COL 351 : Analysis & Design of Algorithms

Lecture 17

Greedy Algorithms III : Huffman Coding (Contd.)

Sept 03, 2024 | Rohit Vaish

BINARY CODES

Alphabet Σ : finite nonempty set of symbols

e.g., $\Sigma = \{A, B, \ldots, Z\}$, $\Sigma = \{\bullet, *\}$, etc.

BINARY CODES

Alphabet Σ : finite nonempty set of symbols

e.g., $\Sigma = \{A, B, \ldots, Z\}$, $\Sigma = \{\bullet, *\}$, etc.

Binary code : maps each character of an alphabet to a binary string

BINARY CODES

Alphabet Σ : finite nonempty set of symbols

e.g., $\Sigma = \{A, B, \ldots, Z\}$, $\Sigma = \{\bullet, *\}$, etc.

Binary code : maps each character of an alphabet to a binary string

e.g., A = 00000

B = 00001

C = 00010

⋮

fixed length binary codes

VARIABLE LENGTH CODES

When some symbols of the alphabet occur more frequently than others

VARIABLE LENGTH CODES

When some symbols of the alphabet occur more frequently than others

e.g.,

	fixed length		variable length
A	00	A	0
B	01	B	01
C	10	C	10
D	11	D	1

VARIABLE LENGTH CODES

When some symbols of the alphabet occur more frequently than others

e.g.,

	fixed length		variable length	
A	00	A	0	Ambiguous ☹
B	01	B	01	How to interpret 001?
C	10	C	10	AB ? AAD ?
D	11	D	1	

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 1 0 1 1 0 0 1 1 0

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B C

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B C A

PREFIX-FREE CODES

For each pair of distinct symbols $i, j \in \Sigma$,

the encoding of i is not a prefix of j and vice versa

e.g., fixed length codes are prefix-free

variable length

A	0
B	01
C	10
D	1

not prefix-free

variable length

A	0
B	10
C	110
D	111

prefix-free

Unambiguous decoding

e.g., 10 110 0 110

B C A C

EFFICIENCY OF PREFIX-FREE CODES

EFFICIENCY OF PREFIX-FREE CODES

symbol	frequency	fixed length	variable length (prefix-free)
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

EFFICIENCY OF PREFIX-FREE CODES

symbol	frequency	fixed length	variable length (prefix-free)
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111
On average:		2 bits/symbol	1.55 bits/symbol

more efficient

$= 0.6 \times 1 + 0.25 \times 2 + 0.15 \times 3$

EFFICIENCY OF PREFIX-FREE CODES

symbol	frequency	fixed length	variable length (prefix-free)
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111
On average :		2 bits/symbol	1.55 bits/symbol (more efficient)

$= 0.6 \times 1 + 0.25 \times 2 + 0.15 \times 3$

How to compute an optimal prefix-free code ?

Optimal Prefix-free codes

Optimal Prefix-free codes

input : a non negative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

Optimal Prefix-free codes

input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output: A prefix-free binary code with minimum possible average encoding length

$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

Optimal Prefix-free codes

input : a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output : A prefix-free binary code with minimum possible average encoding length

$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

brute force : ?

Optimal Prefix-free codes

input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output: A prefix-free binary code with minimum possible average encoding length

$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

brute force: at least $n!$

n symbols encoded as

0, 10, 110, 1110, ...

CODES AS TREES

CODES AS TREES

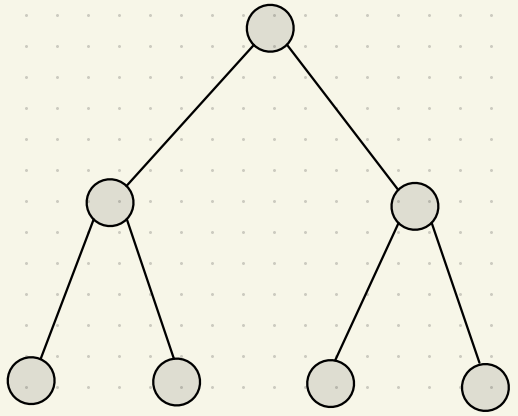
$\Sigma = \{A, B, C, D\}$

symbol	fixed length
A	00
B	01
C	10
D	11

CODES AS TREES

$\Sigma = \{A, B, C, D\}$

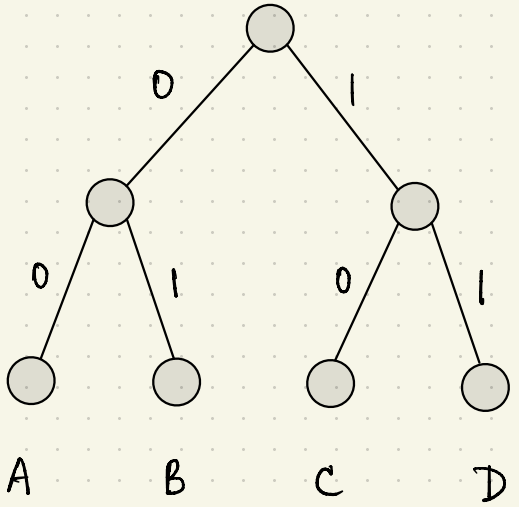
symbol	fixed length
A	00
B	01
C	10
D	11



CODES AS TREES

$\Sigma = \{A, B, C, D\}$

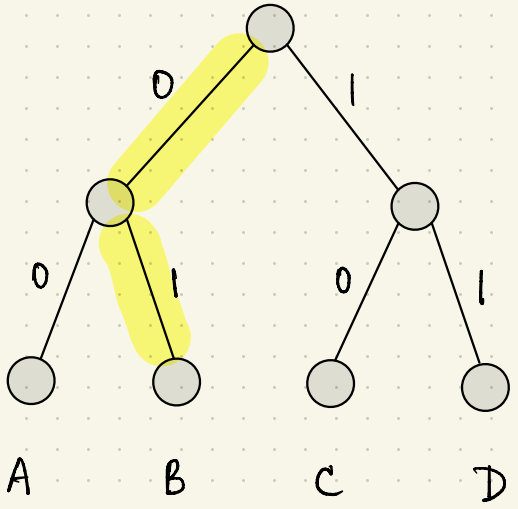
symbol	fixed length
A	00
B	01
C	10
D	11



CODES AS TREES

$\Sigma = \{A, B, C, D\}$

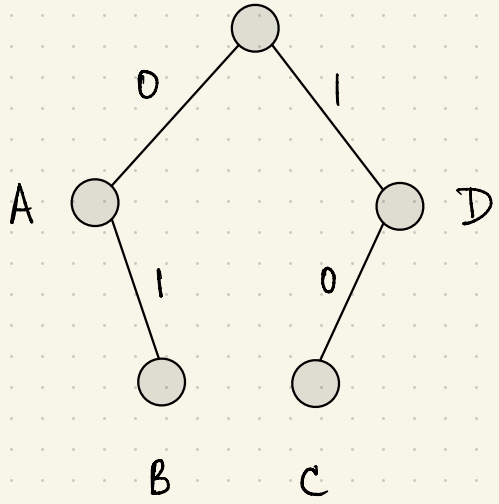
symbol	fixed length
A	00
B	01
C	10
D	11



CODES AS TREES

$\Sigma = \{A, B, C, D\}$

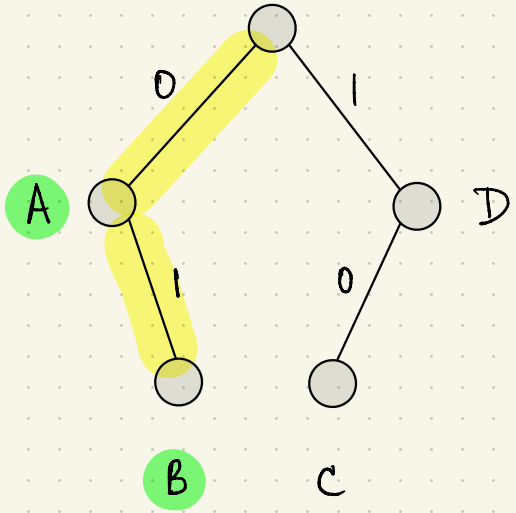
Symbol	variable length
A	0
B	01
C	10
D	1



