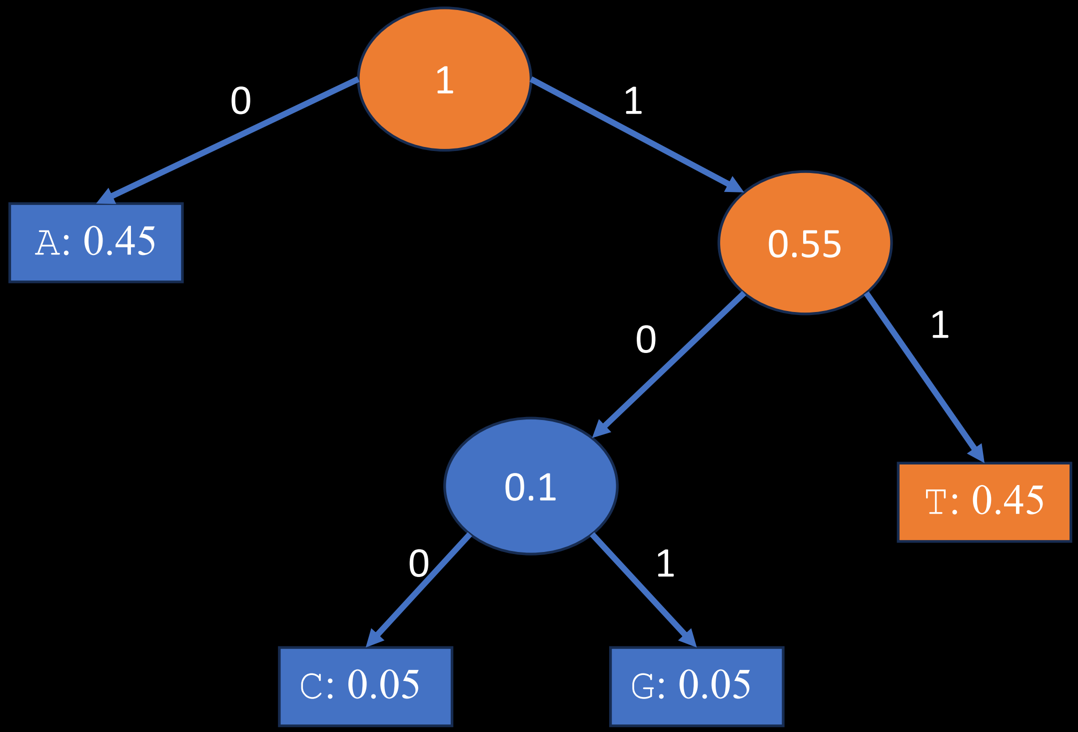
TAATTAGAAAT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 1100111101010001111001101111011

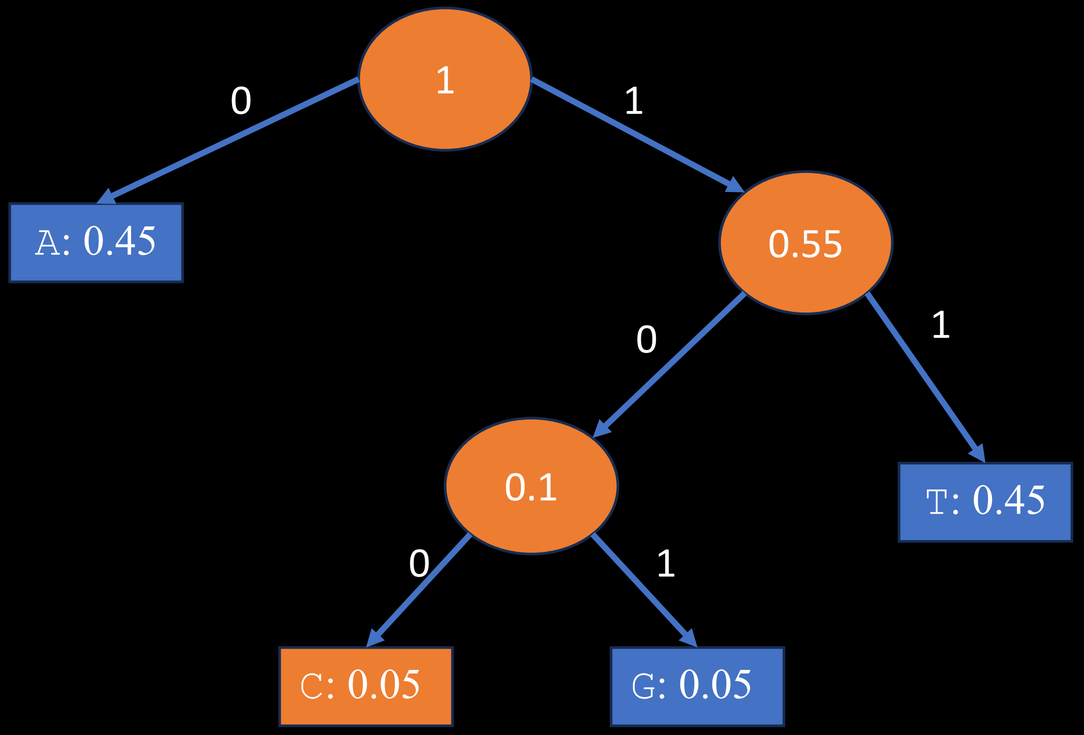
TAATTAGAAATT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

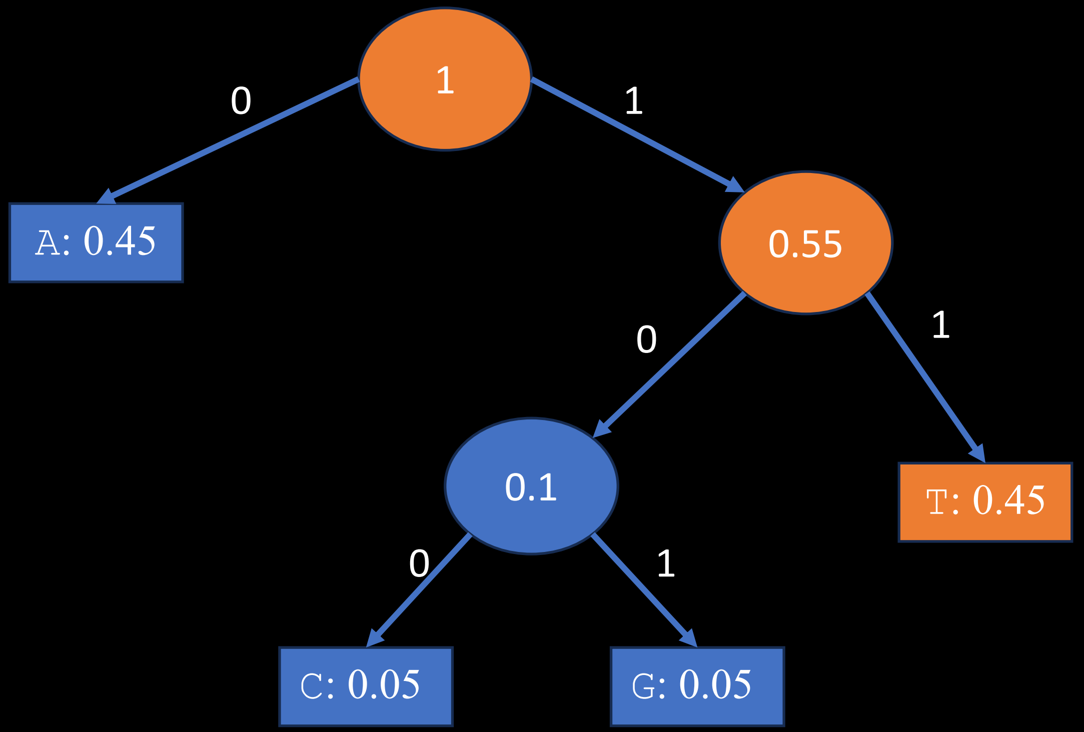
TAATTAGAAATTC



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

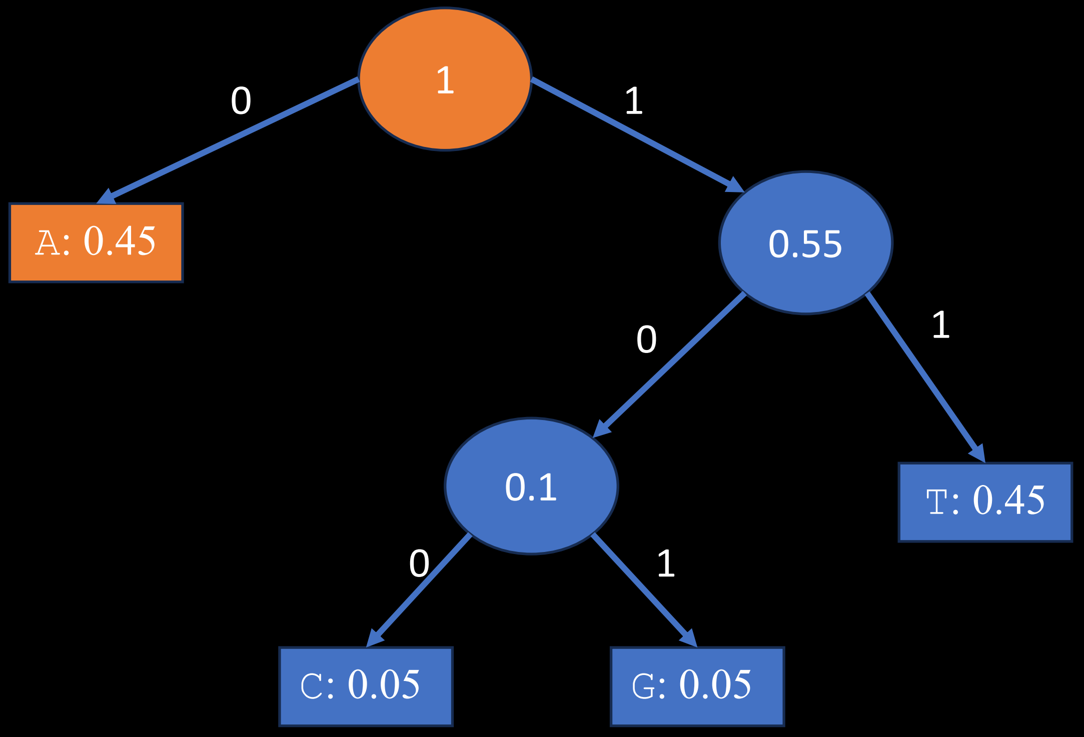
TAATTAGAAATTCT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

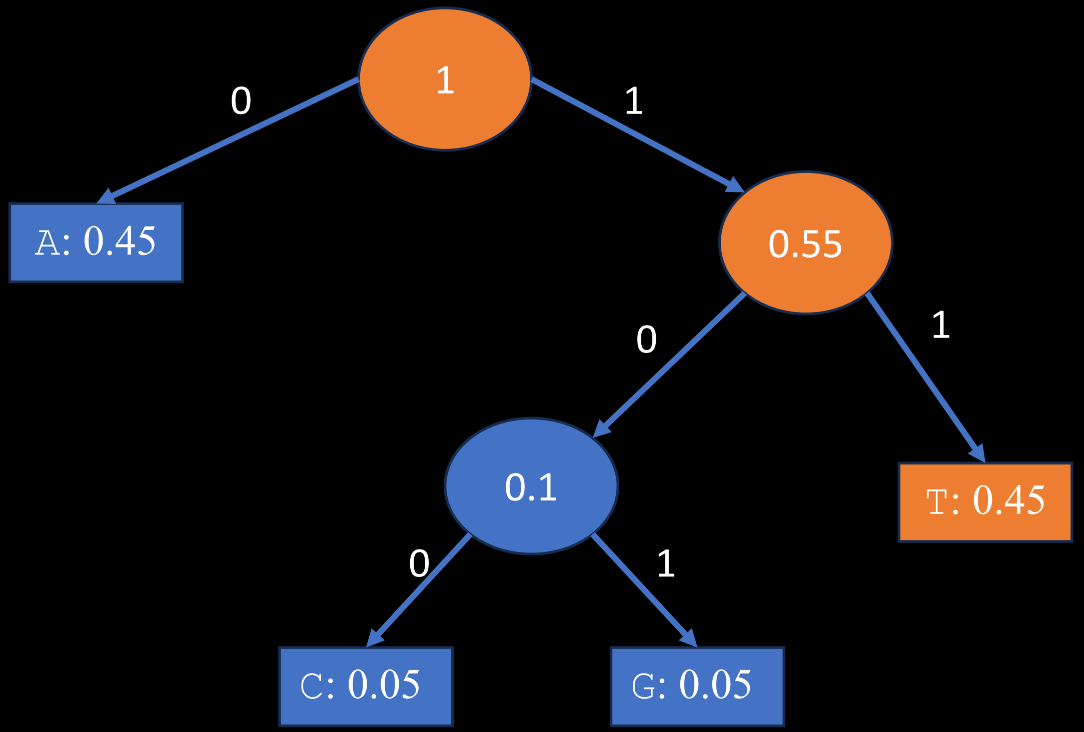
TAATTAGAAATTCTA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

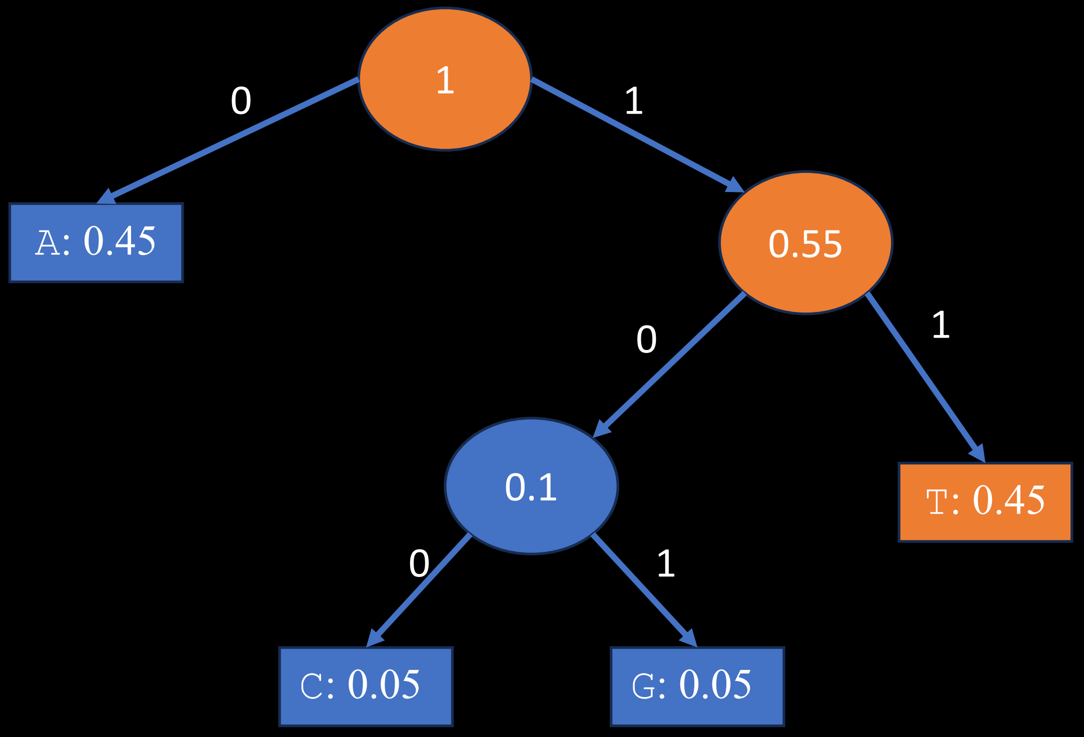
TAATTAGAAATTCTAT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011111011

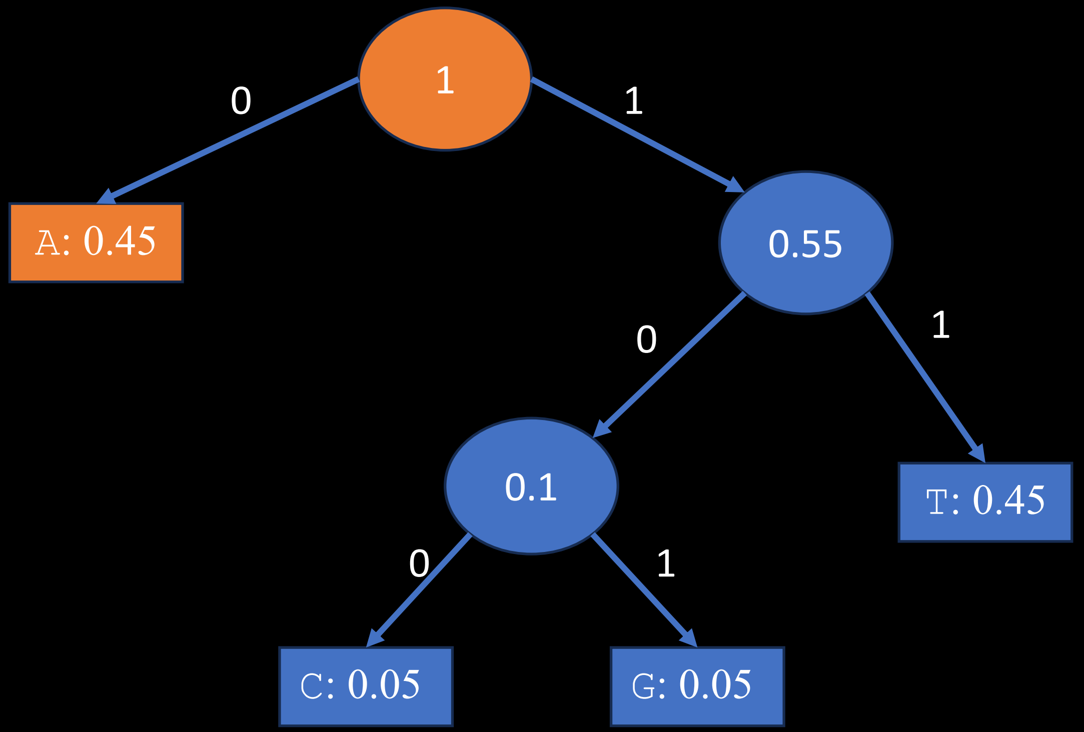
TAATTAGAAATTCTATT



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 110011110101000111110011011110011

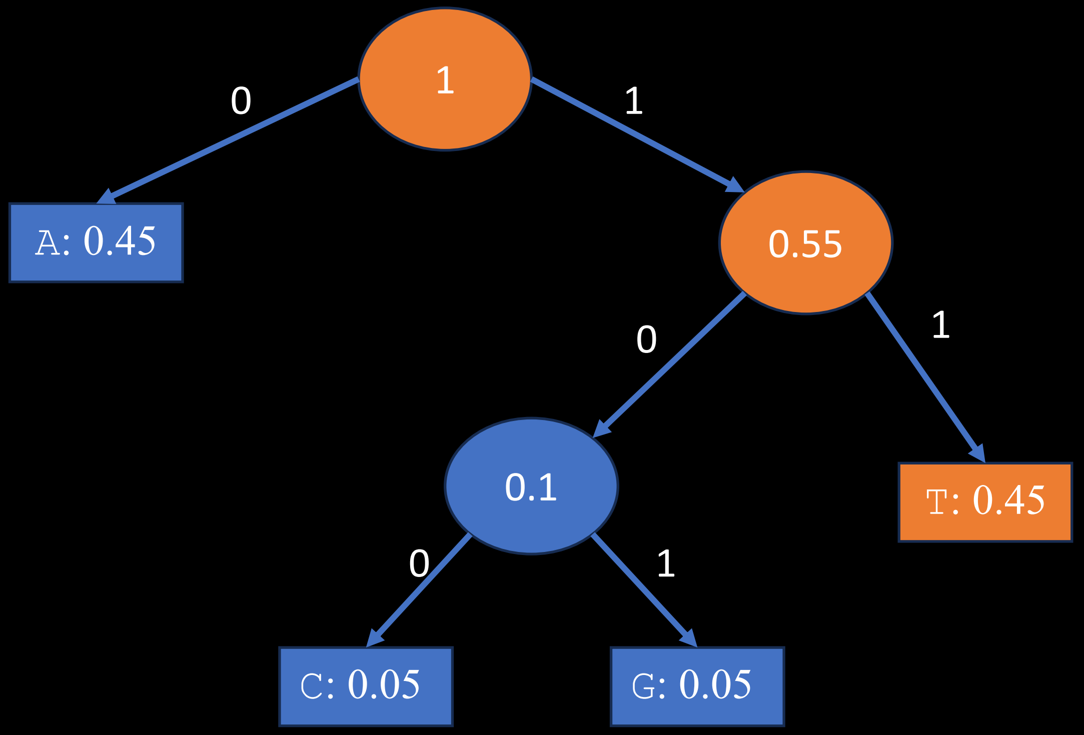
TAATTAGAAATTCTATTA



Huffman Coding

- When decompressing, follow the path from the root to the leaf.
- Example: decompress 11001111010100011111001101111011

TAATTAGAAATTCTATTAT



Huffman Coding

- When sending compressed data to another party, a copy of the tree must be sent to perform decompressing.
  - This can be achieved by sending a decoding table with the compressed data.

