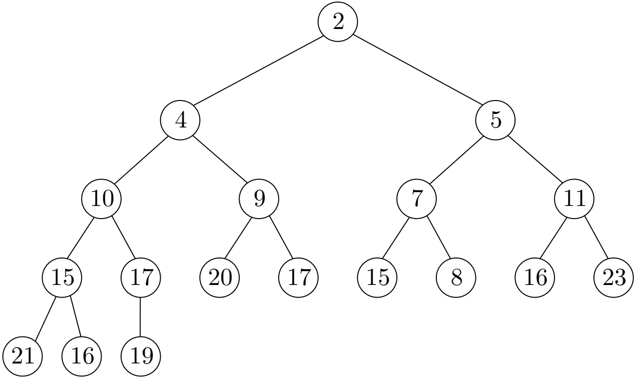
Heap

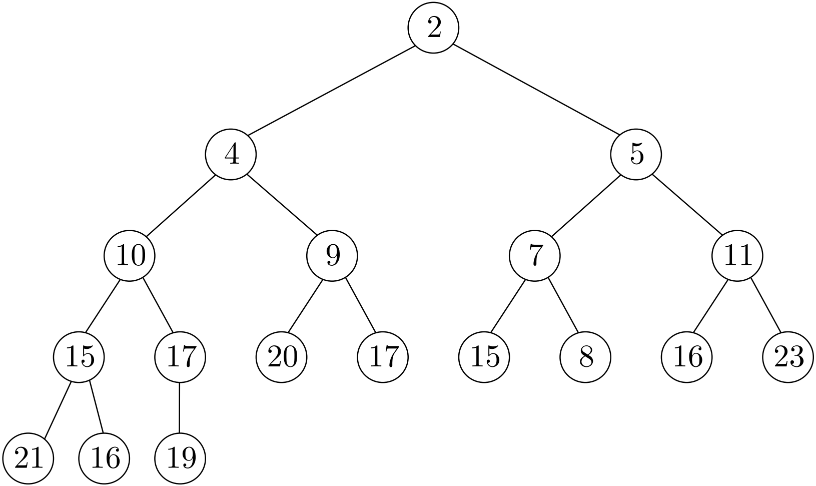
The following heap property is satisfied:

- for any two nodes i, j such that i is the parent of j, we have $key[A[i]] \leq key[A[j]]$.

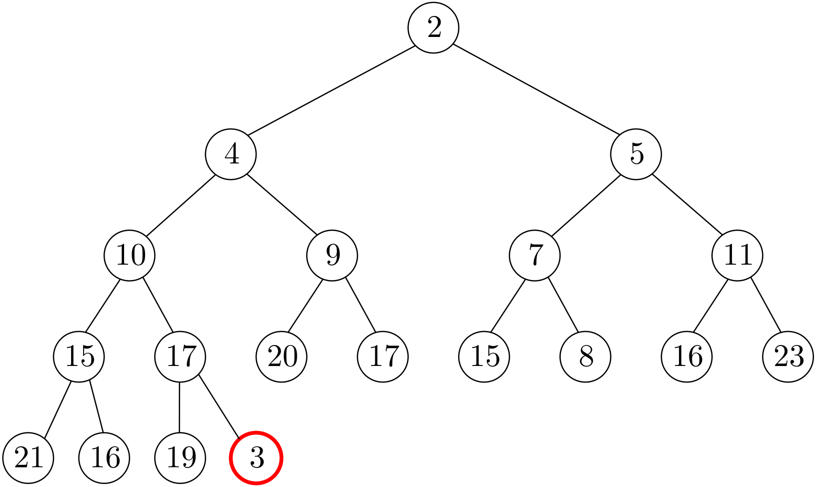


A heap. Numbers in the circles denote key values of elements.