CODES AS TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1

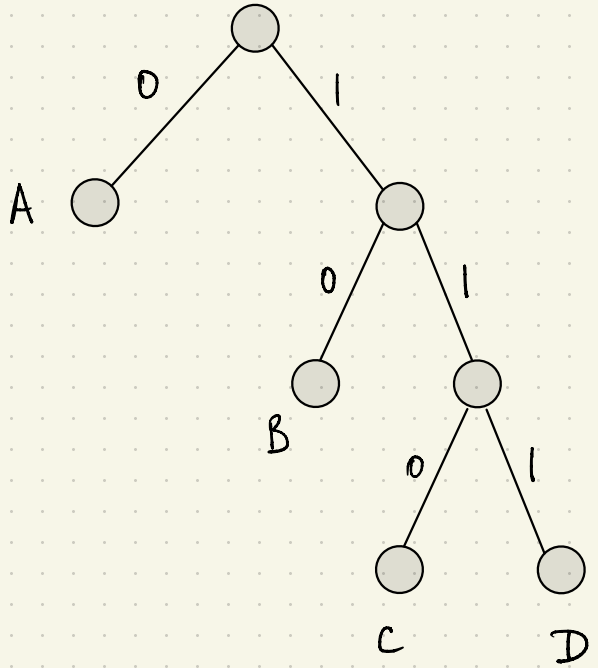


not prefix-free

CODES AS TREES

$\Sigma = \{A, B, C, D\}$

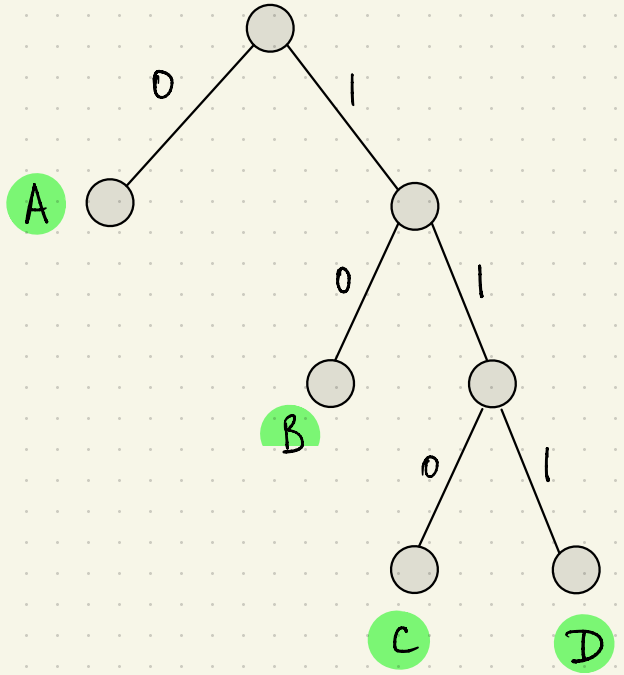
Symbol	variable length
A	0
B	10
C	110
D	111



CODES AS TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	10
C	110
D	111



prefix-free

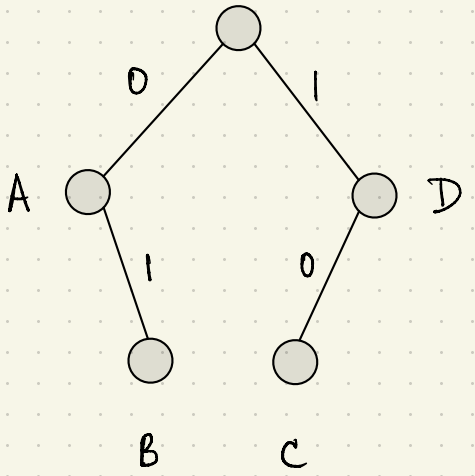
BINARY CODES ⟷ LABELED BINARY TREES

BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1

⟷



BINARY CODES ⟷ LABELED BINARY TREES

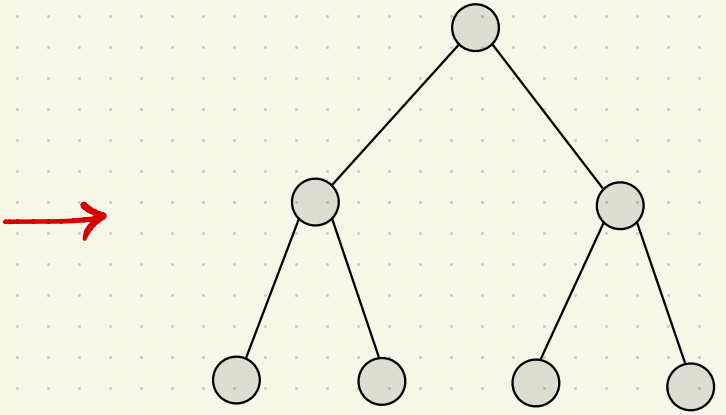
$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1

BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1

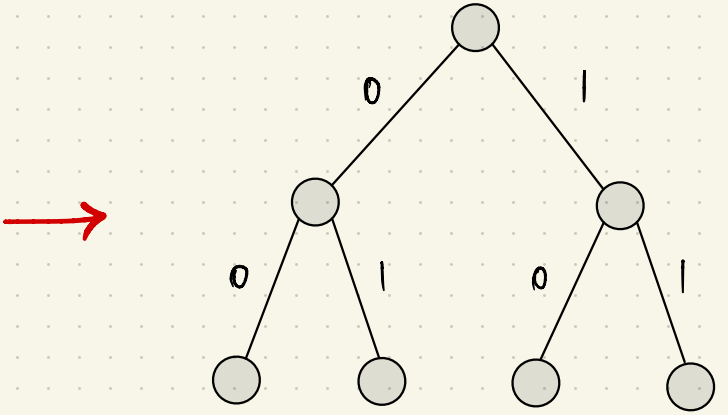


Depth = max # bits used for any symbol

BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1



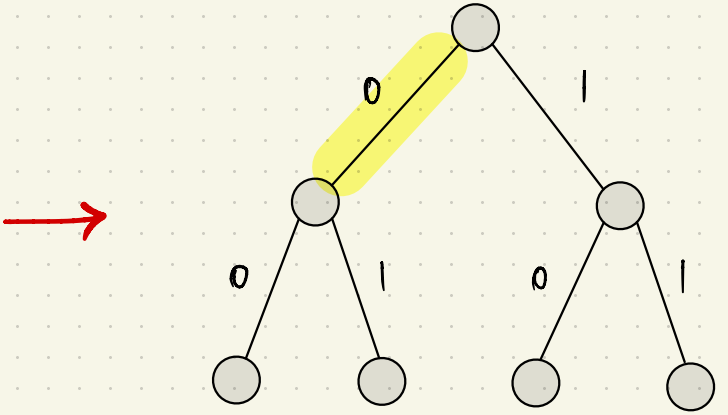
Left child : 0

Right child : 1

BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

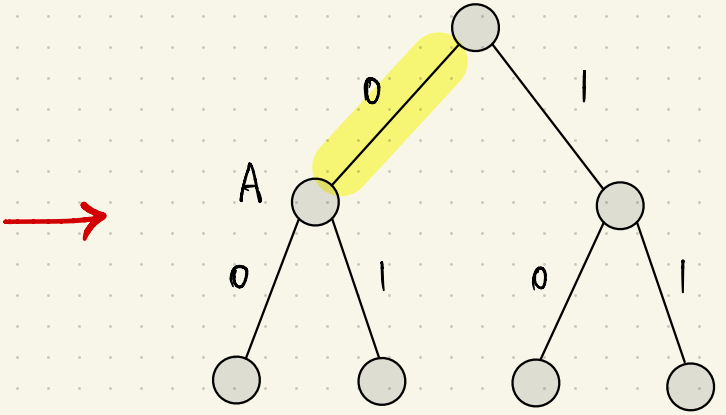
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