Character	Number of bits	Code
A	1	0
C	3	111
G	3	110
T	2	10

Huffman Coding

- Compress the following text “Eerie eyes seen near lake.”

Huffman Coding

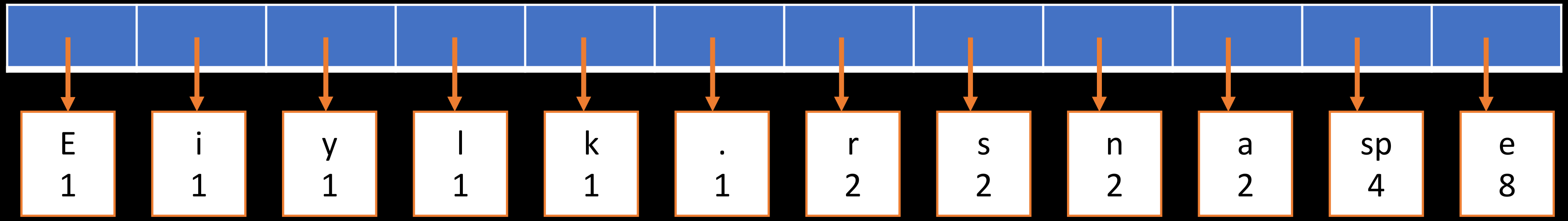
- Compress the following text "Eerie eyes seen near lake."

Count the frequency of the characters:

Char	Freq	Char	Freq
E	1	s	2
e	8	n	2
r	2	a	2
i	1	l	1
$space$	4	k	1
y	1	.	1

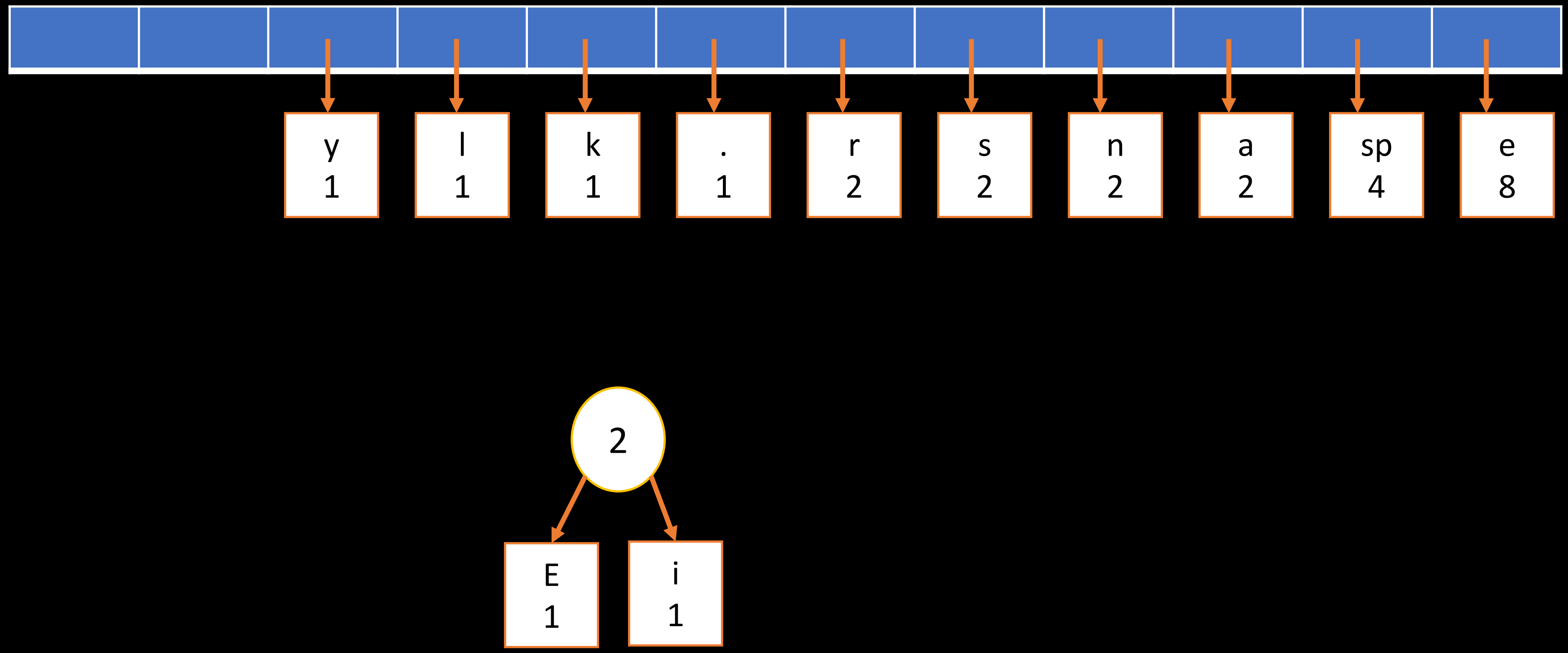
Huffman Coding

- Represent them in a min heap



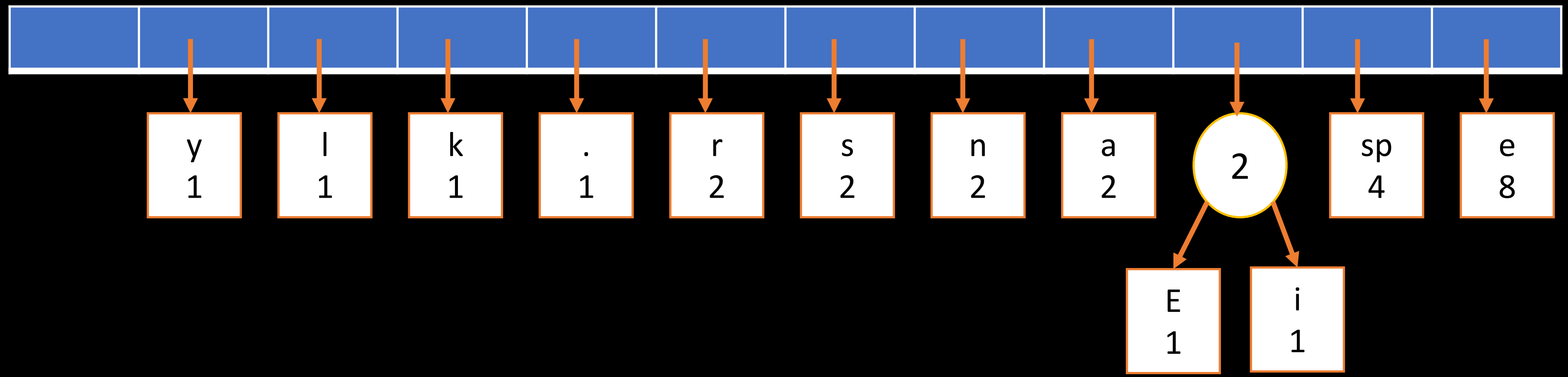
Huffman Coding

- Represent them in a min heap



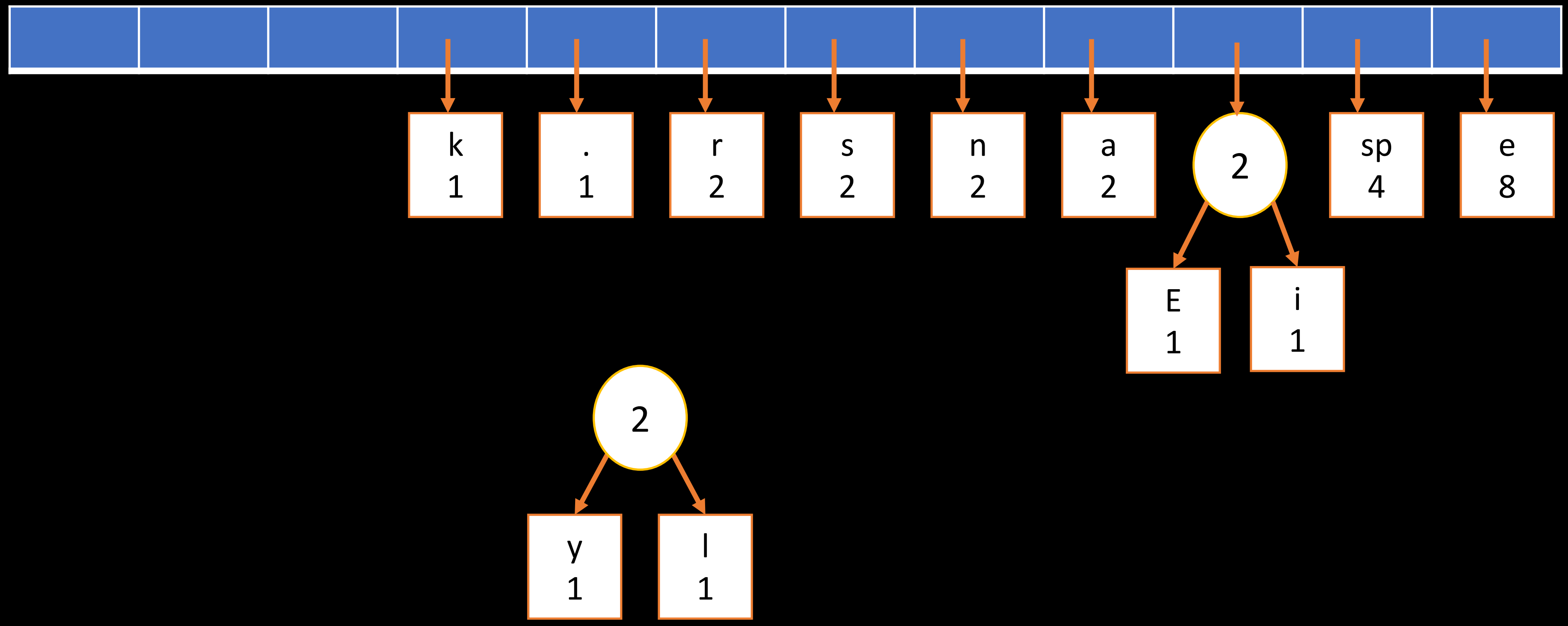
Huffman Coding

- Represent them in a min heap



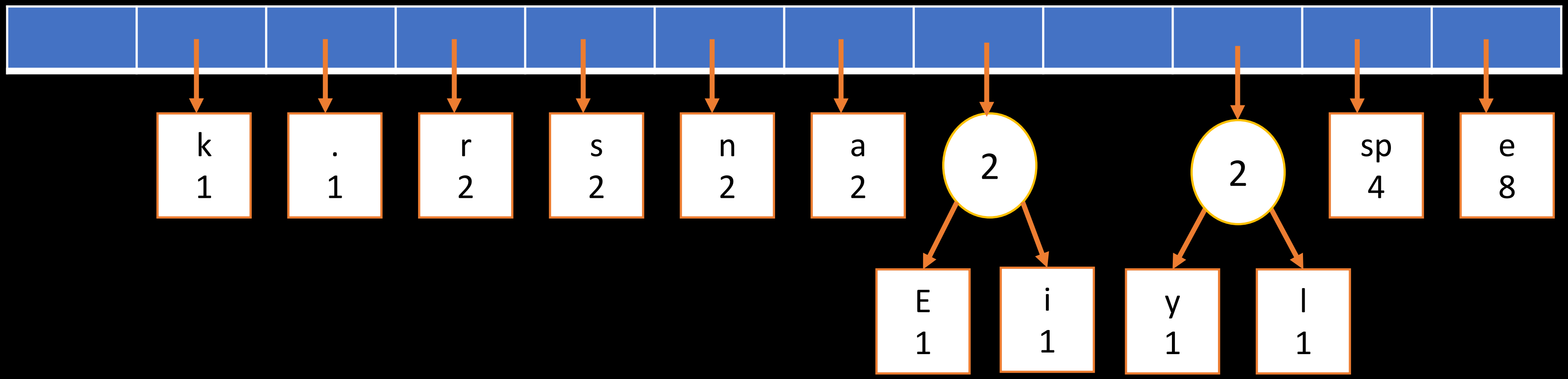
Huffman Coding

- Represent them in a min heap



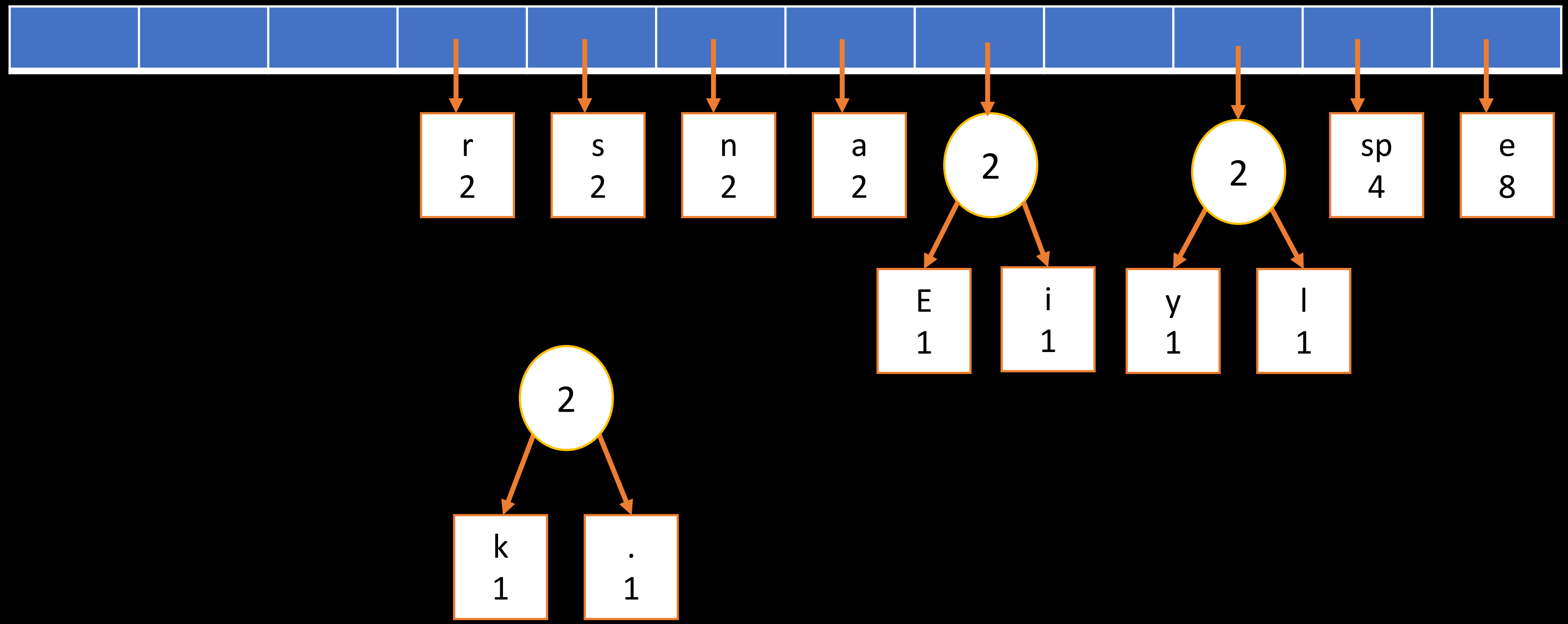
Huffman Coding

- Represent them in a min heap



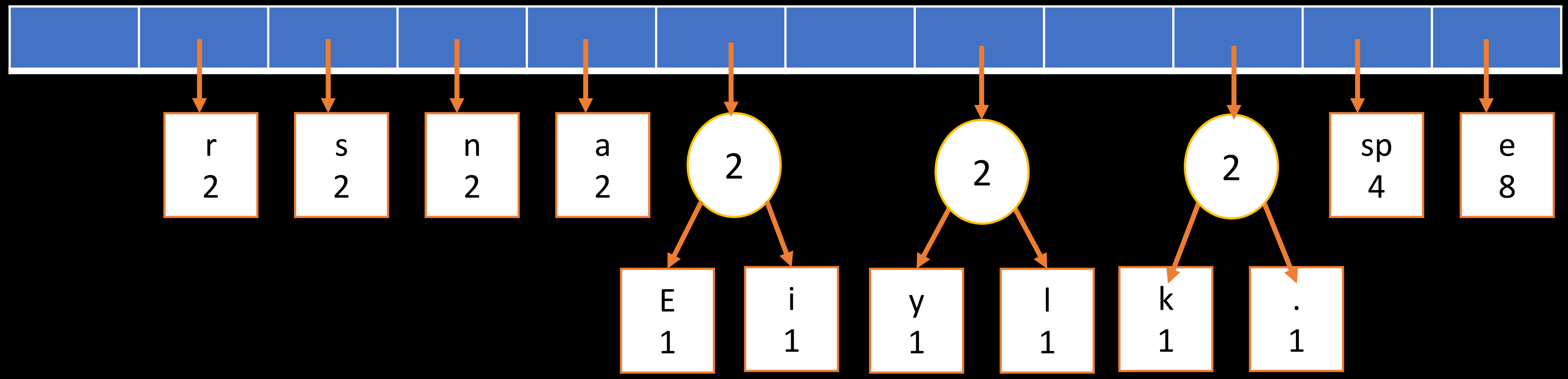
Huffman Coding

- Represent them in a min heap



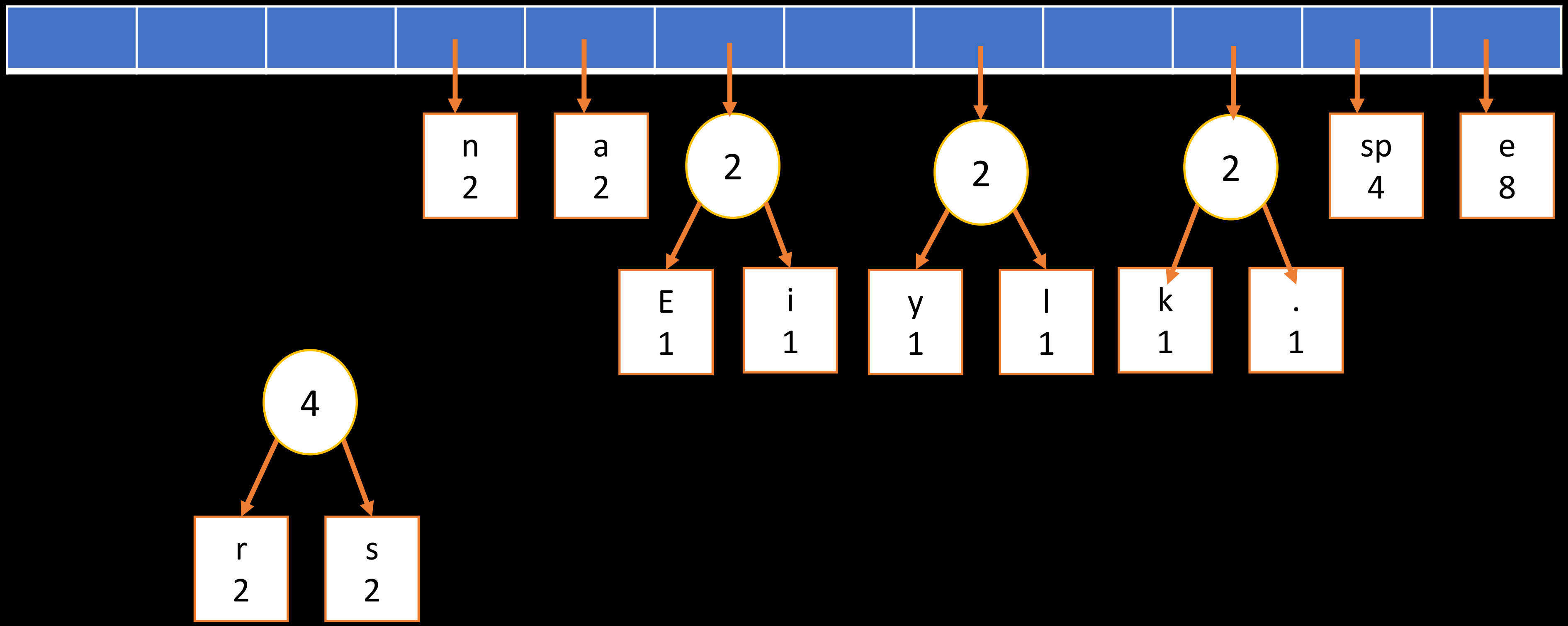
Huffman Coding

- Represent them in a min heap



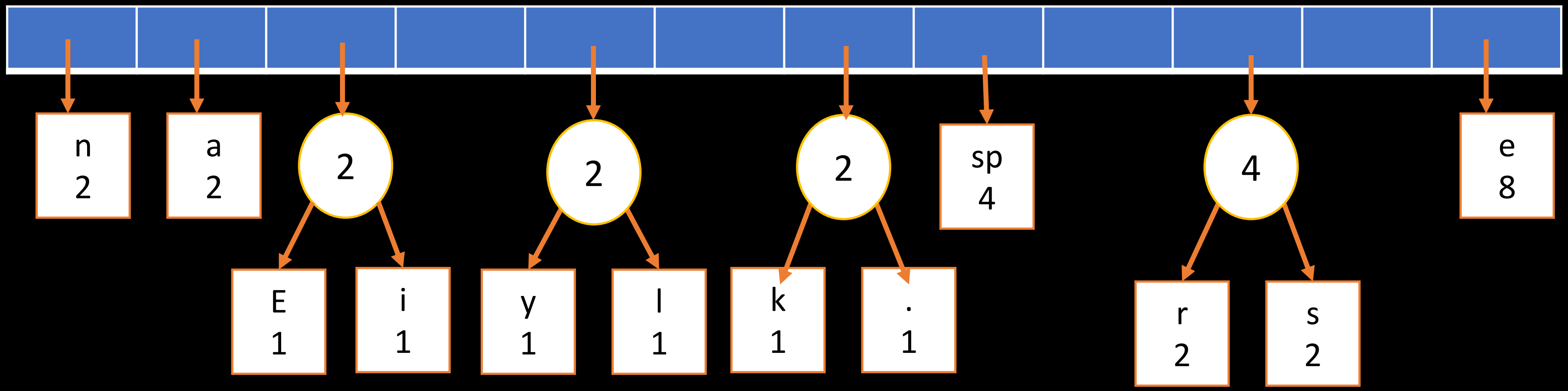
Huffman Coding

- Represent them in a min heap



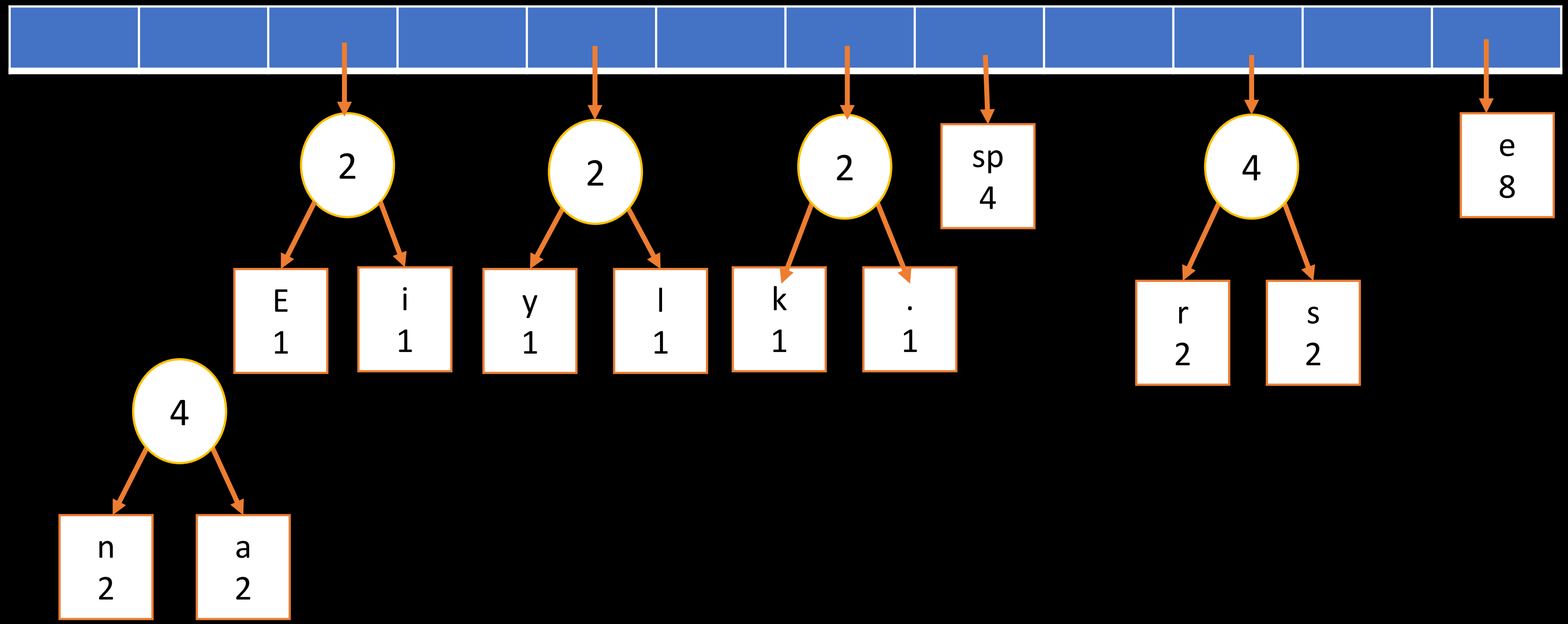
Huffman Coding

- Represent them in a min heap



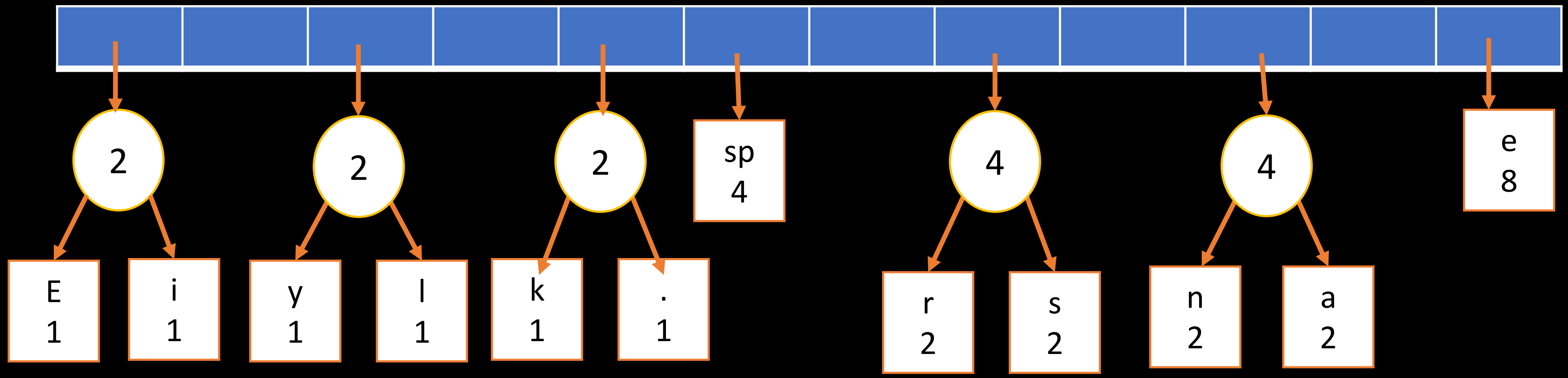
Huffman Coding

- Represent them in a min heap



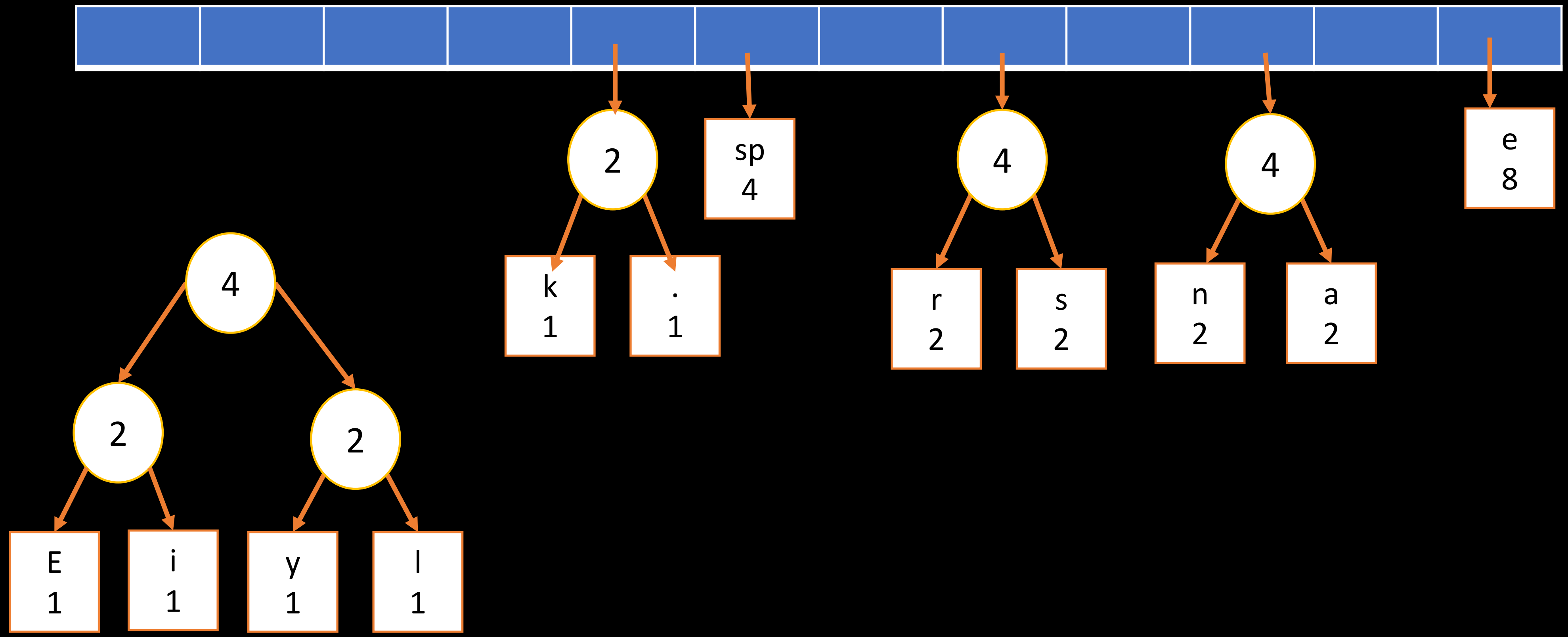
Huffman Coding

- Represent them in a min heap



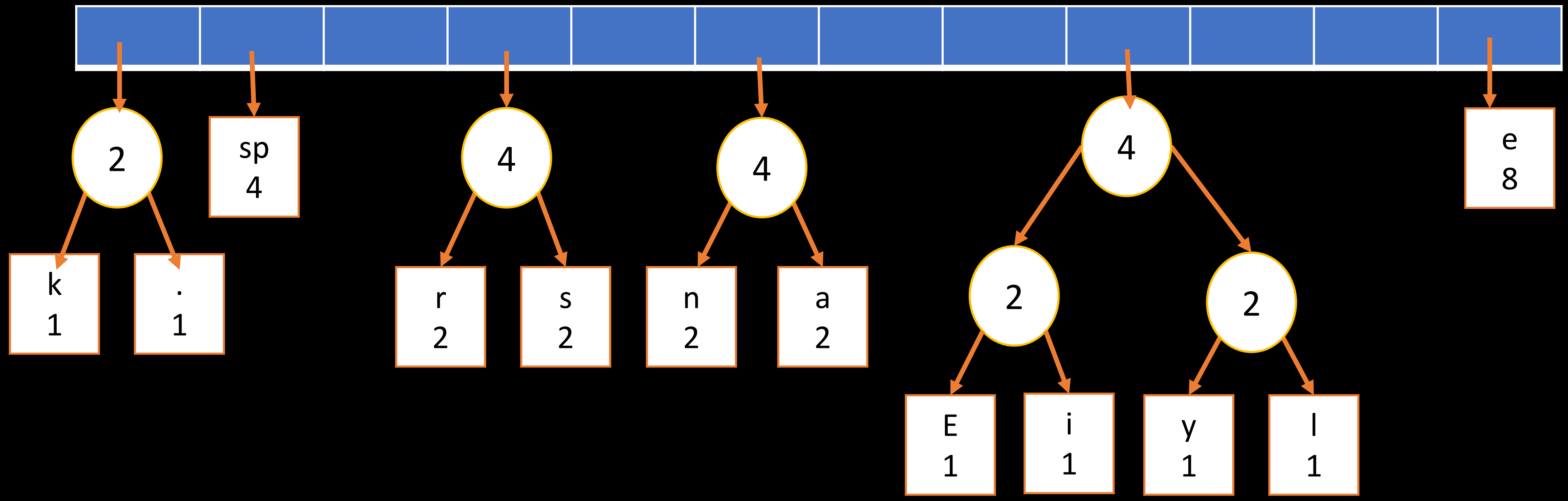
Huffman Coding

- Represent them in a min heap



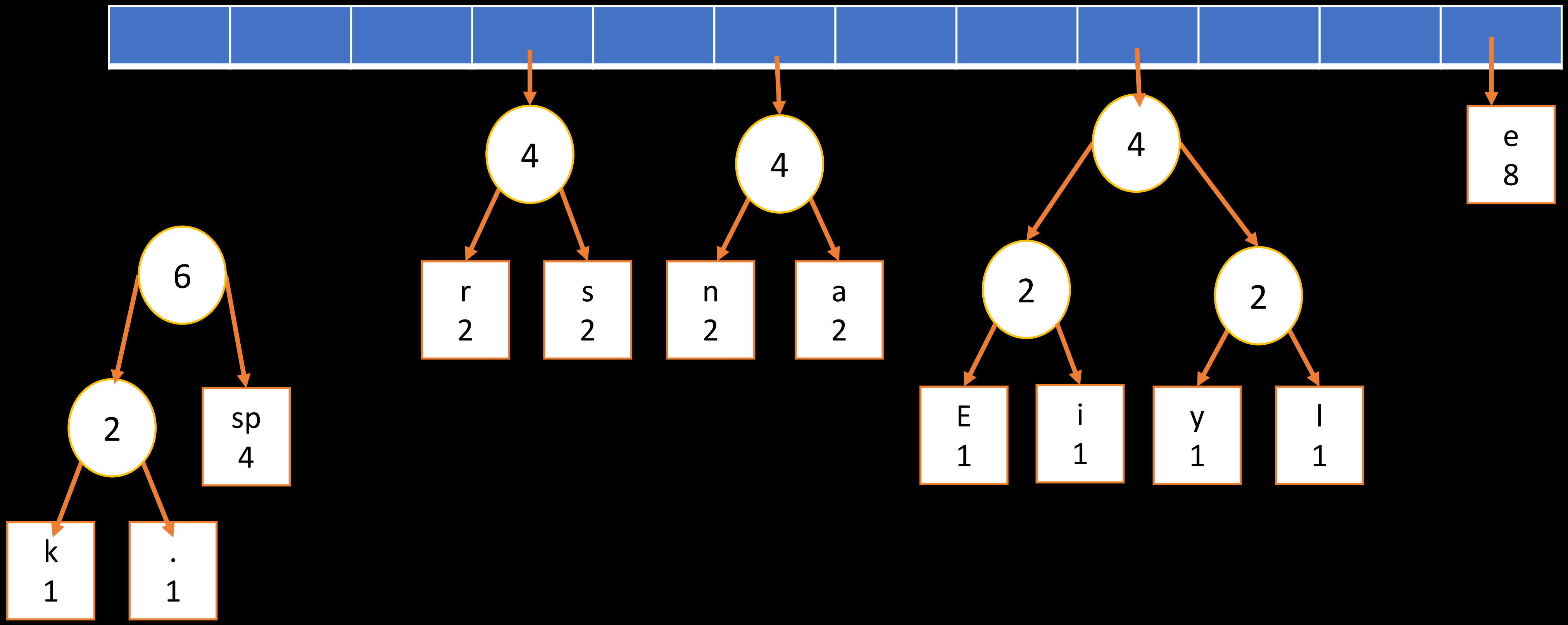
Huffman Coding

- Represent them in a min heap



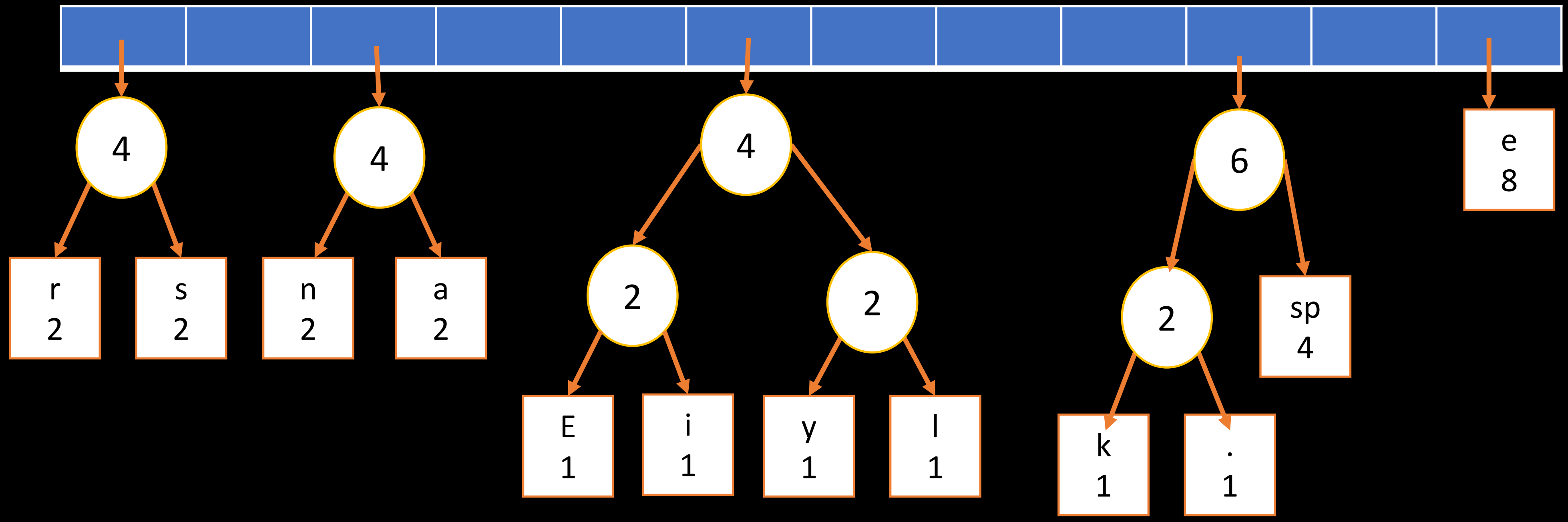
Huffman Coding

- Represent them in a min heap



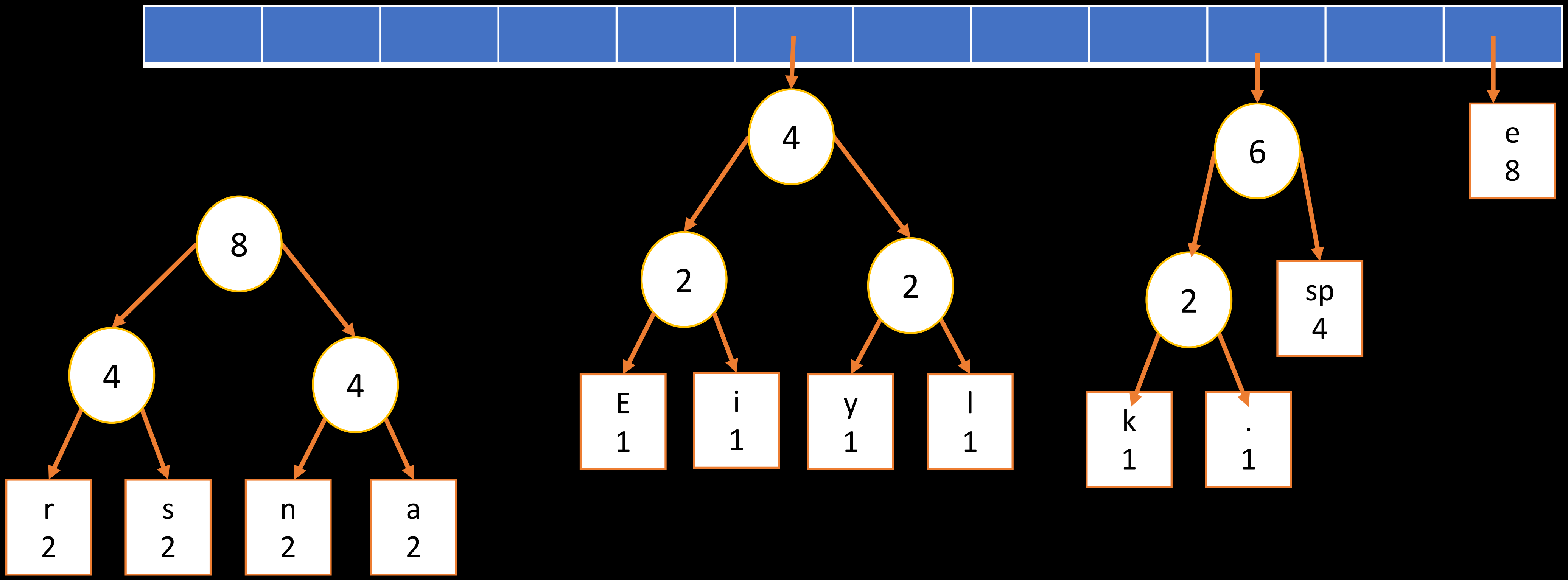
Huffman Coding

- Represent them in a min heap



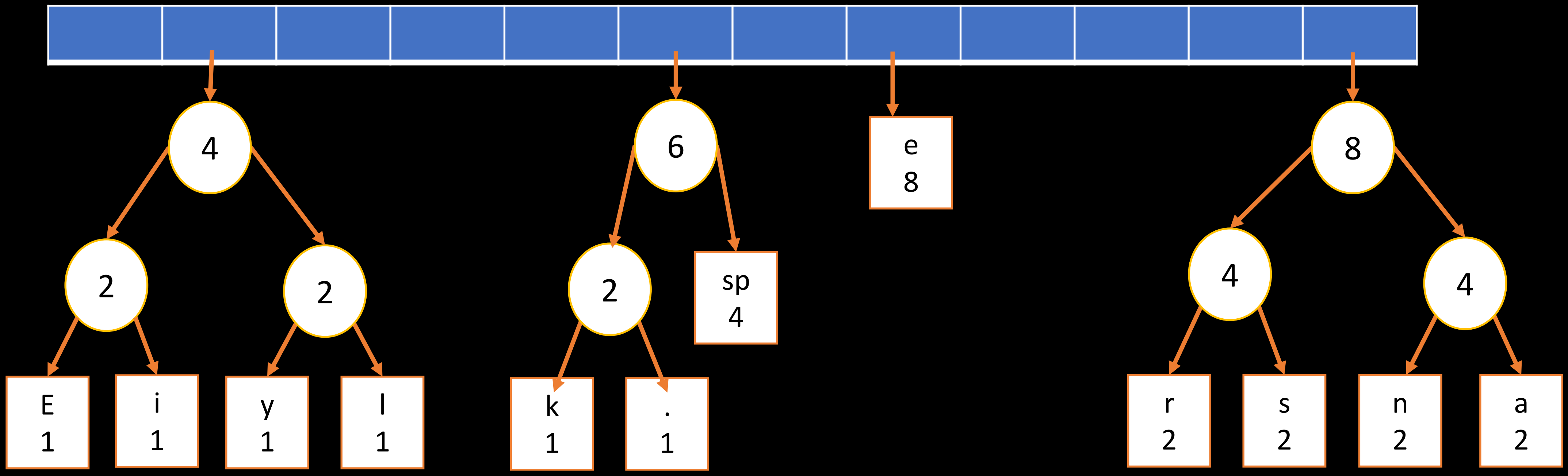
Huffman Coding

- Represent them in a min heap



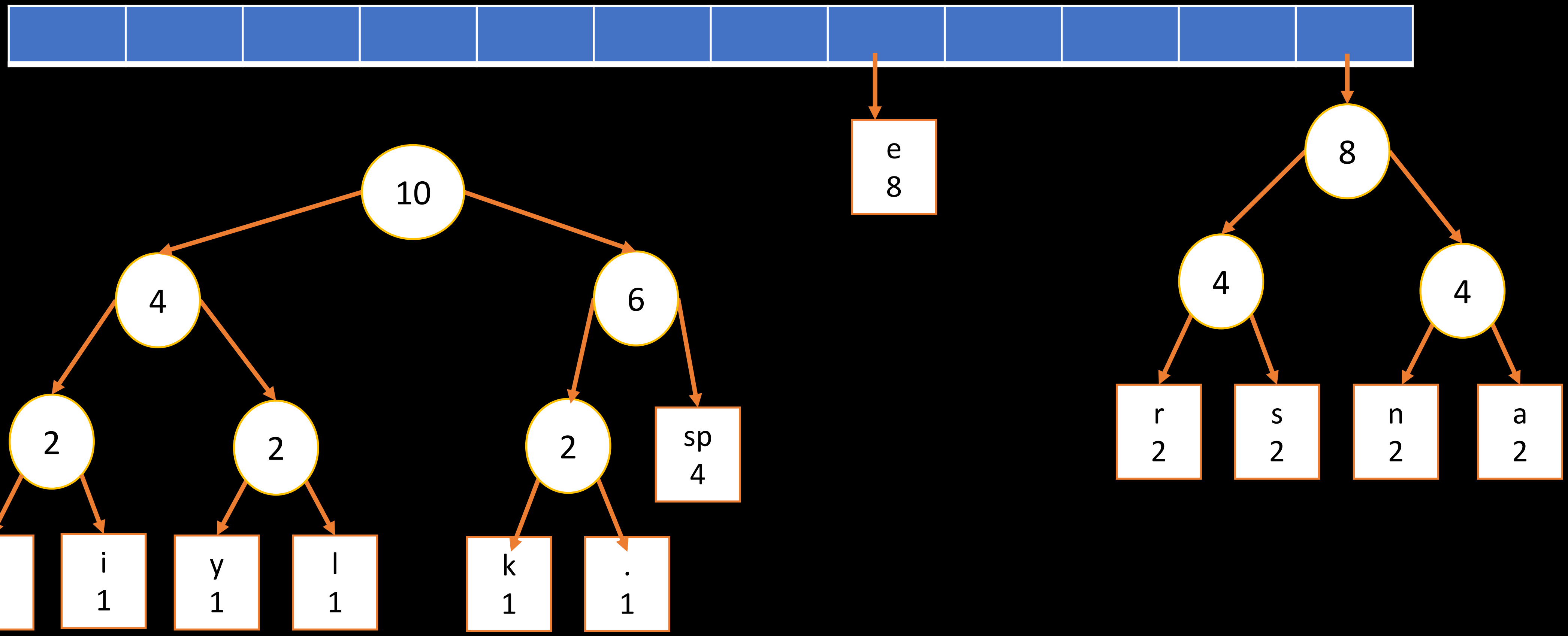