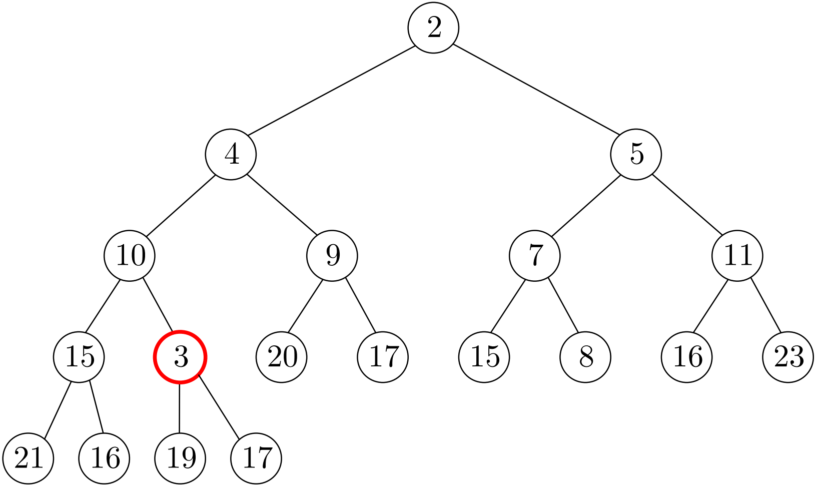
insert(v, key_value)



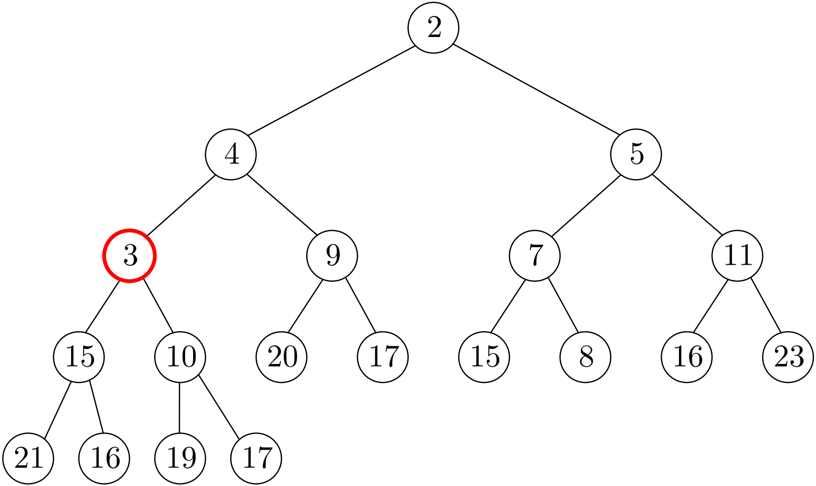
insert(v, key_value)



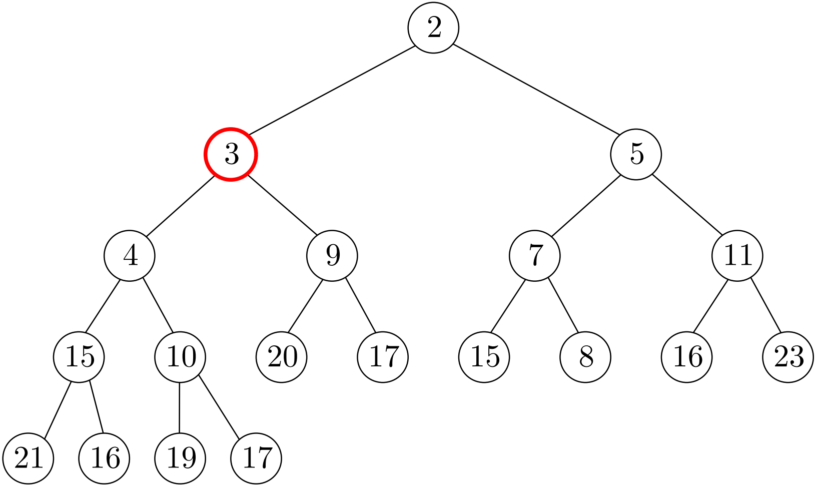
insert(v, key_value)



insert(v, key_value)



insert(v, key_value)