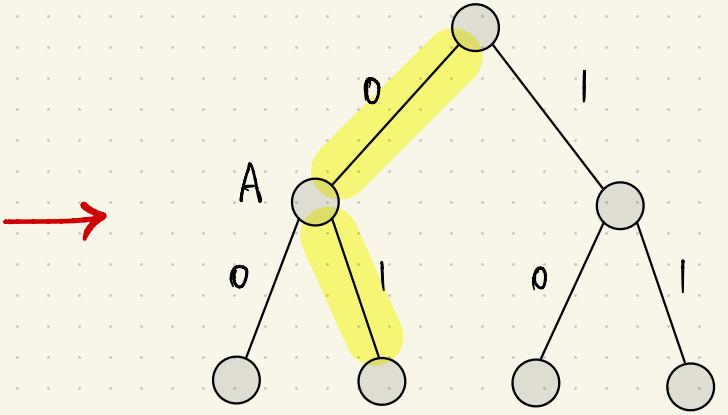
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

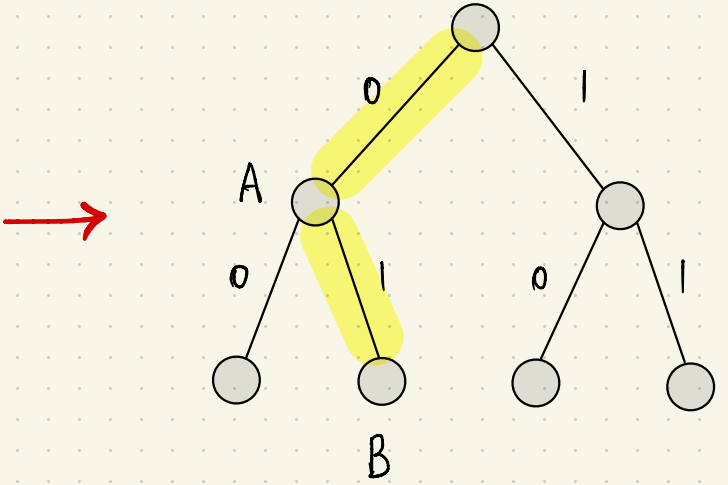
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

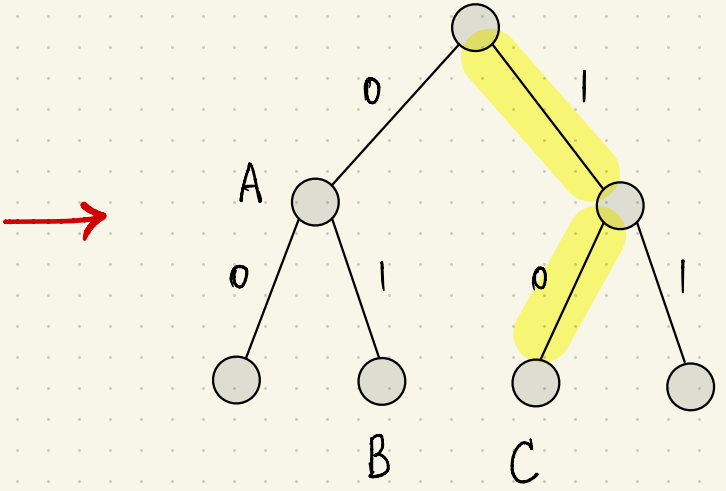
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

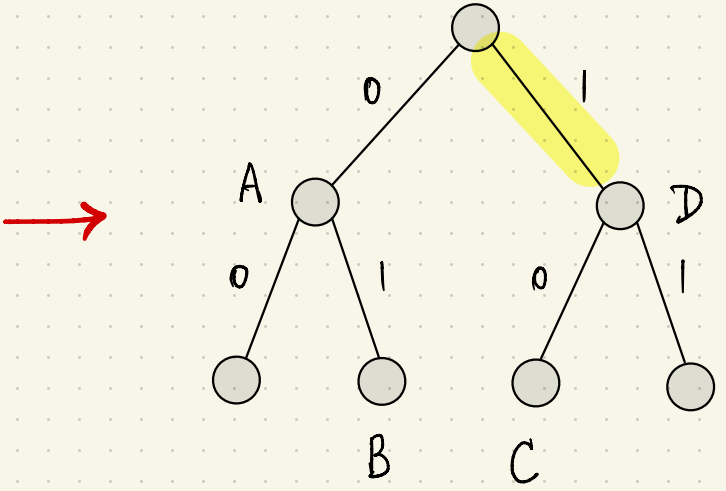
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

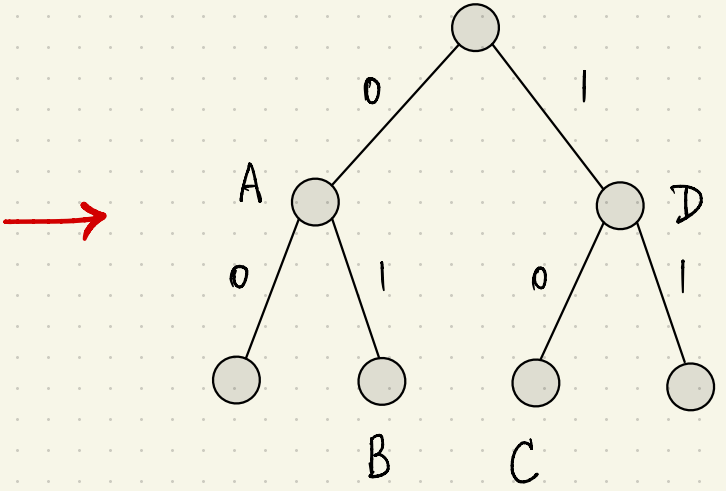
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

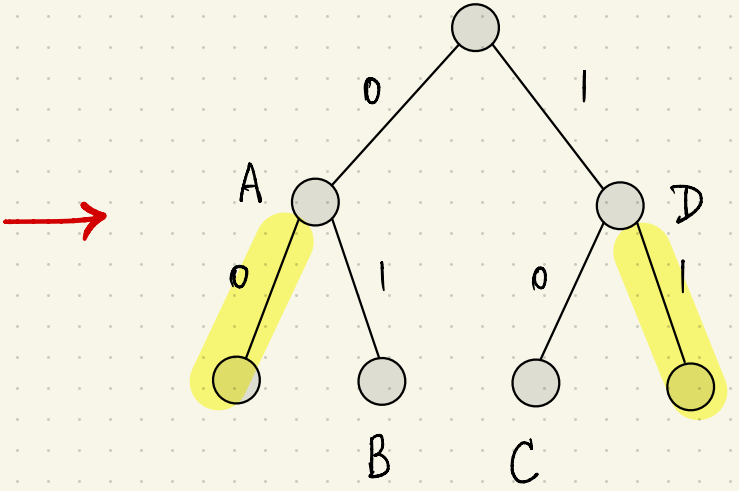
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1