Huffman Coding

- Represent them in a min heap



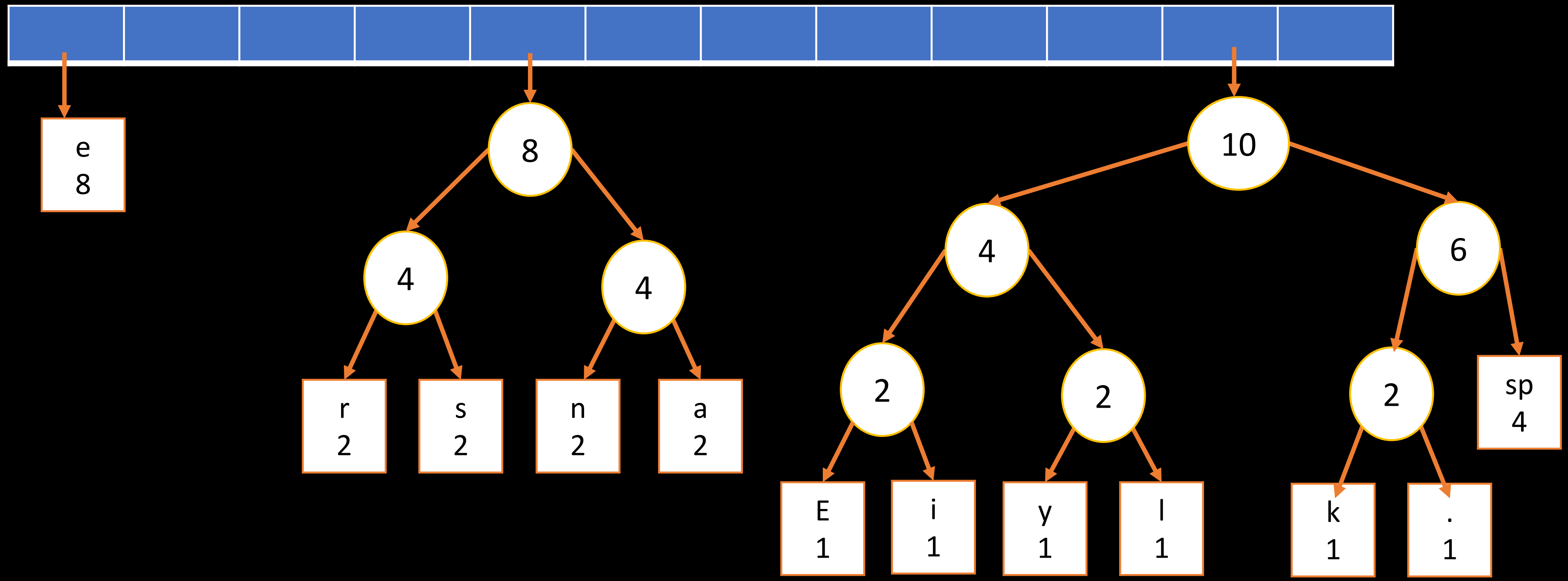
Huffman Coding

- Represent them in a min heap



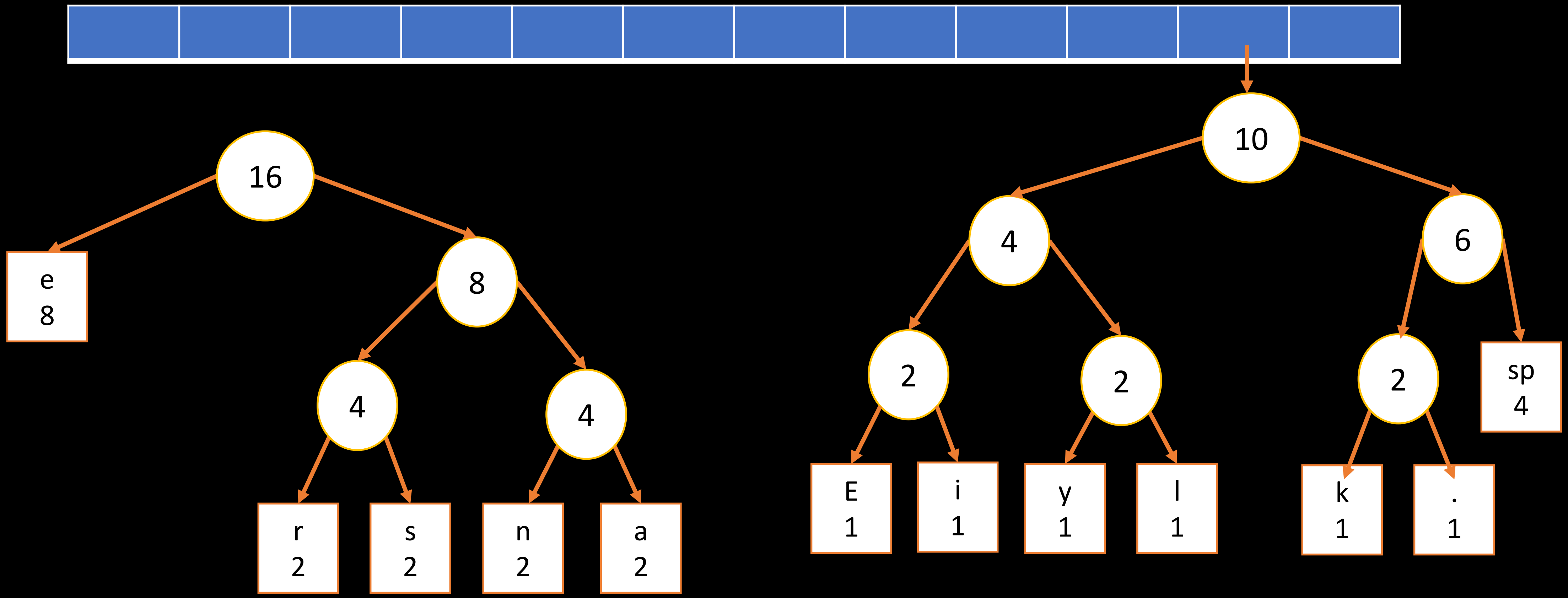
Huffman Coding

- Represent them in a min heap



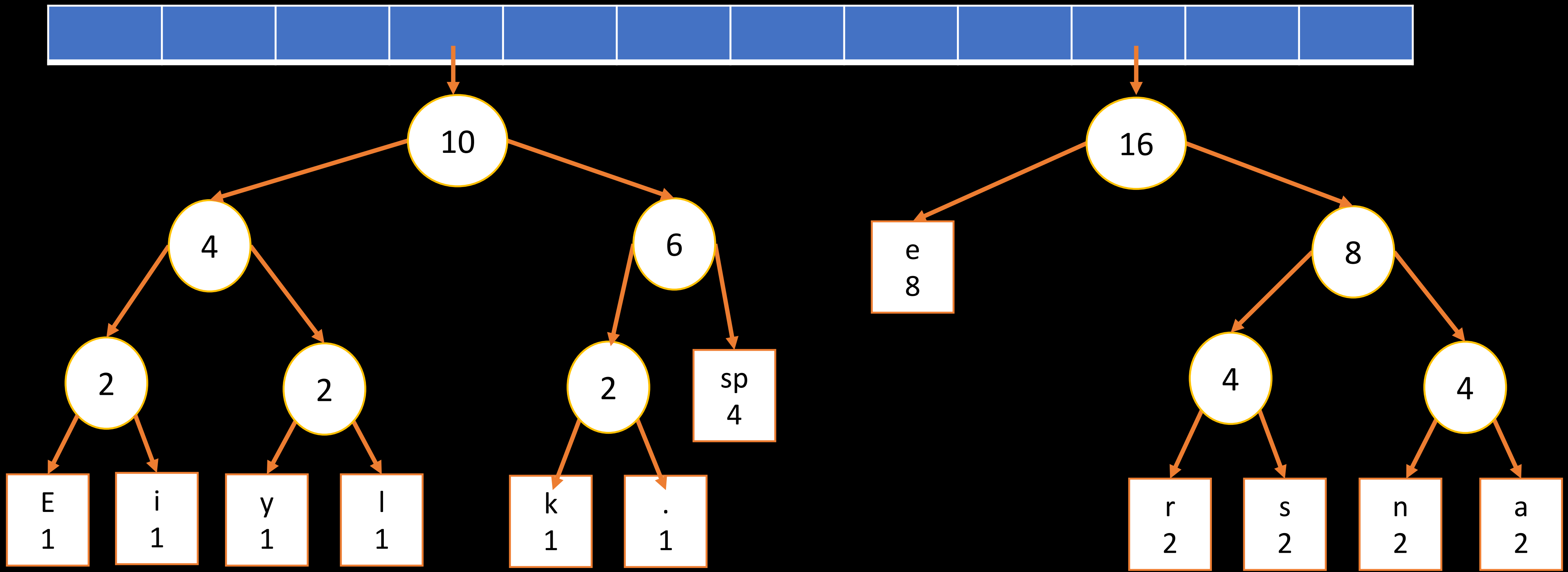
Huffman Coding

- Represent them in a min heap

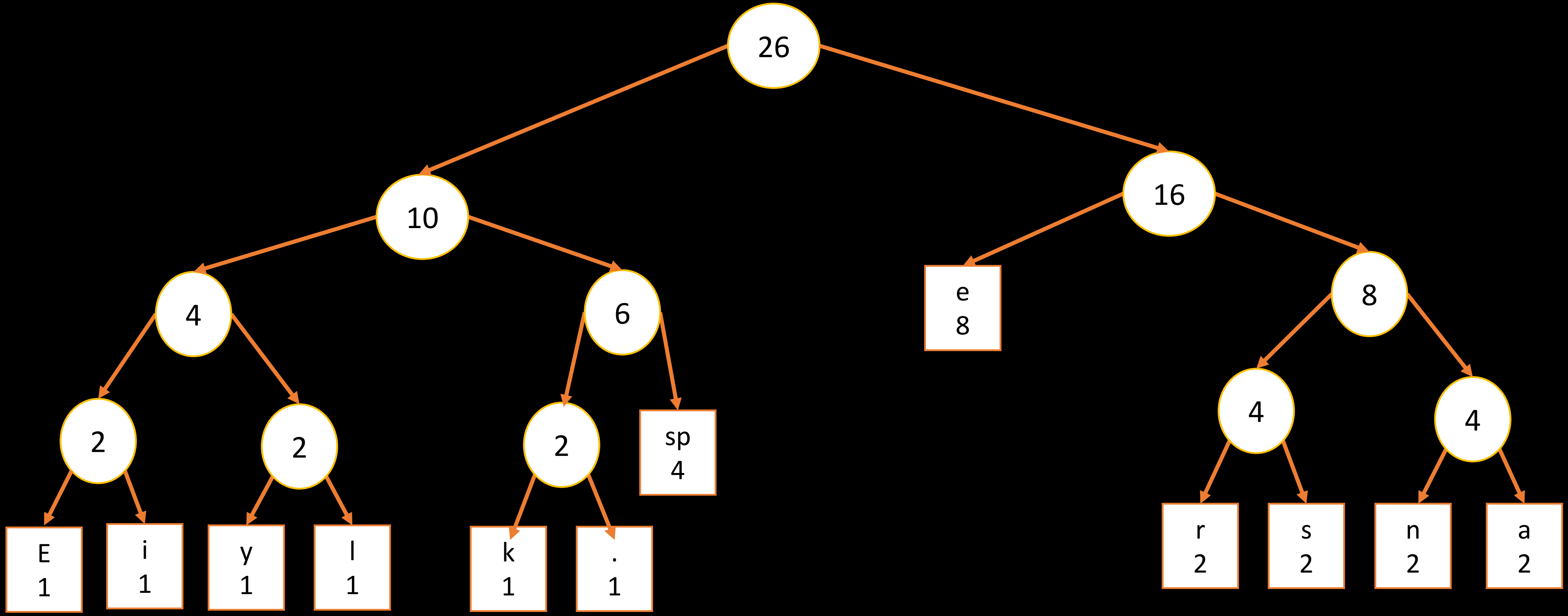


Huffman Coding

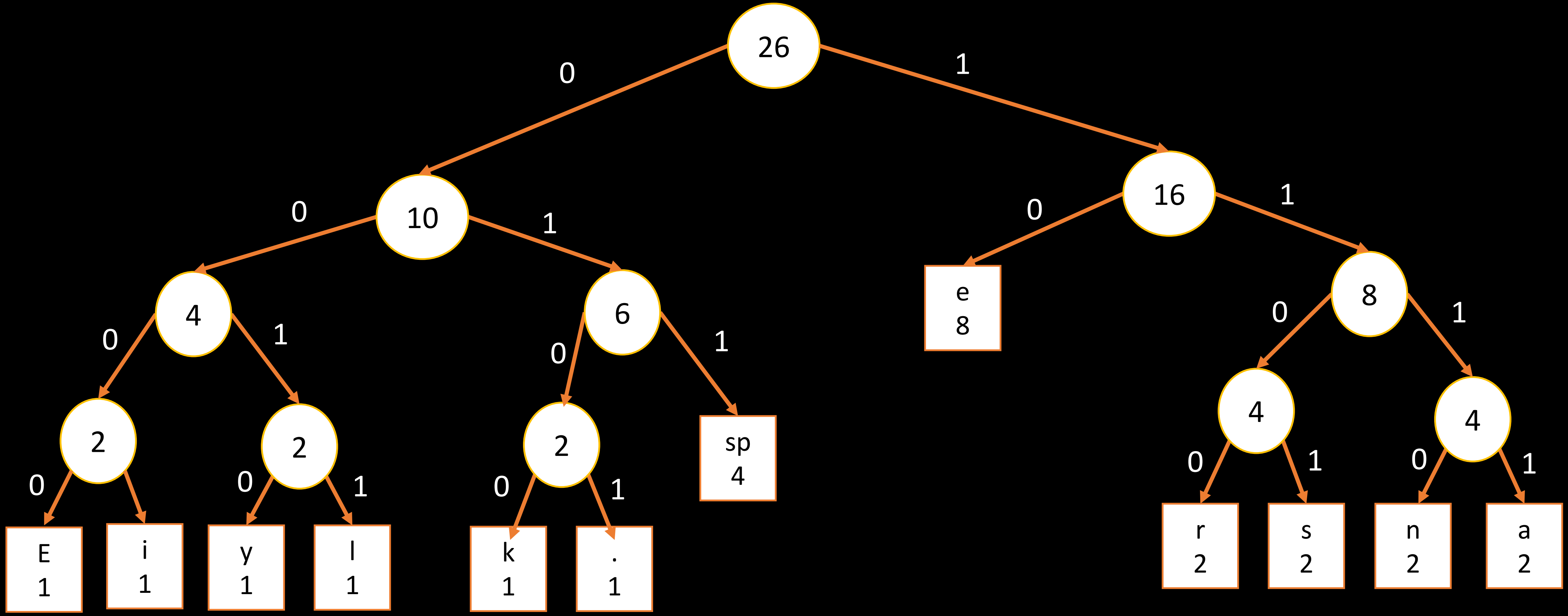
- Represent them in a min heap



Huffman Coding

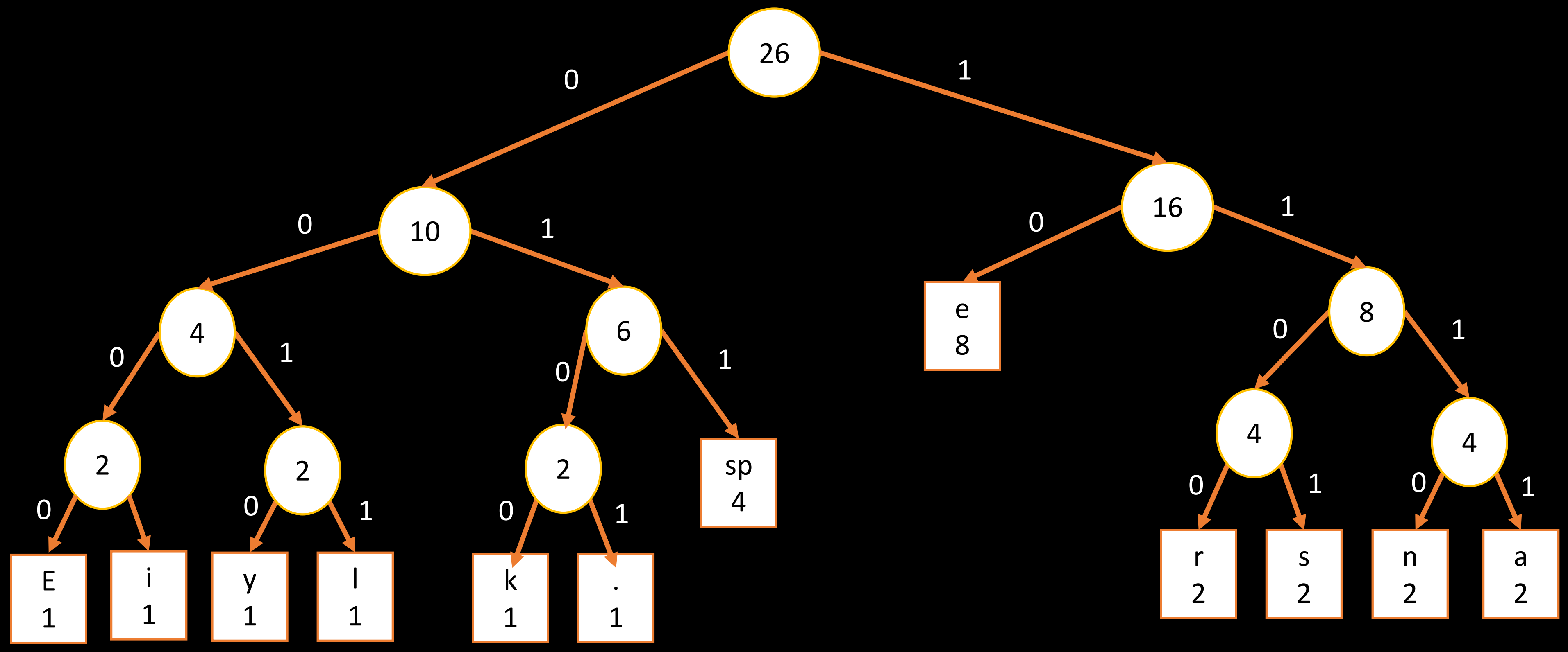


Huffman Coding



Huffman Coding

Character	code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
space	011
e	10
r	1100
s	1101
n	1110
a	1111



Huffman Coding

- Huffman algorithm is a greedy algorithm.

- We make the decision that seems best at the moment.
  - The least-frequently appearing characters far from the root of the binary tree,
  - Selects the two nodes with the lowest frequency to place under a new node.

Content

Content
Greedy Algorithms
Huffman Coding
Adaptive Huffman Coding
Exercise

Adaptive Huffman Coding

- Instead of making two passes over the information to compute frequencies.

- Make compression and decompression work adaptively.
  - Updating character frequencies and the binary tree as they compress or decompress in just one pass.

Adaptive Huffman codes

- Steps:
  - Start with an empty binary tree
  - Read a character α
  - If α is in the tree:
    - output the character according to the current encoding of the tree
    - Increase α's frequency
    - Update the binary tree to reflect new frequencies
  - If α is not in the tree:
    - Output the original encoding of the character (as is)
    - Add α to the binary tree
    - Update the tree accordingly

Adaptive Huffman codes

- Both internal and leaf nodes in the adaptive Huffman tree has two elements:
    - A weight: the number of times the node has been encountered so far.
    - A number: a unique integer id for each node.

- The root node starts with weight 0 and its number is calculated by:

$$2n \ - 1$$

    - Where n is the number of possible characters.
    - Empty nodes are referred to as Not Yet Transmitted (NYT) nodes.
- The alphabets can be represented in 5 bits.

Adaptive Huffman codes

- When updating the tree, ensure that we maintain the siblings property.

- Weight(left sibling) $\leq$ weight(right sibling)

Adaptive Huffman codes

- Example: compress “aardvark” using adaptive Huffman encoding.
- Assume the character space is all the 26 alphabets.
- The first (NYT) root node has weight 0 and number $2 * 26 - 1 = 51$

Adaptive Huffman codes

NYT: 0 51

aardvark

Char	Code
NYT	0
a	
r	
d	
v	
k	

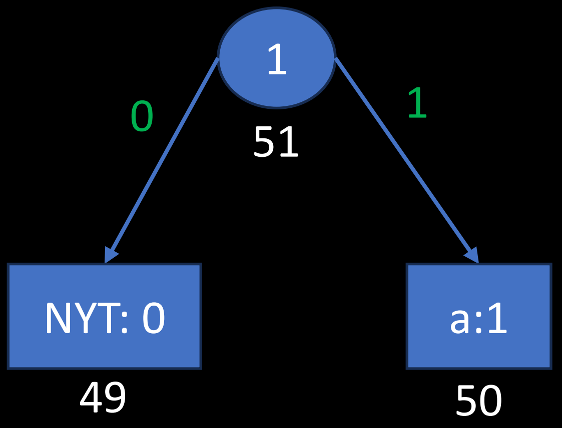
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

00000

Char	Code
NYT	0
a	1
r	
d	
v	
k	



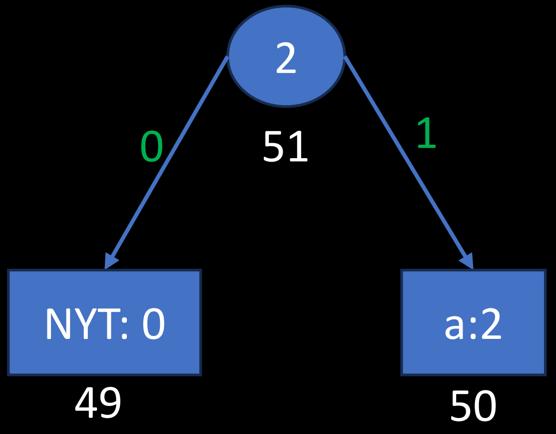
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

000001

Char	Code
NYT	0
a	1
r	
d	
v	
k	



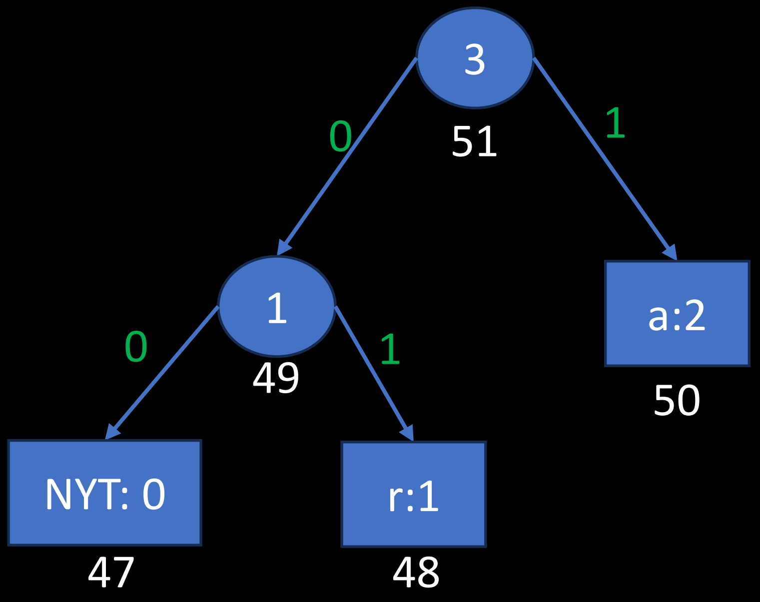
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

000001010001

Char	Code
NYT	00
a	1
r	01
d	
v	
k	



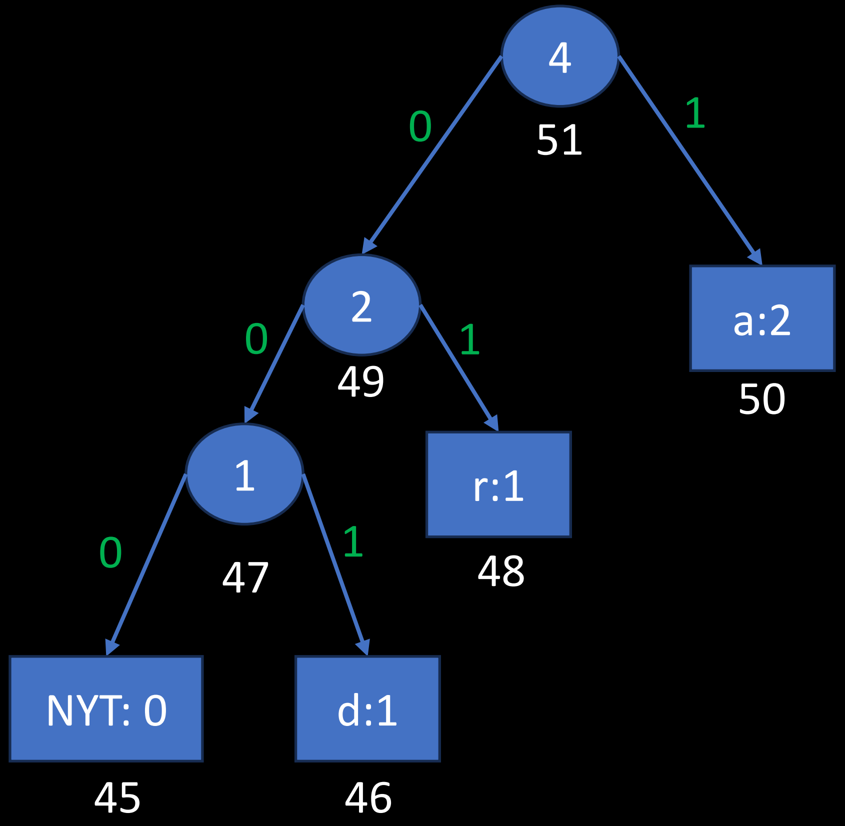
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