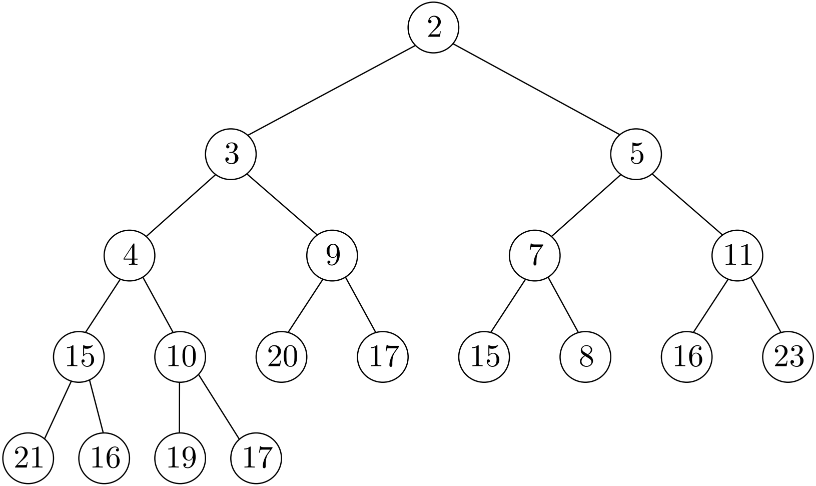
insert(v, key_value)

$s \leftarrow s + 1$
$A[s] \leftarrow v$
$p[v] \leftarrow s$
$key[v] \leftarrow key\_value$
heapify_up(s)

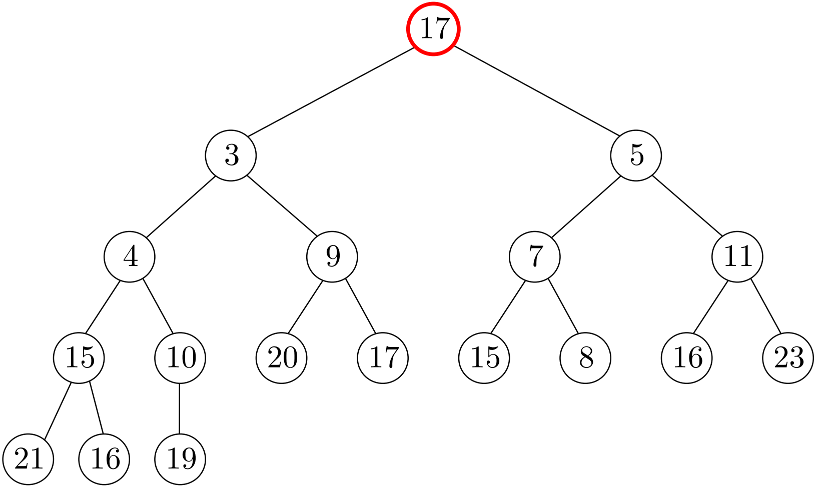
heapify-up(i)

while $i > 1$ **do**
  $j \leftarrow \lfloor i/2 \rfloor$
  if $key[A[i]] < key[A[j]]$ **then**
    swap $A[i]$ and $A[j]$
    $p[A[i]] \leftarrow i$, $p[A[j]] \leftarrow j$
    $i \leftarrow j$
  else break

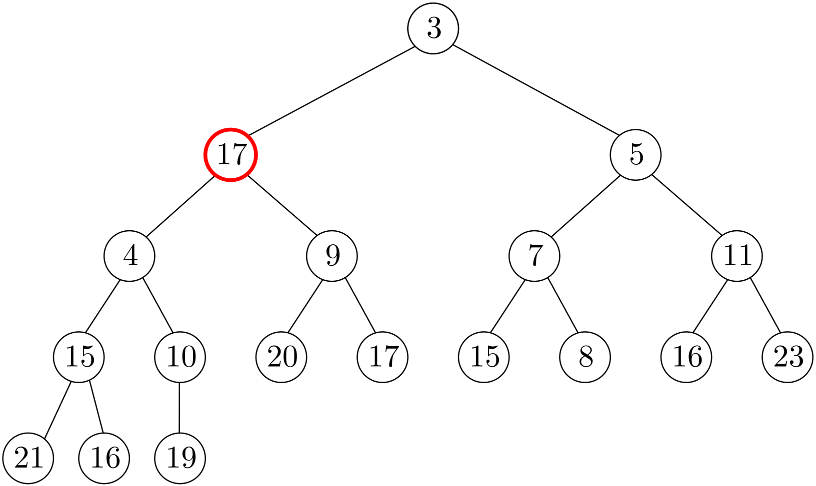
extract_min()