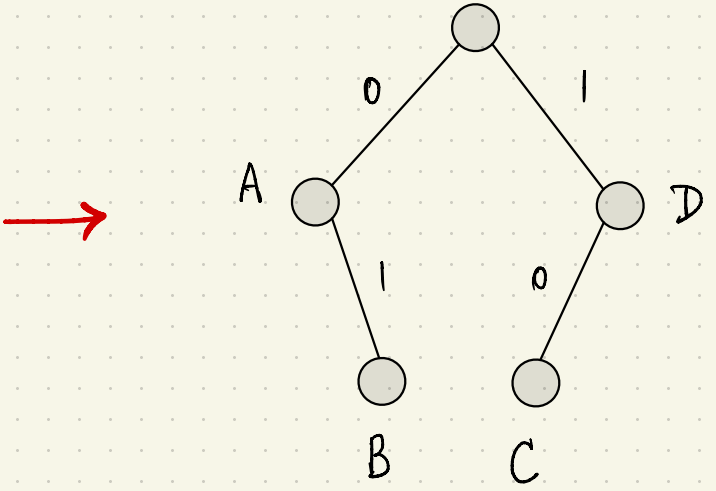


Repeatedly prune unlabeled leaves

BINARY CODES ⟷ LABELED BINARY TREES

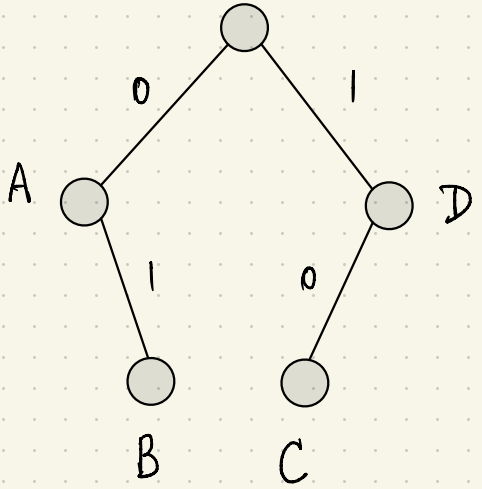
$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	01
C	10
D	1



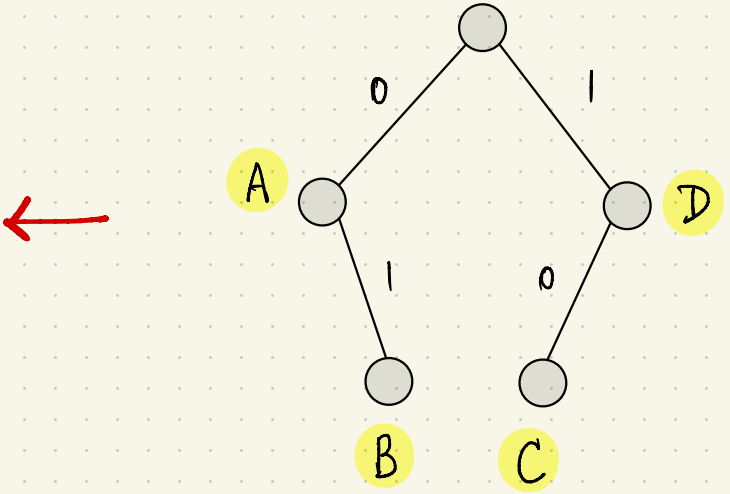
BINARY CODES ⟷ LABELED BINARY TREES

Symbol variable length



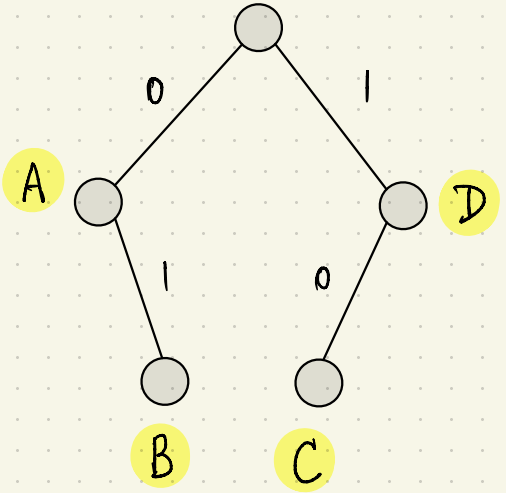
BINARY CODES ⟷ LABELED BINARY TREES

Symbol variable length



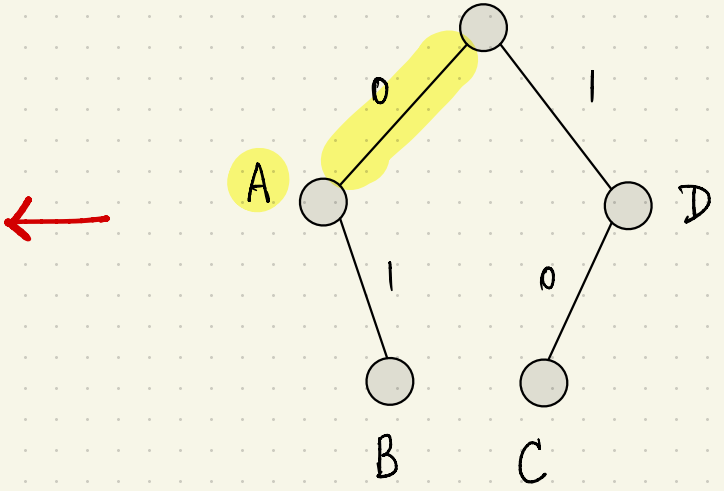
BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	
B	
C	
D	



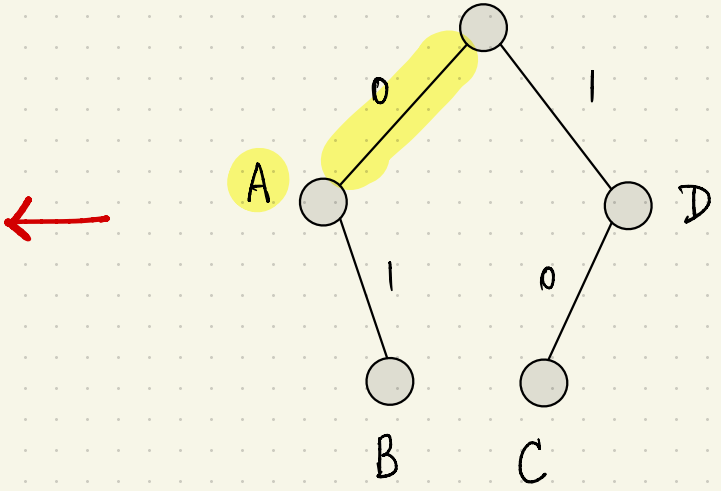
BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	
B	
C	
D	