0000010100010000011

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	



a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

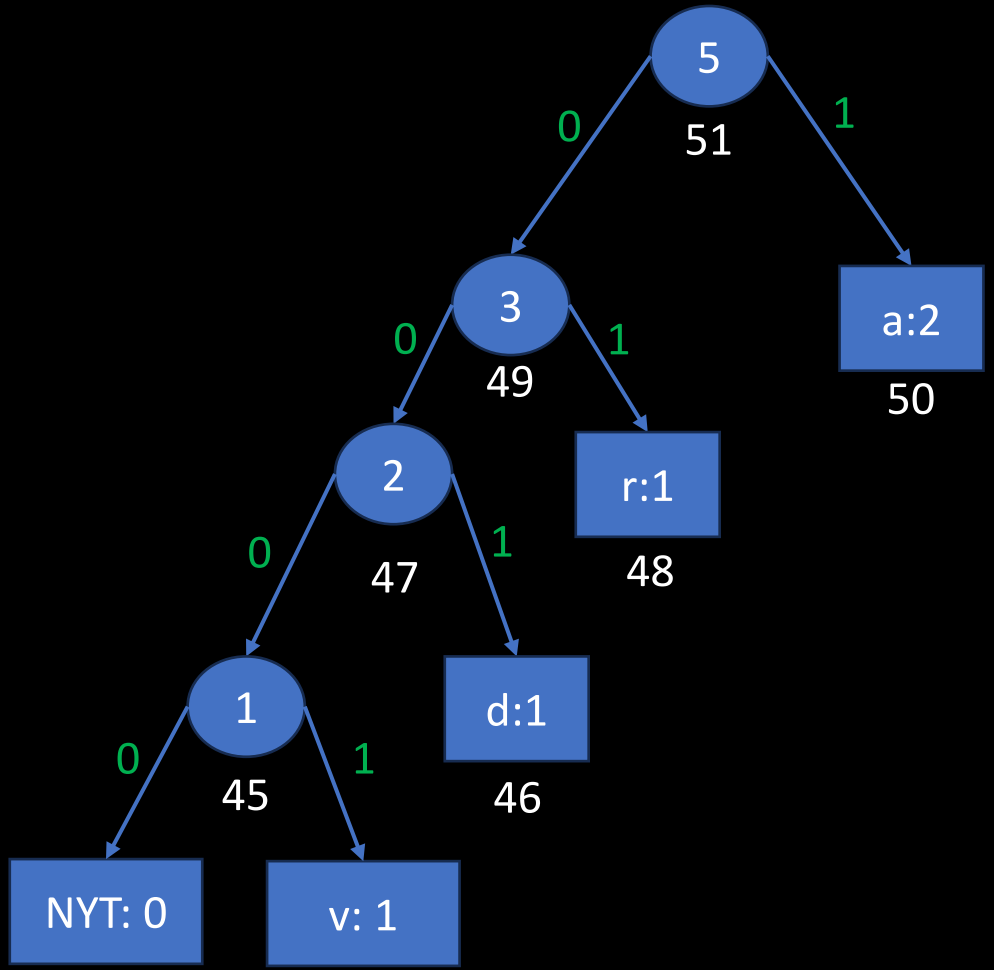
Adaptive Huffman codes

aardvark

000001010001000000011000

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	

a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000



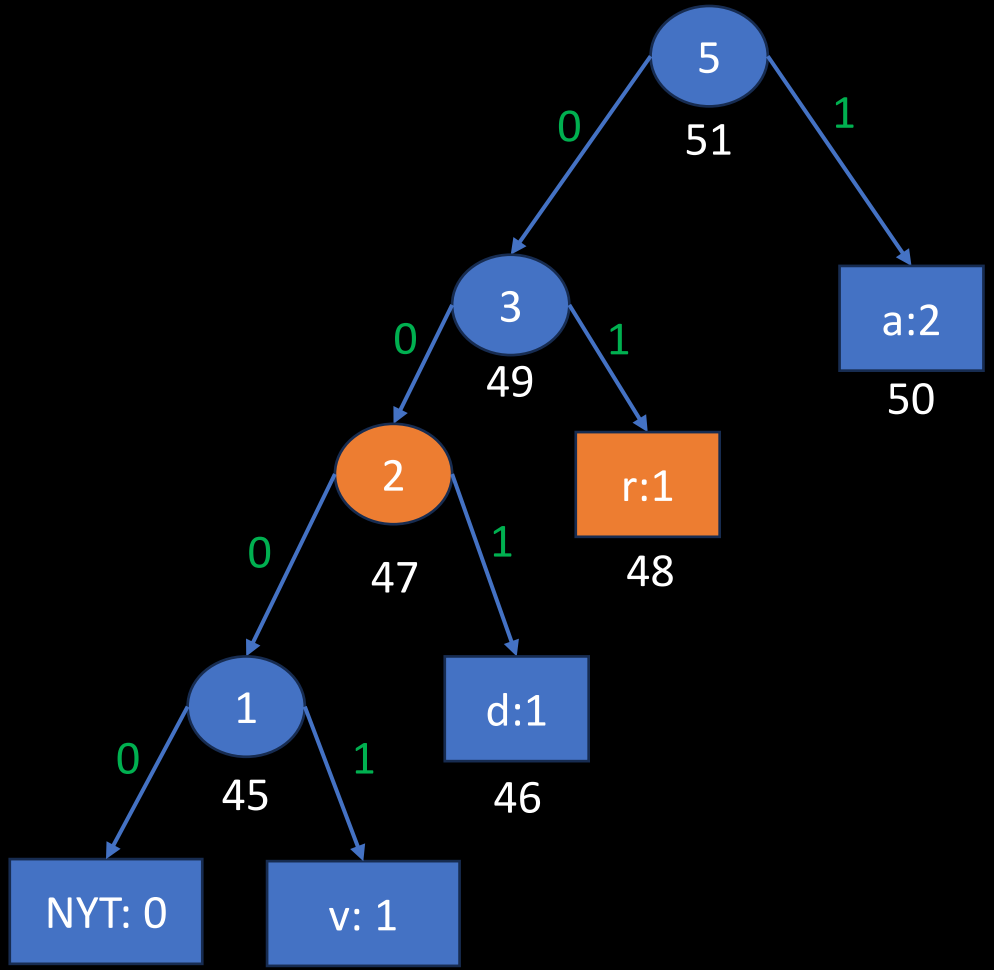
Adaptive Huffman codes

aardvark

0000010100010000011000

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	

a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

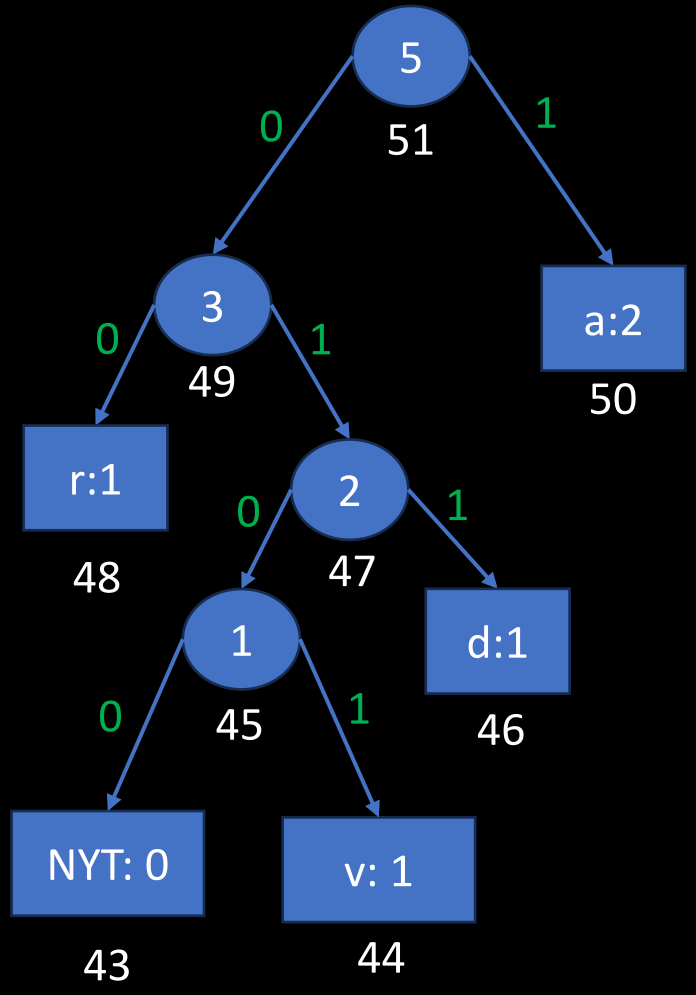


Adaptive Huffman codes

aardvark

00000101000100000011000

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	



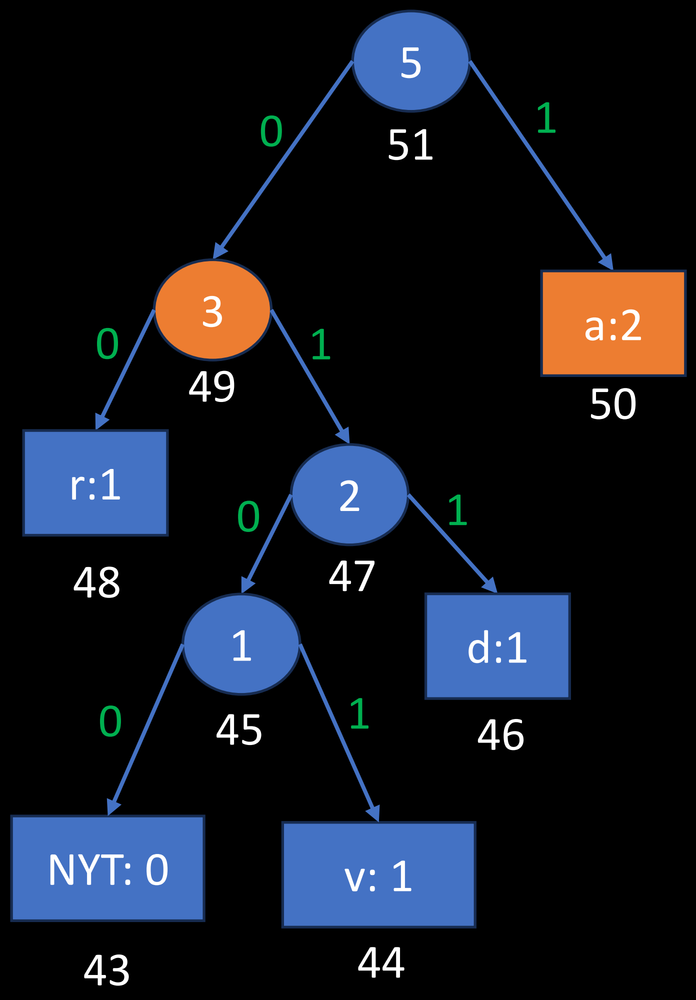
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

00000101000100000011000

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	



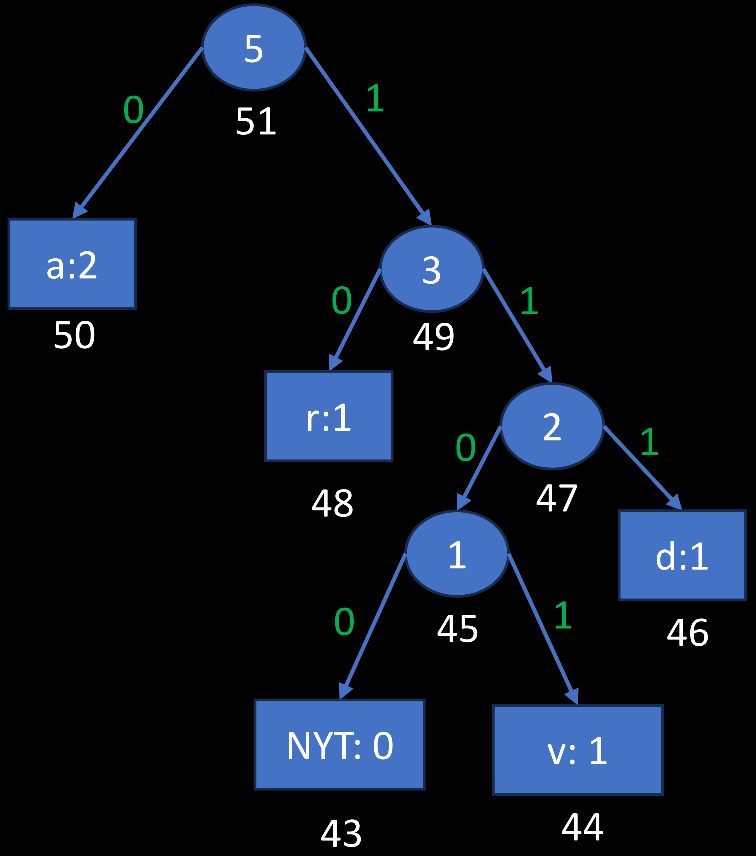
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

000001010001000000110001 0101

Char	Code
NYT	1100
a	0
r	10
d	111
v	1101
k	



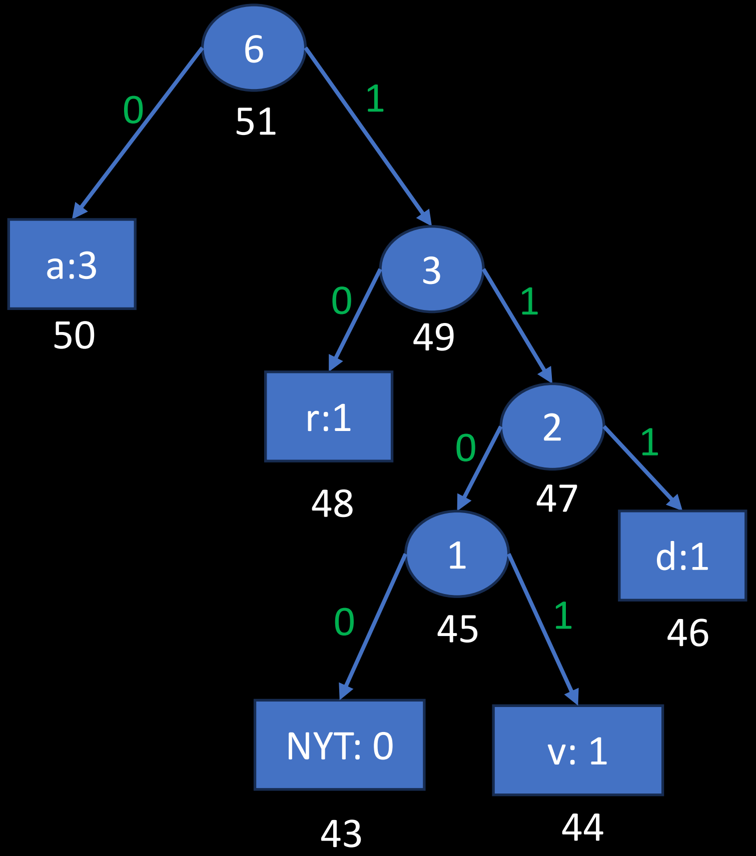
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