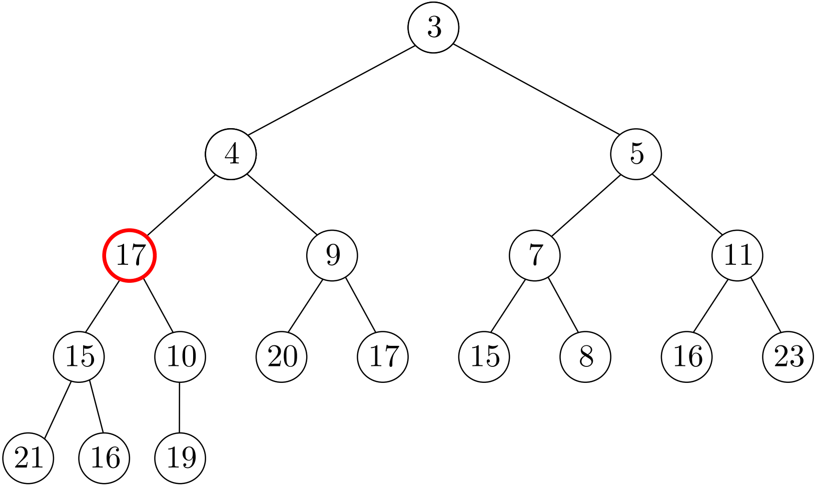
extract_min()



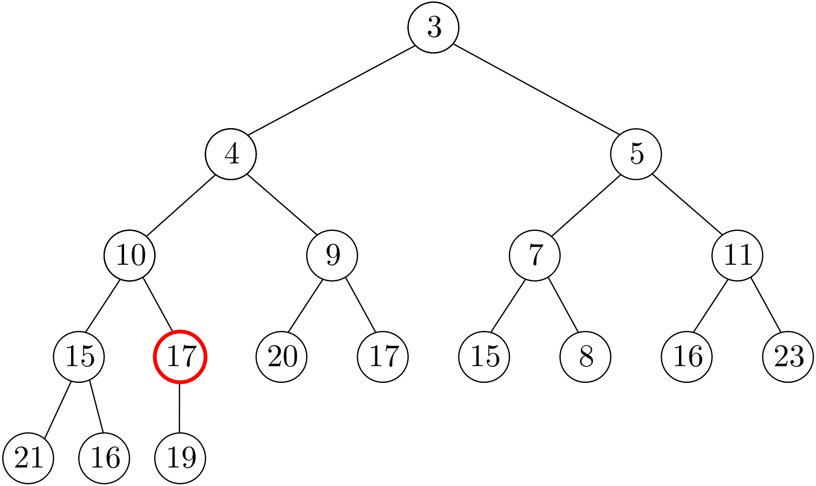
extract_min()



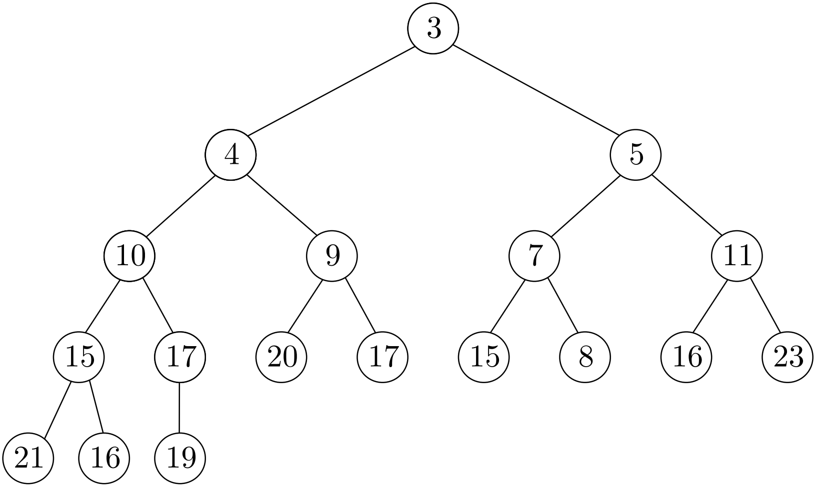
extract_min()



extract_min()



extract_min()



extract_min()

$ret \leftarrow A[1]$
$A[1] \leftarrow A[s]$
$p[A[1]] \leftarrow 1$
$s \leftarrow s - 1$
if $s \geq 1$ **then**
    heapify_down(1)
return ret

decrease_key$(v, key_val$

$key[v] \leftarrow key\_value$
heapify-up$(p[v])$

heapify-down(i)

while $2i \leq s$ **do**
    if $2i = s$ or $key[A[2i]] \leq key[A[2i+1]]$ **then**
        $j \leftarrow 2i$
    else
        $j \leftarrow 2i + 1$
    if $key[A[j]] < key[A[i]]$ **then**
        swap $A[i]$ and $A[j]$
        $p[A[i]] \leftarrow i$, $p[A[j]] \leftarrow j$
        $i \leftarrow j$
    else break