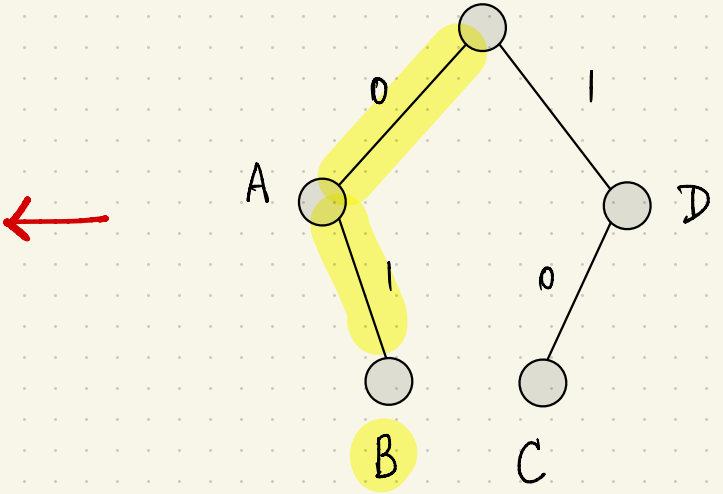
BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	0
B	
C	
D	



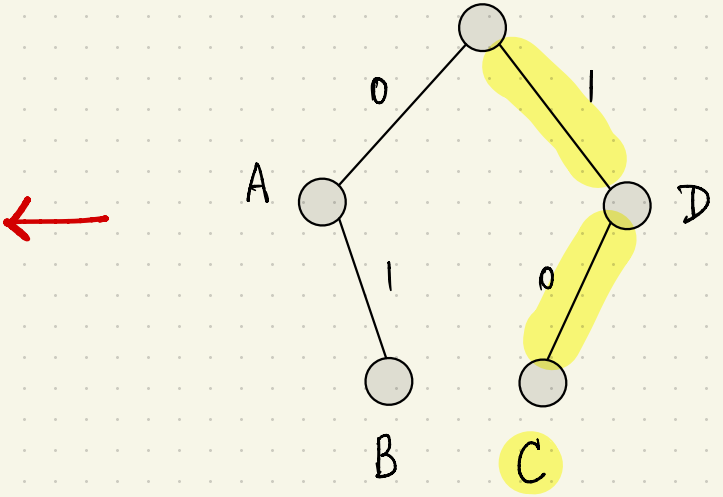
BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	0
B	01
C	
D	



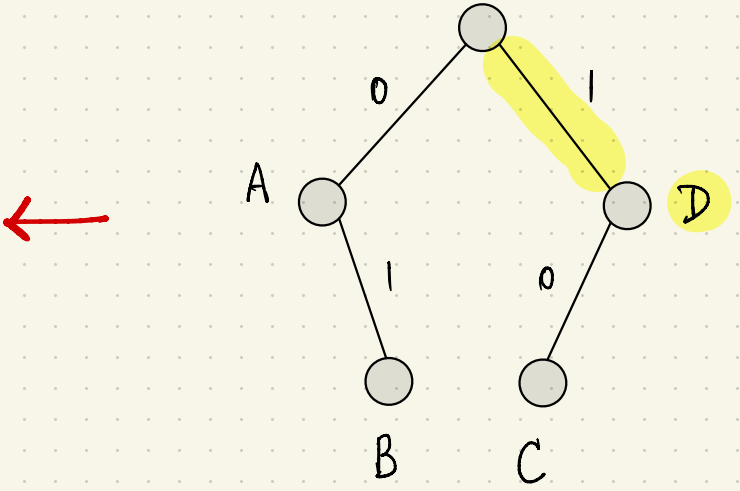
BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	0
B	01
C	10
D	



BINARY CODES ⟷ LABELED BINARY TREES

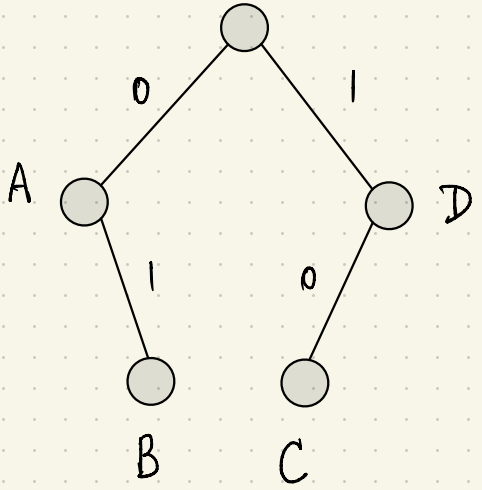
Symbol	variable length
A	0
B	01
C	10
D	1



BINARY CODES ⟷ LABELED BINARY TREES

Symbol	variable length
A	0
B	01
C	10
D	1

⟷

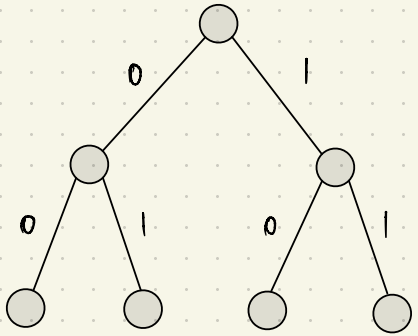


BINARY CODES ⟷ LABELED BINARY TREES

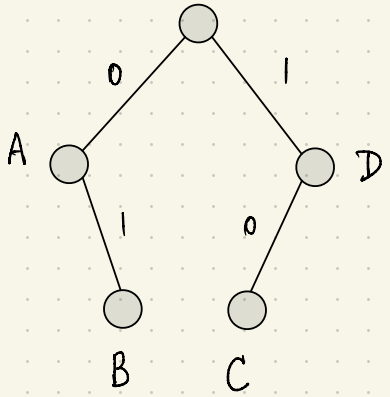
* Left child "0", right child "1"
* for each $i \in \Sigma$, exactly one node labeled "i"
* encoding of $i \in \Sigma$: bits along the path from root to node "i".