aardvark

00000101000100000110001011010

Char	Code
NYT	1100
a	0
r	10
d	111
v	1101
k	



a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

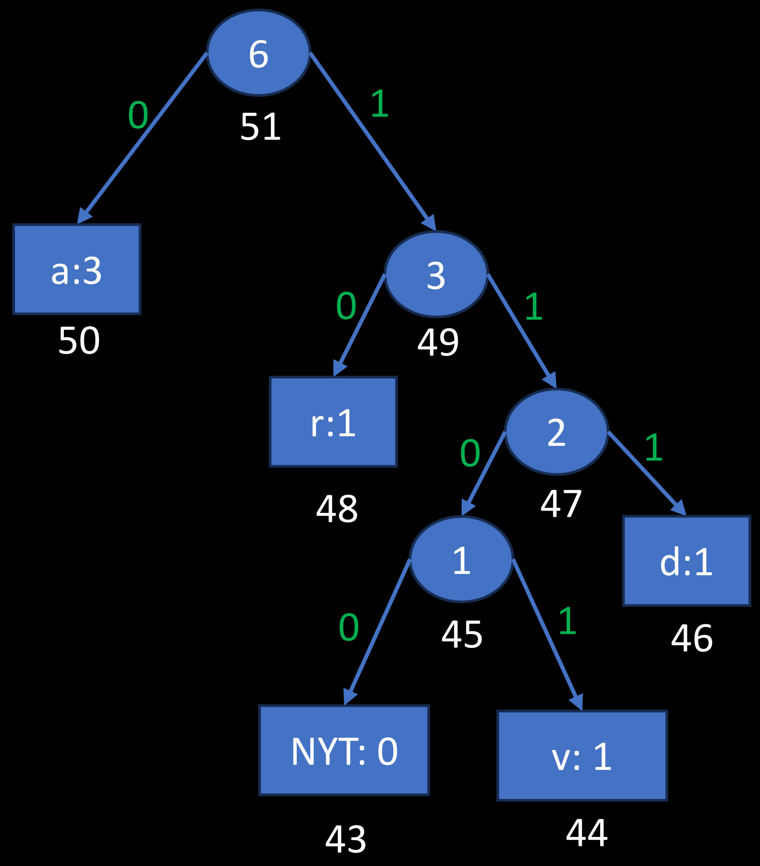
Adaptive Huffman codes

aardva<u>r</u>k

00000101000100000110001010101 0

Char	Code
NYT	1100
a	0
r	10
d	111
v	1101
k	

a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000



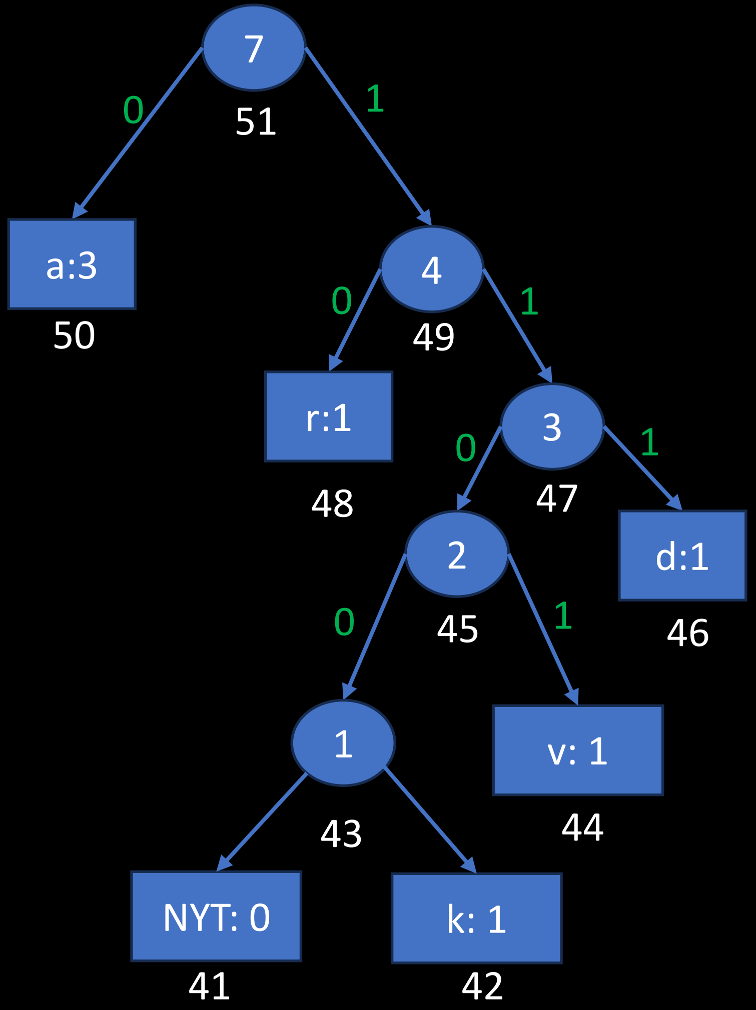
Adaptive Huffman codes

aardvark

0000010100010000011000101010101100

Char	Code
NYT	1100
a	0
r	10
d	111
v	1101
k	

a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

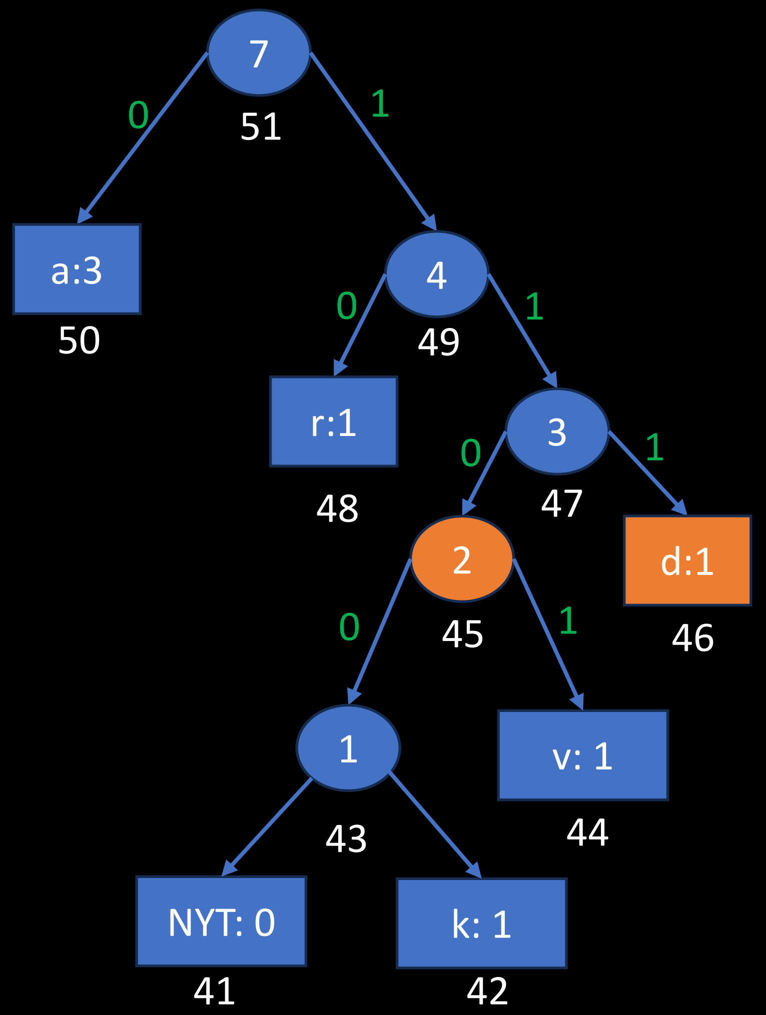


Adaptive Huffman codes

aardvark

0000010100010000011000101010101100

Char	Code
NYT	1100
a	0
r	10
d	111
v	1101
k	



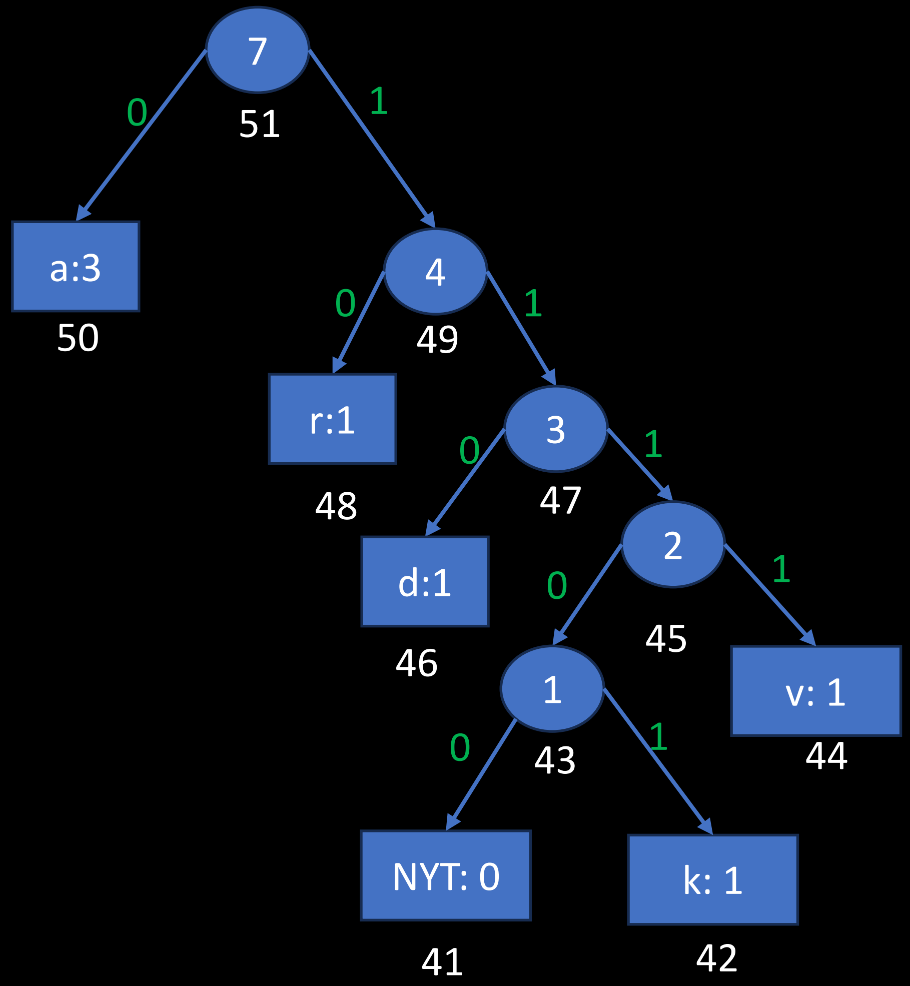
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

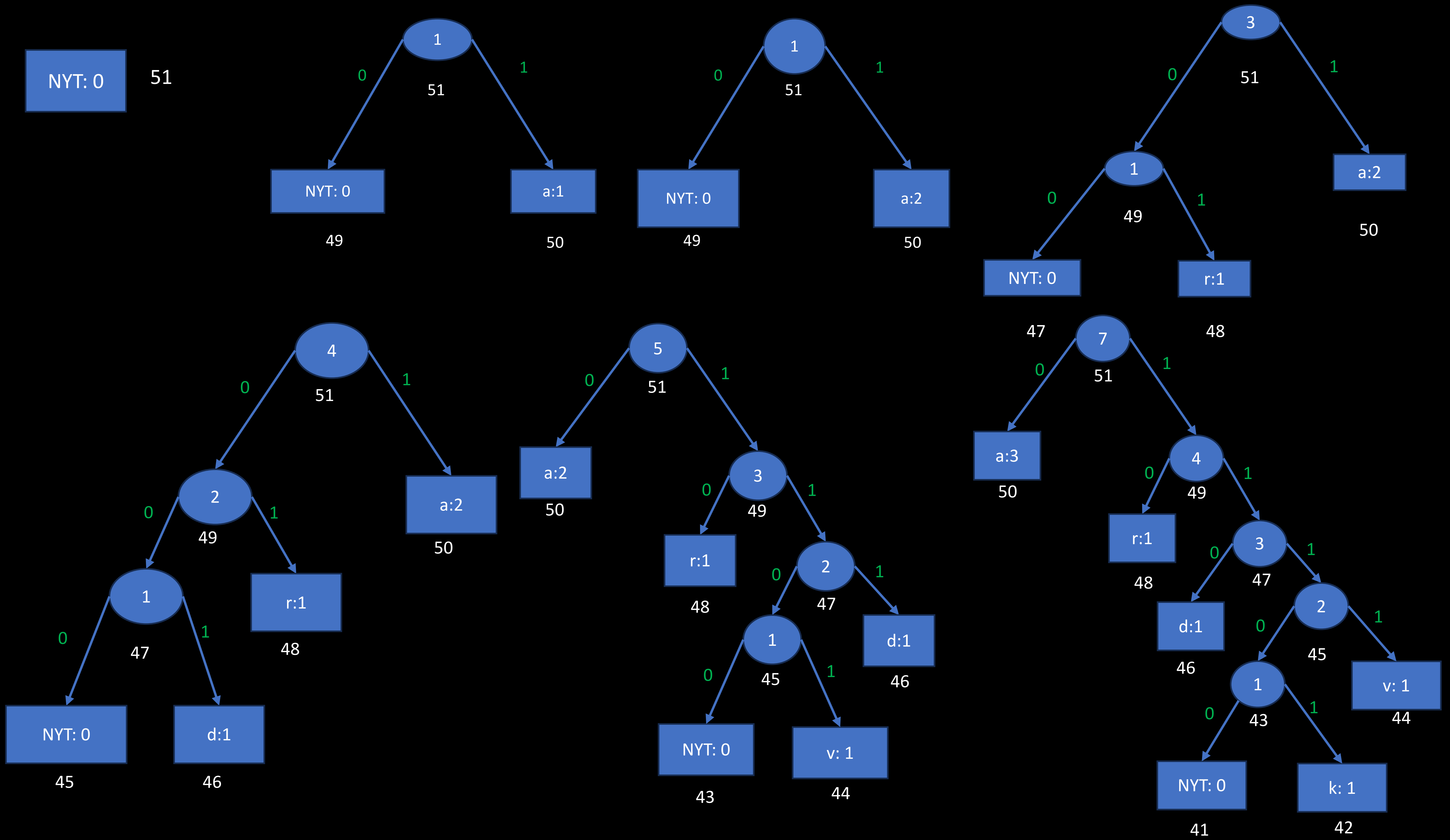
aardvark

00000101000100000110001010101010110001010

Char	Code
NYT	11100
a	0
r	10
d	110
v	1111
k	11101



a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000


NYT: 0
51
1
51
0
1
NYT: 0
49
a:1
50
1
51
0
1
NYT: 0
49
a:2
50
3
51
0
1
1
49
0
1
NYT: 0
47
r:1
48
a:2
50
4
51
0
1
2
49
0
1
1
47
0
1
NYT: 0
45
d:1
46
r:1
48
a:2
50
5
51
0
1
a:2
50
3
49
0
1
r:1
48
2
47
0
1
1
45
0
1
NYT: 0
43
v: 1
44
d:1
46
7
51
0
1
a:3
50
4
49
0
1
r:1
48
3
47
0
1
d:1
46
2
45
0
1
1
43
0
1
NYT: 0
41
k: 1
42
v: 1
44

Adaptive Huffman codes

000001010001000001100010101010110001010

NYT: 0 51

Char	Code
NYT	
a	
r	
d	
v	
k	

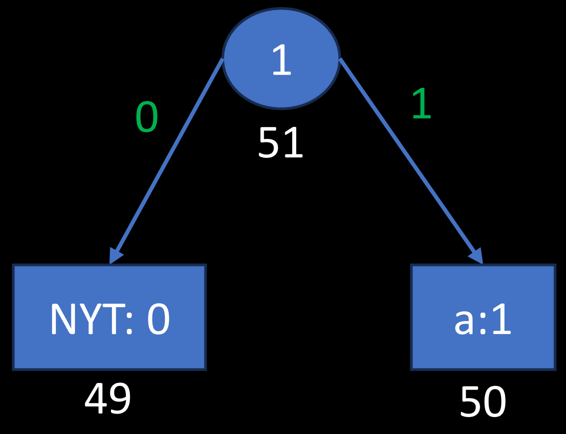
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

000001010001000001100010101010110001010

Char	Code
NYT	0
a	1
r	
d	
v	
k	

a



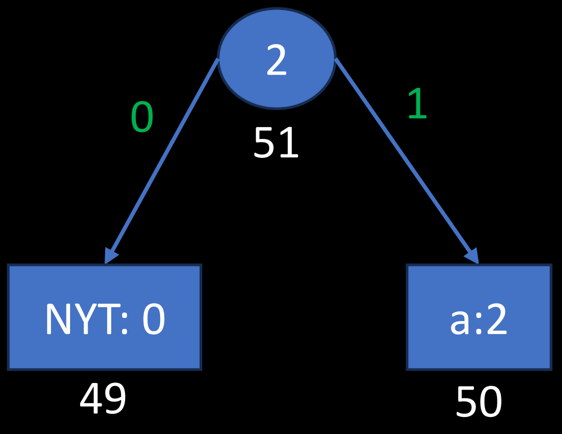
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

<u>1</u>0100010000011000101010110001010

Char	Code
NYT	<u>0</u>
a	1
r	
d	
v	
k	

aa



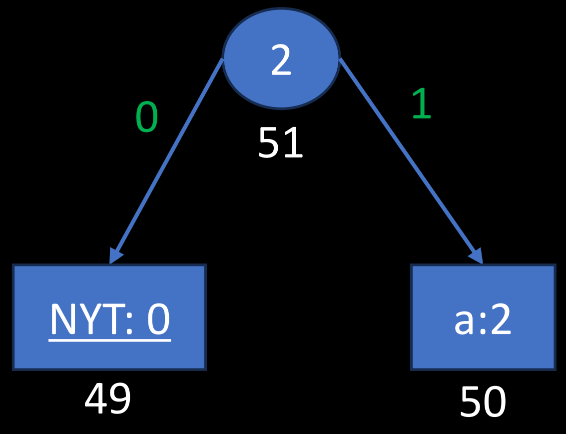
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

0100010000011000101010101100010 10

Char	Code
NYT	0
a	1
r	
d	
v	
k	

aa



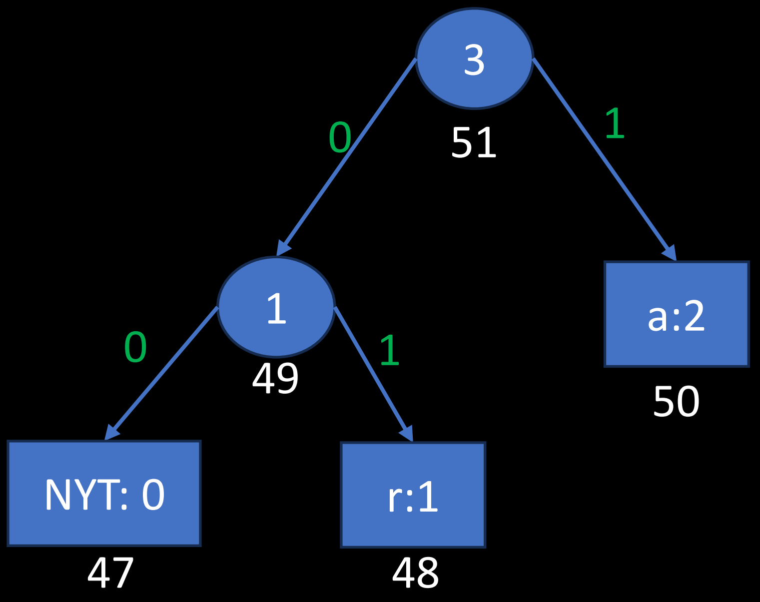
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

<u>1000</u>10000011000101010101110001010

Char	Code
NYT	<u>00</u>
a	1
r	01
d	
v	
k	

aar



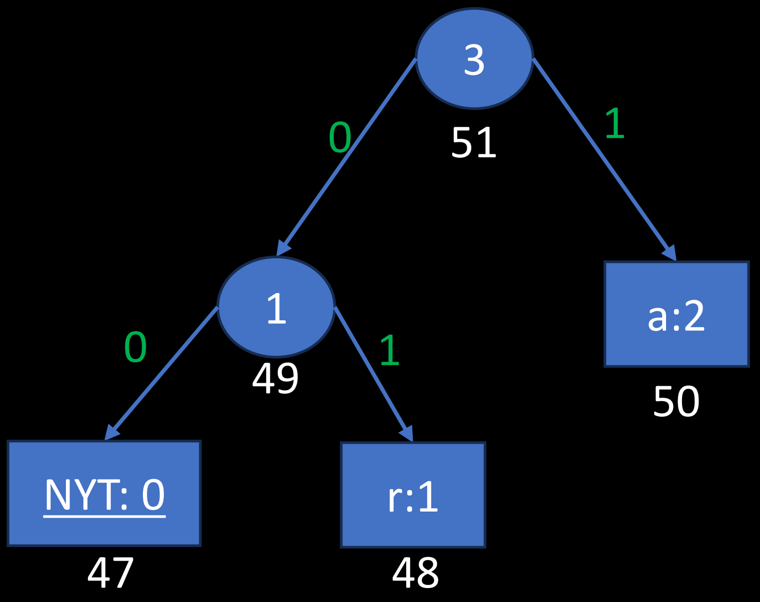
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

<u>00</u>0001100010101010110001010

Char	Code
NYT	<u>00</u>
a	1
r	01
d	
v	
k	

aar



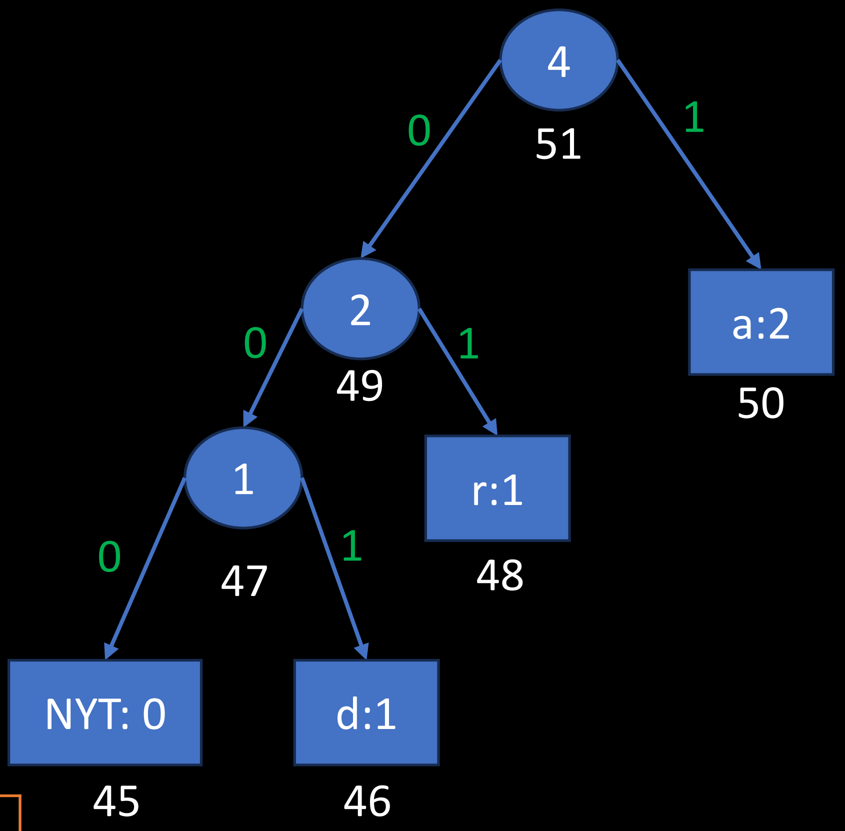
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

<u>00011</u>000101010110001010

Char	Code
NYT	<u>000</u>
a	1
r	01
d	001
v	
k	

aard



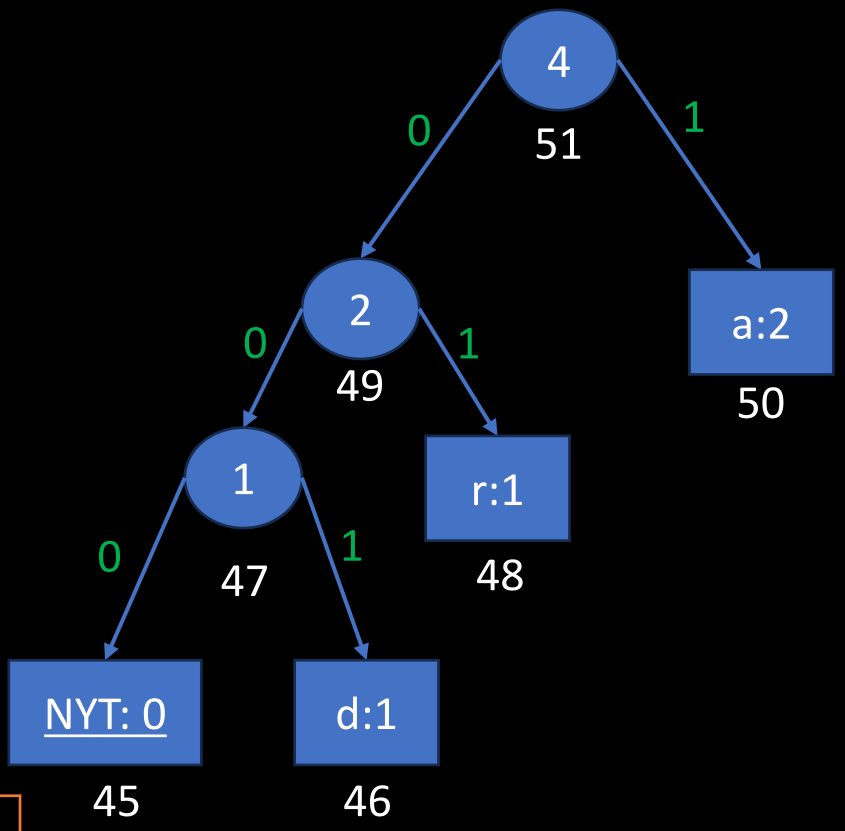
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