- Running time of heapify_up and heapify_down: $O(\lg n)$

- Running time of heapify_up and heapify_down: $O(\lg n)$
- Running time of insert, exact_min and decrease_key: $O(\lg n)$

- Running time of heapify_up and heapify_down: $O(\lg n)$
- Running time of insert, exact_min and decrease_key: $O(\lg n)$

data structures	insert	extract_min	decrease_key
array	$O(1)$	$O(n)$	$O(1)$
sorted array	$O(n)$	$O(1)$	$O(n)$
heap	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$

Two Definitions Needed to Prove that the Procedures Maintain Heap Property

Def. We say that H is almost a heap except that $key[A[i]]$ is too small if we can increase $key[A[i]]$ to make H a heap.

Def. We say that H is almost a heap except that $key[A[i]]$ is too big if we can decrease $key[A[i]]$ to make H a heap.

Outline

Encoding Letters Using Bits

- 8 letters a, b, c, d, e, f, g, h in a language
- need to encode a message using bits
- idea: use 3 bits per letter

a	b	c	d	e	f	g	h
000	001	010	011	100	101	110	111

$$deacfg \to 011100000010101110$$

Q: Can we have a better encoding scheme?

- Seems unlikely: must use 3 bits per letter

Q: What if some letters appear more frequently than the others?

Q: If some letters appear more frequently than the others, can we have a better encoding scheme?

A: Using variable-length encoding scheme might be more efficient.

Idea

- using fewer bits for letters that are more frequently used, and more bits for letters that are less frequently used.

Q: What is the issue with the following encoding scheme?

- a: 0 b: 1 c: 00

Q: What is the issue with the following encoding scheme?

- a: 0 b: 1 c: 00

A: Can not guarantee a unique decoding. For example, 00 can be decoded to aa or c.

Q: What is the issue with the following encoding scheme?

- a: 0 b: 1 c: 00

A: Can not guarantee a unique decoding. For example, 00 can be decoded to aa or c.

Solution

Use prefix codes to guarantee a unique decoding.

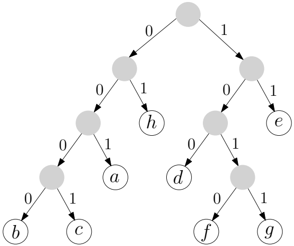
Prefix Codes

Def. A prefix code for a set S of letters is a function $\gamma : S \to \{0,1\}^*$ such that for two distinct $x, y \in S$, $\gamma(x)$ is not a prefix of $\gamma(y)$.

Prefix Codes

Def. A prefix code for a set S of letters is a function $\gamma : S \to \{0,1\}^*$ such that for two distinct $x, y \in S$, $\gamma(x)$ is not a prefix of $\gamma(y)$.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01



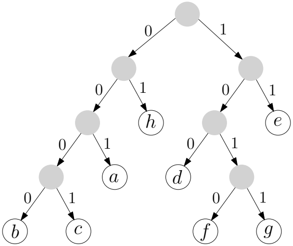
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

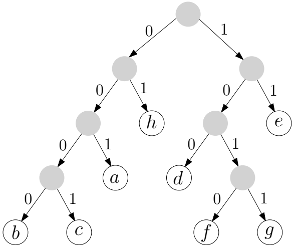
a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01



Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

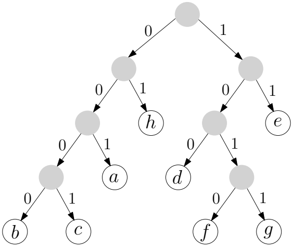


- 0001001100000001011110100001001

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

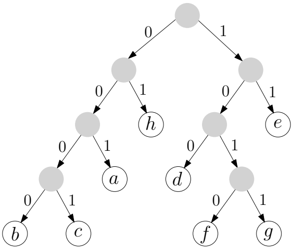


- 0001/001100000001011110100001001
- c

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

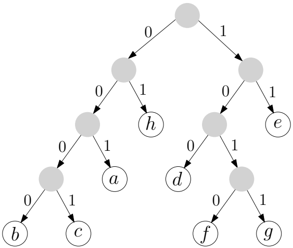


- 0001/001/100000001011110100001001
- ca

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

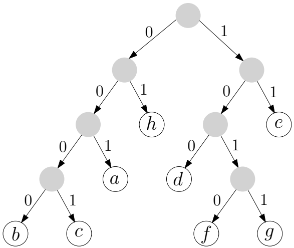


- 0001/001/100/0000010111110100001001
- cad

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

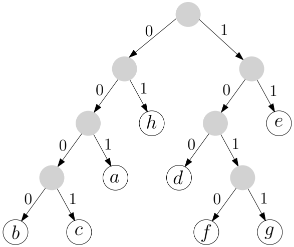


- 0001/001/100/0000/010111101000001001
- cadb

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

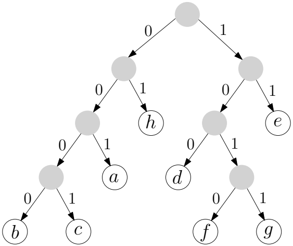


- 0001/001/100/0000/01/011110100001001
- cadbh

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

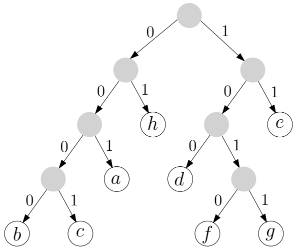


- 0001/001/100/0000/01/01/1110100001001
- cadbhh

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

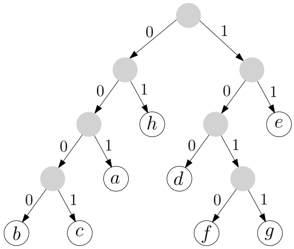


- 0001/001/100/0000/01/01/11/10100001001
- cadbhhe

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

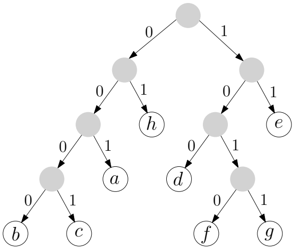


- 0001/001/100/0000/01/01/11/1010/0001001
- cadbhhef

Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01

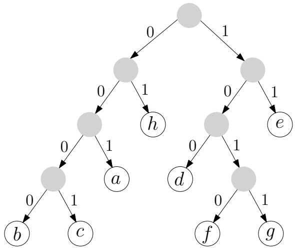


- 0001/001/100/0000/01/01/11/1010/0001/001
- cadbhhefc

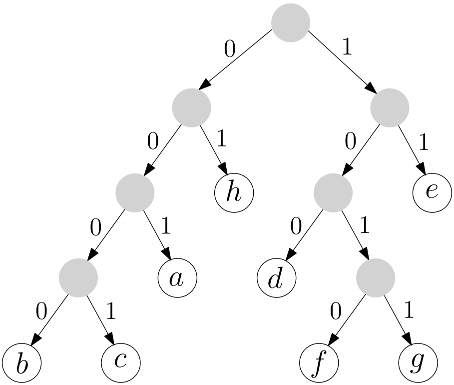
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

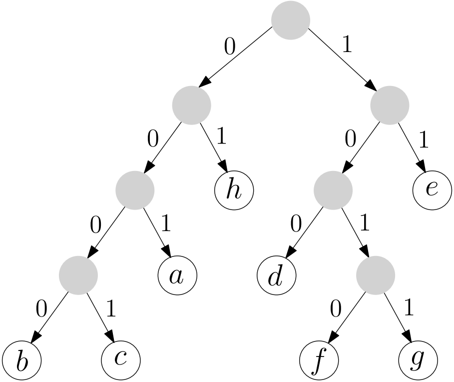
a	b	c	d
001	0000	0001	100
e	f	g	h
11	1010	1011	01