PREFIX-FREE CODES AS TREES

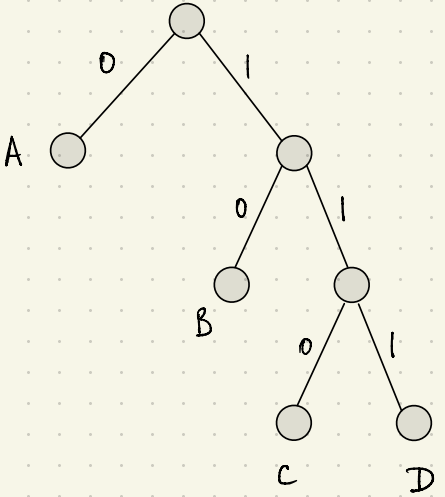
PREFIX-FREE CODES AS TREES



prefix-free

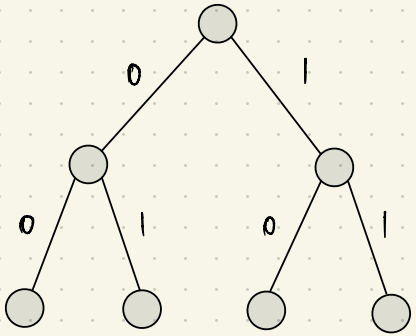


not prefix-free

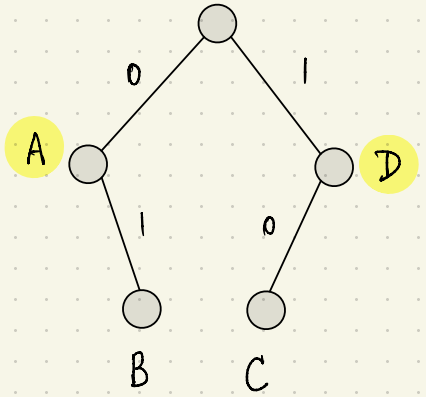


prefix-free

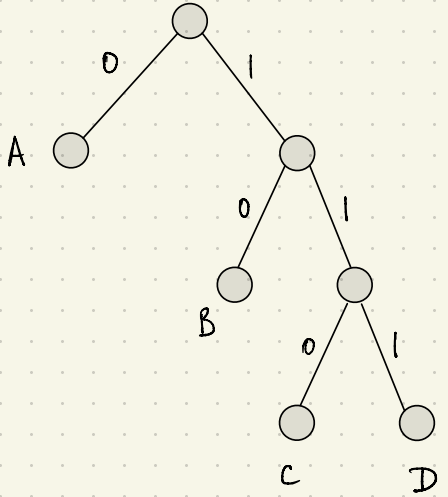
PREFIX-FREE CODES AS TREES



prefix-free

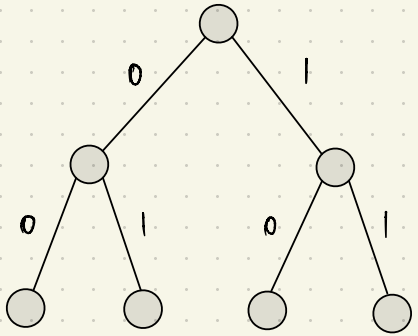


not prefix-free

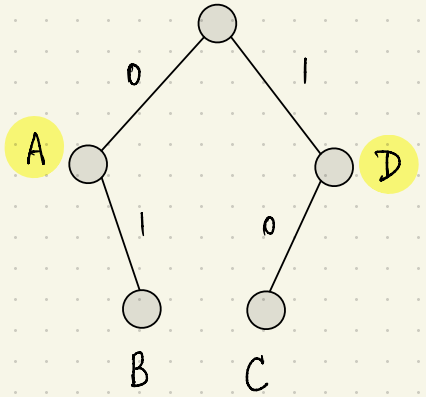


prefix-free

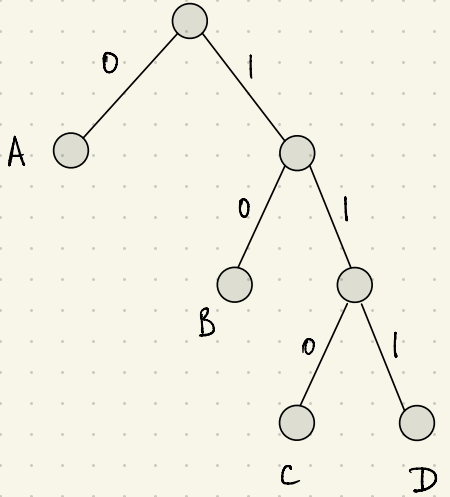
PREFIX-FREE CODES AS TREES



prefix-free



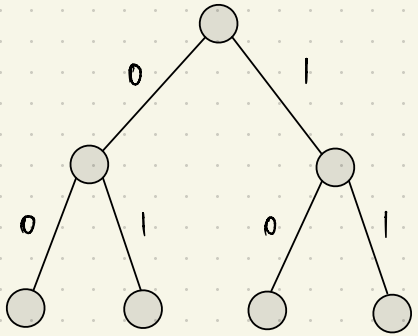
not prefix-free



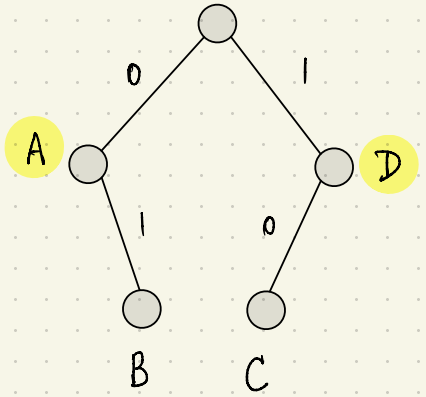
prefix-free

symbol i prefix of symbol j $\iff$ node i ancestor of node j

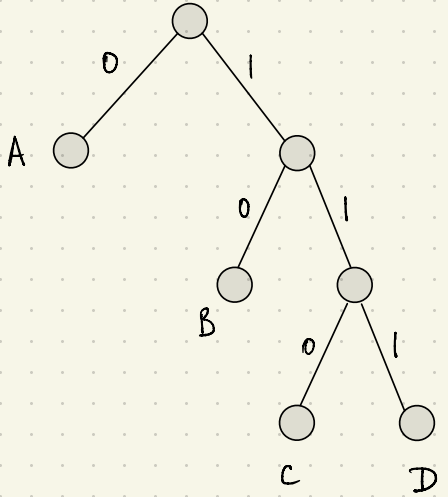
PREFIX-FREE CODES AS TREES



prefix-free



not prefix-free



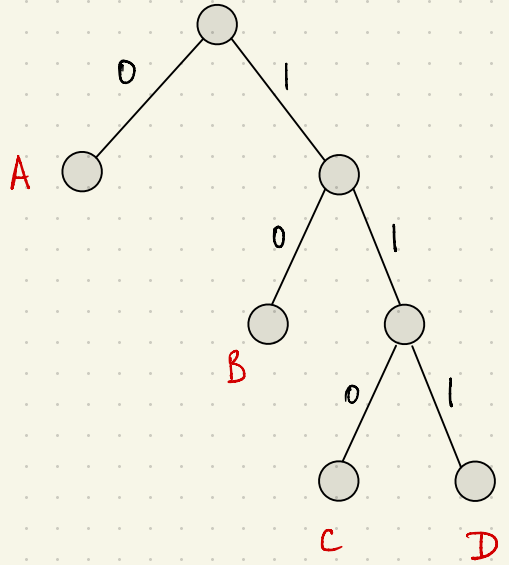
prefix-free

symbol i prefix of symbol j $\iff$ node i ancestor of node j

prefix-free code $\iff$ only leaves are labeled

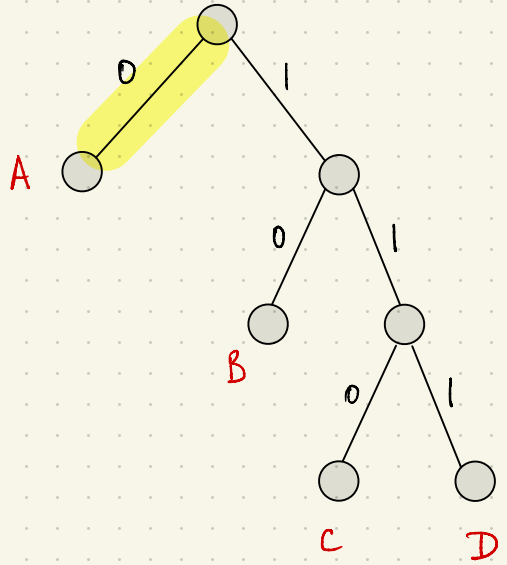
PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1