000101010101110001010

Char	Code
NYT	000
a	1
r	01
d	001
v	
k	

aard



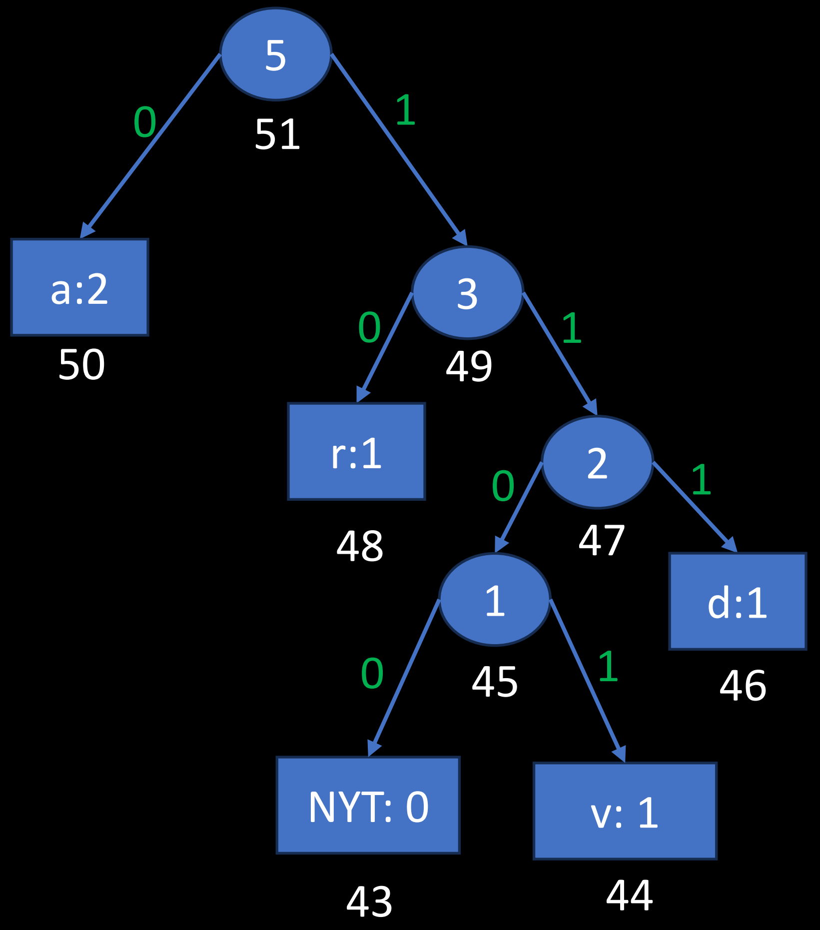
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

<u>10101</u>010110001010

Char	Code
NYT	<u>1100</u>
a	0
r	10
d	111
v	10101
k	

aardv



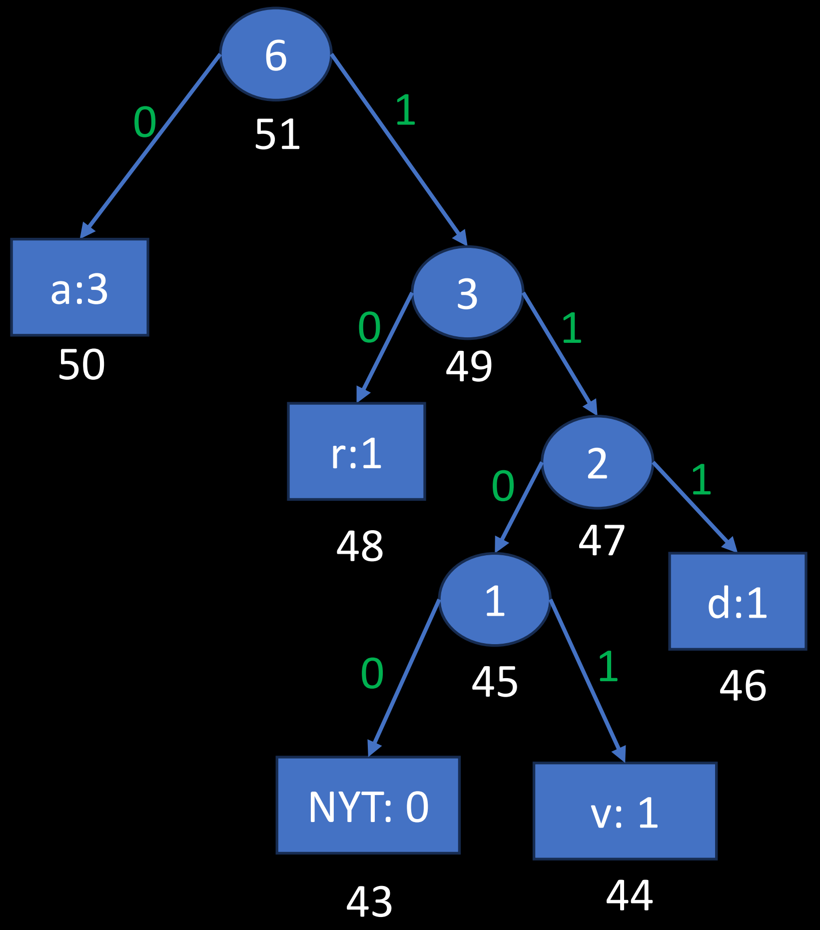
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

010110001010

Char	Code
NYT	1100
a	0
r	10
d	111
v	10101
k	

aardva



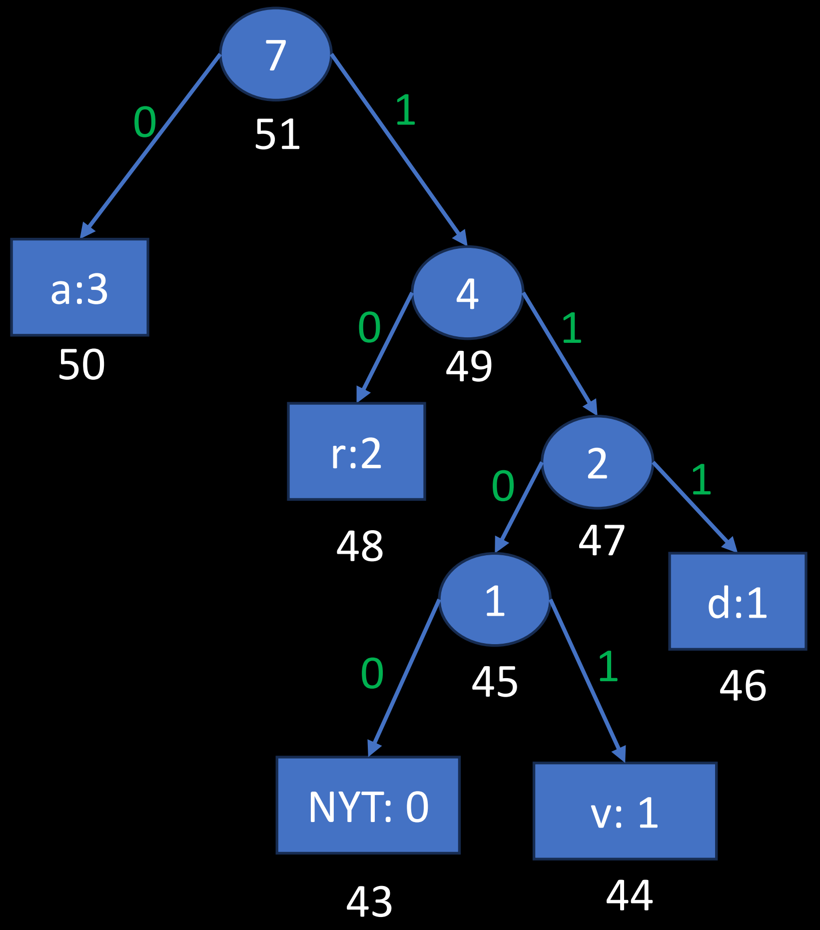
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

10110001010

Char	Code
NYT	1100
a	0
r	10
d	111
v	10101
k	

aardvar



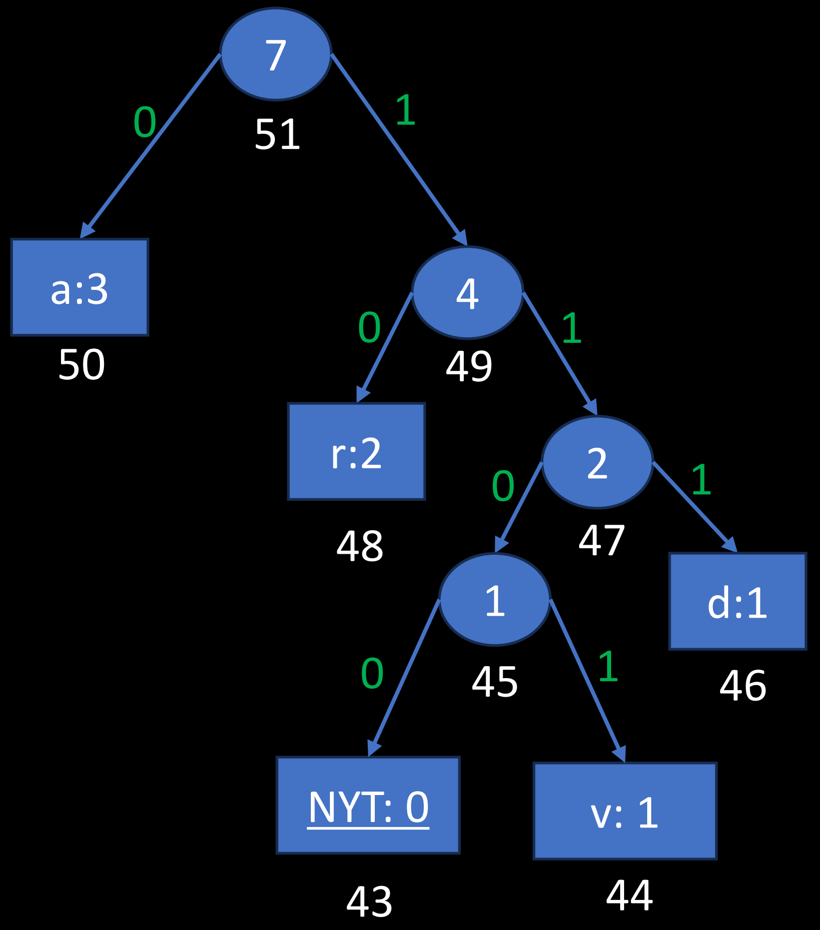
a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

Adaptive Huffman codes

110001010

Char	Code
NYT	1100
a	0
r	10
d	111
v	10101
k	

aardvar



a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000

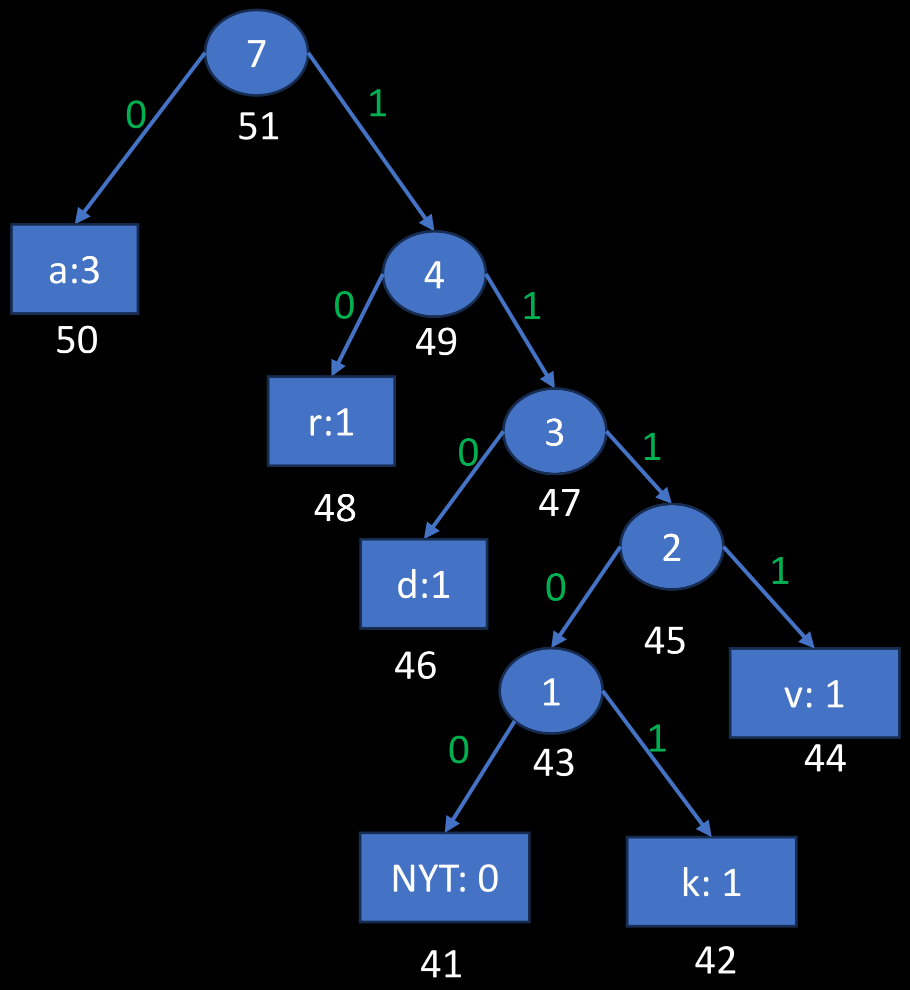
Adaptive Huffman codes

01010

aardvark

Char	Code
NYT	1100
a	0
r	10
d	111
v	10101
k	11101

a	00000	f	00101	k	01010	p	01111	u	10100
b	00001	g	00110	l	01011	q	10000	v	10101
c	00010	h	00111	m	01100	r	10001	w	10110
d	00011	i	01000	n	01101	s	10010	x	10111
e	00100	j	01001	o	01110	t	10011	y	11000



Content

Content
Greedy Algorithms
Huffman Coding
Adaptive Huffman Coding
Exercise

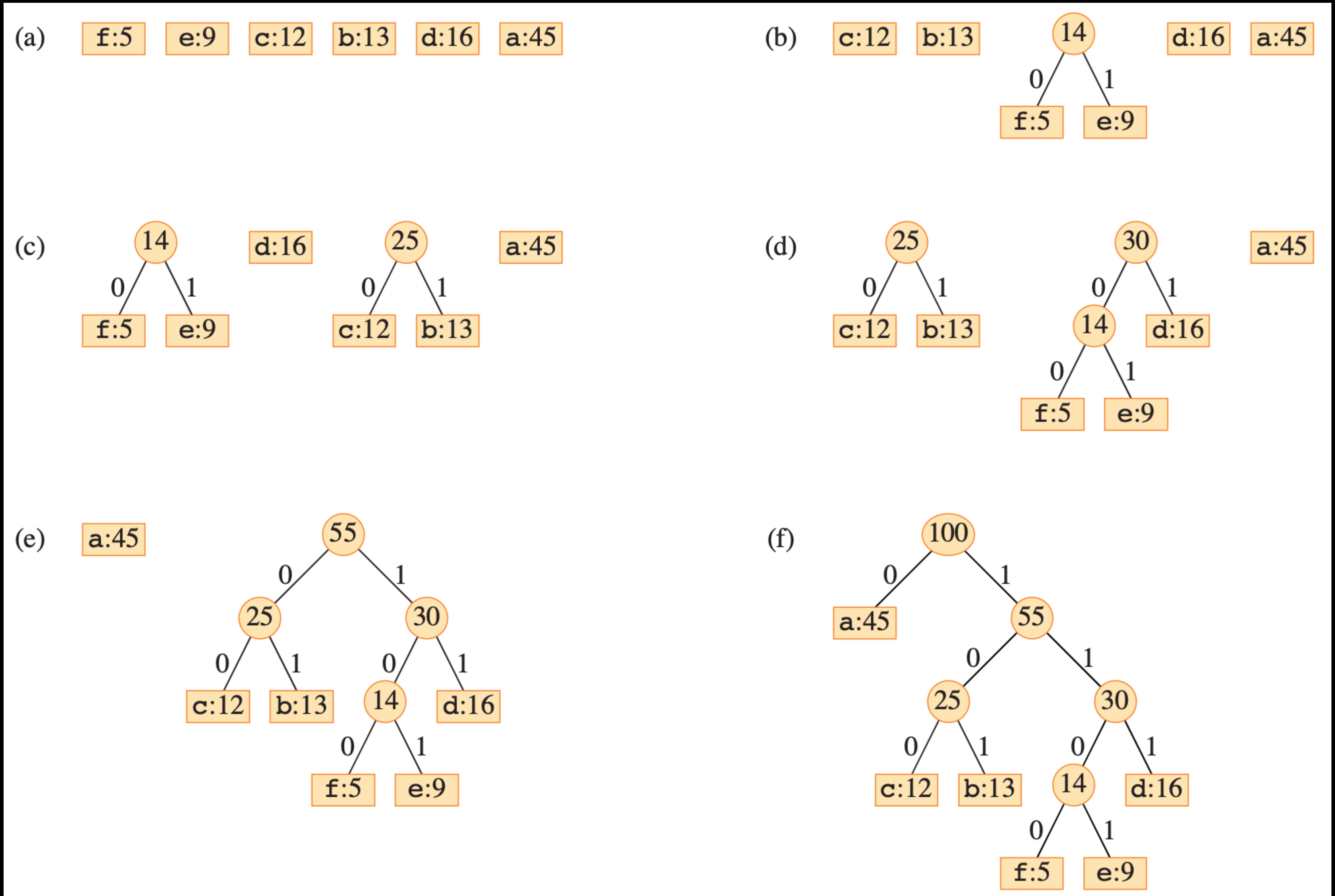
Exercise

Given the following characters with their corresponding frequency. Build the Huffman coding tree variable length code.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

Exercise

variable
length coding



References

CLRS 4th edition

Algorithms unlocked

https://people.cs.nycu.edu.tw/~cmliu/Courses/Compression/chap3.pdf