- 0001/001/100/0000/01/01/11/1010/0001/001/
- cadbhhefca

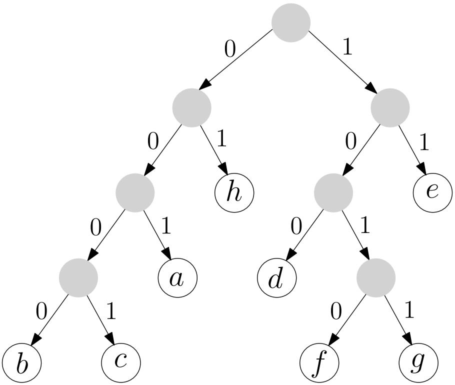


Properties of Encoding Tree



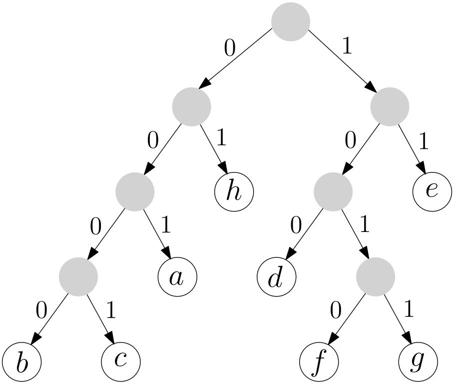
Properties of Encoding Tree

- Rooted binary tree



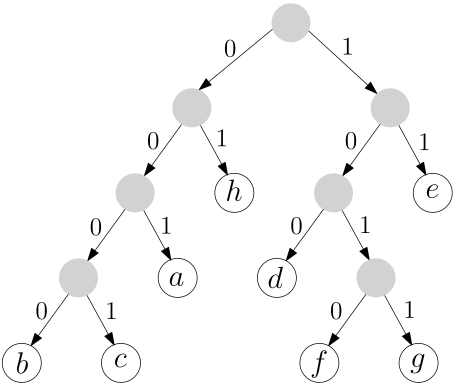
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1



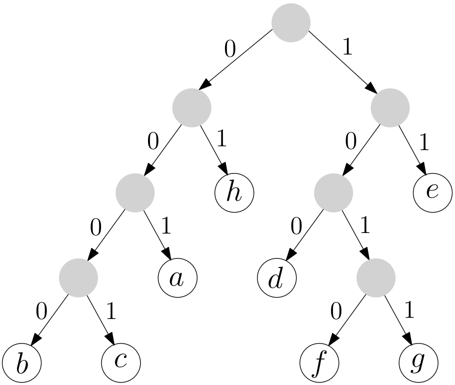
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter



Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter
- If coding scheme is not wasteful: a non-leaf has exactly two children



Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter
- If coding scheme is not wasteful: a non-leaf has exactly two children

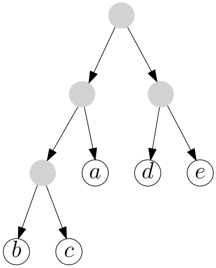
Best Prefix Codes

Input: frequencies of letters in a message

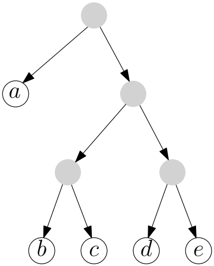
Output: prefix coding scheme with the shortest encoding for the message

example

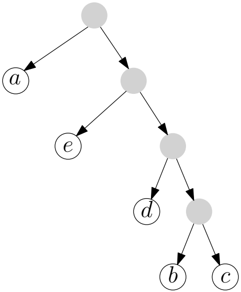
letters	a	b	c	d	e	
frequencies	18	3	4	6	10	



scheme 1



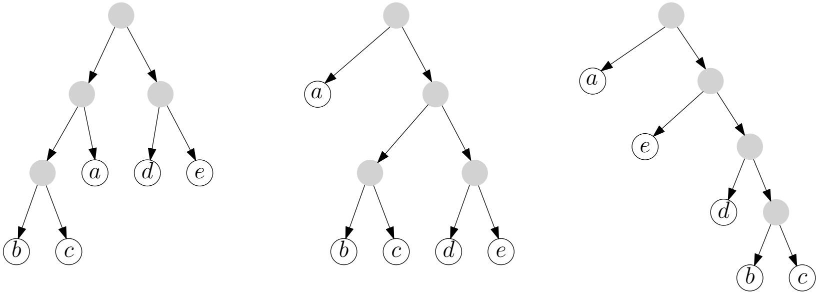
scheme 2



scheme 3

example

letters	a	b	c	d	e	
frequencies	18	3	4	6	10	
scheme 1 length	2	3	3	2	2	total = 89
scheme 2 length	1	3	3	3	3	total = 87
scheme 3 length	1	4	4	3	2	total = 84



scheme 1 scheme 2 scheme 3