PREFIX-FREE CODES AS TREES

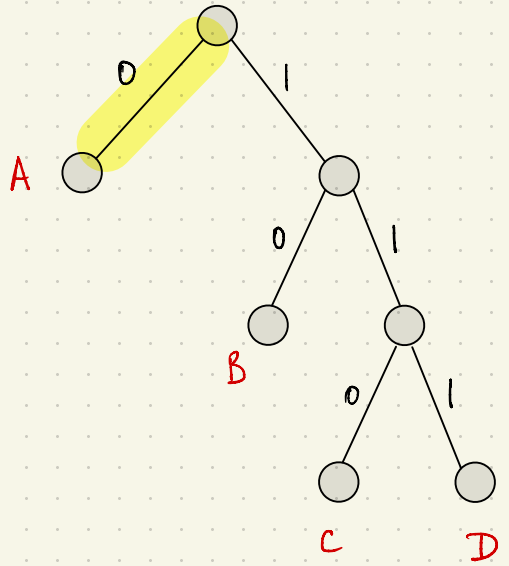
* Decoding is easy! e.g., 0 1 1 0 1 1 1



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1

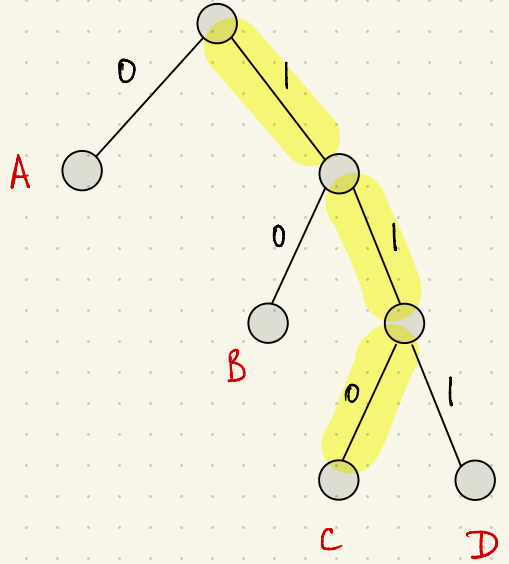
A



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1

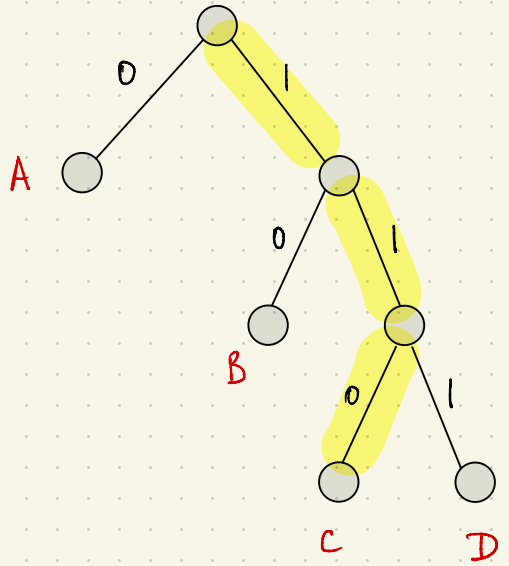
A



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1

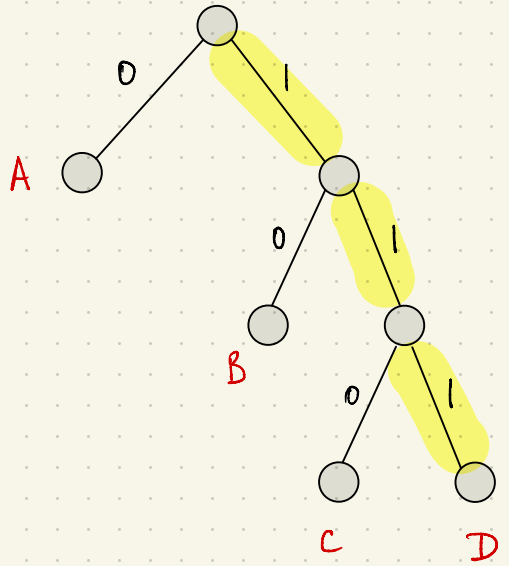
A C



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1

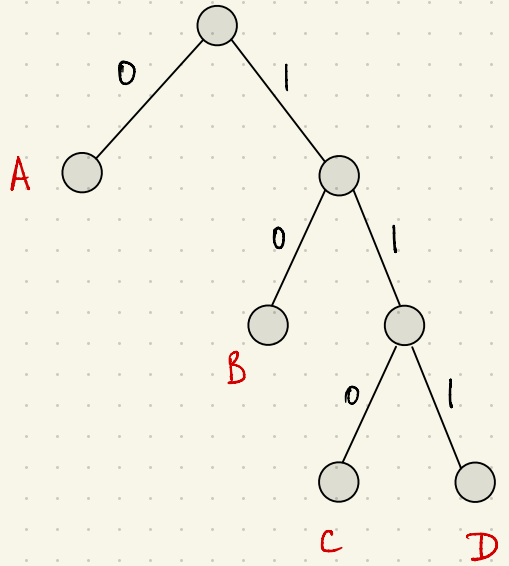
A C D



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1

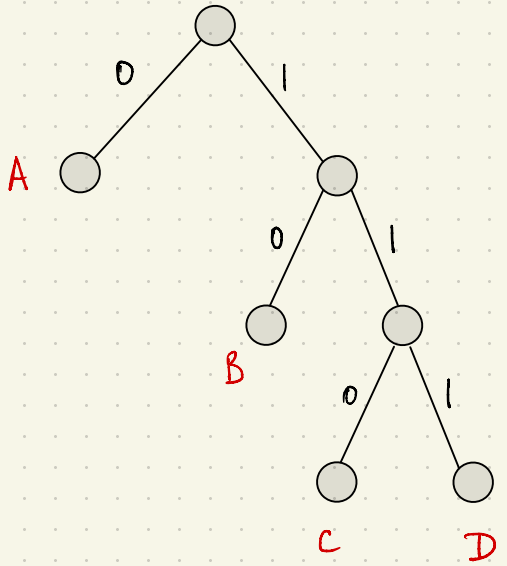
A C D



PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1
  A C D

* Encoding length of symbol $i \in \Sigma$
  = depth of leaf labeled i in the tree

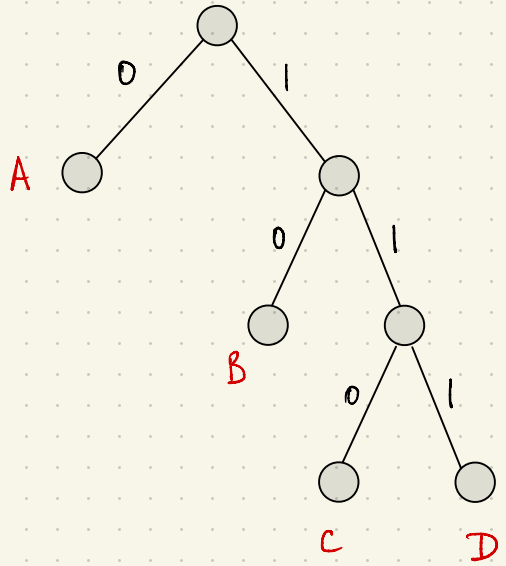


PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1
  A C D

* Encoding length of symbol $i \in \Sigma$
  = depth of leaf labeled i in the tree

* Σ tree: a binary tree with leaves labeled in one-to-one correspondence with symbols of Σ.



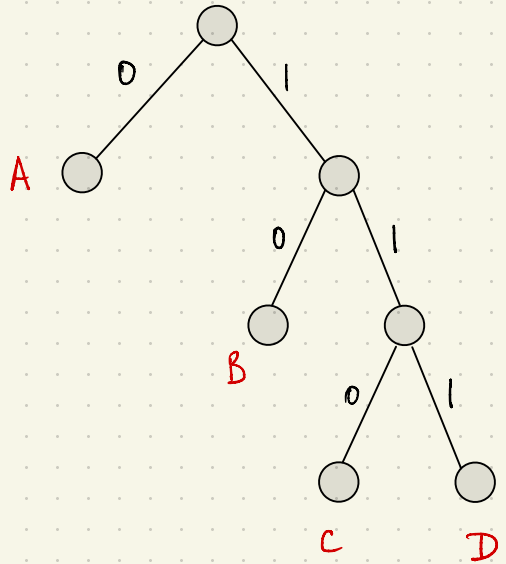
PREFIX-FREE CODES AS TREES

* Decoding is easy! e.g., 0 1 1 0 1 1 1
  A C D

* Encoding length of symbol $i \in \Sigma$
  = depth of leaf labeled i in the tree

* Σ tree: a binary tree with leaves labeled in one-to-one correspondence with symbols of Σ.

* For any Σ, some Σ-tree always exists
  e.g., $\{0, 10, 110, 1110, \ldots\}$



Optimal Prefix-free codes

input : a non negative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

Optimal Prefix-free codes

input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output: A prefix-free binary code with minimum possible average encoding length

$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

Optimal Prefix-free codes

input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output: A prefix-free binary code with minimum possible average encoding length

$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

Optimal Prefix-free codes

input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

output: A Σ-tree T with minimum possible average leaf depth

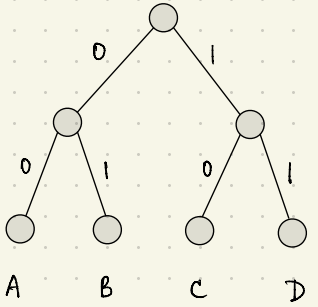
$$L(T) := \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i \text{ in } T$$

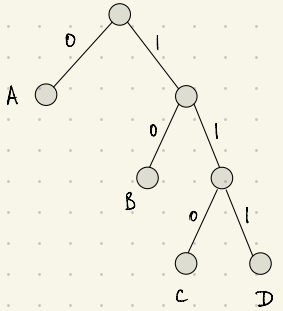
Optimal Prefix-free codes

symbol	frequency	fixed length	variable length
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

Optimal Prefix-free codes

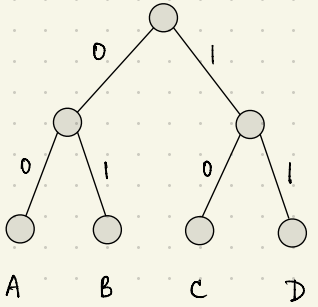
symbol	frequency	fixed length	variable length
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

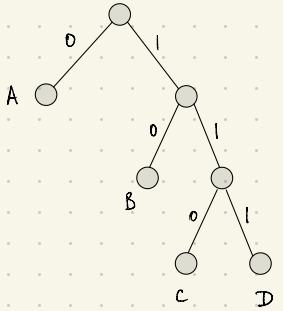




Optimal Prefix-free codes

symbol	frequency	fixed length	variable length
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111





Average leaf depth : 2 1.55

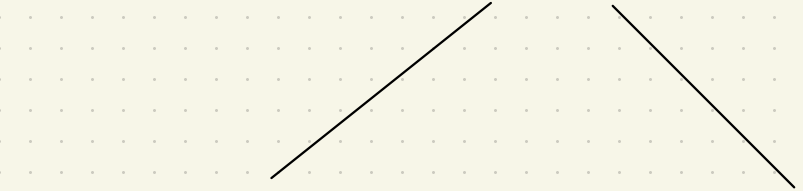
$= 0.6 \times 1 + 0.25 \times 2 + 0.15 \times 3$

BUILDING A TREE

What's a principled approach for building a Σ-tree?

BUILDING A TREE

What's a principled approach for building a Σ-tree?



Top down

(divide and conquer)

Bottom up

(greedy)

TOP DOWN APPROACH

[Shannon - Fano encoding]

TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer

TOP DOWN APPROACH

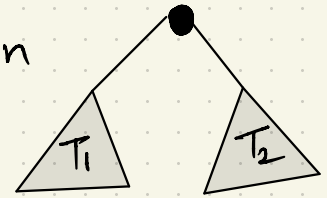
[Shannon - Fano encoding]

* Divide and conquer
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency
* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



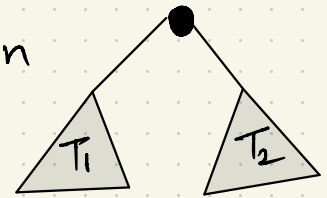
TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer

* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



e.g., $\Sigma = \{ a \quad b \quad c \quad d \quad e \}$

9% 41% 35% 5% 10%

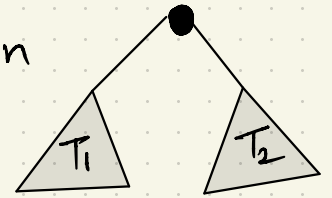
TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer

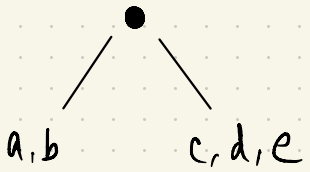
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



e.g., $\Sigma = \{ a \quad b \quad c \quad d \quad e \}$

9% 41% 35% 5% 10%



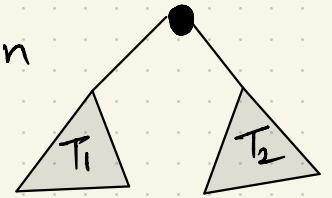
TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer

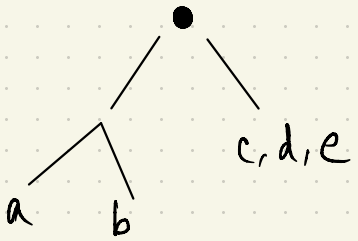
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



e.g., $\Sigma = \{ a \quad b \quad c \quad d \quad e \}$

9% 41% 35% 5% 10%



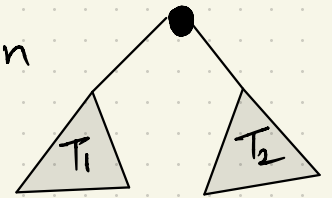
TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer

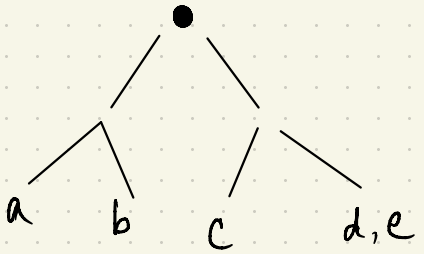
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



e.g., $\Sigma = \{ a \quad b \quad c \quad d \quad e \}$

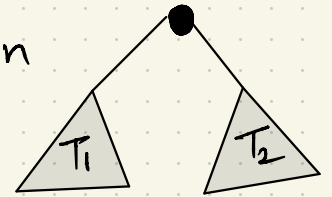
9% 41% 35% 5% 10%



TOP DOWN APPROACH

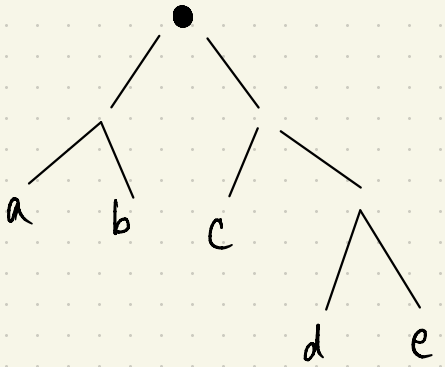
[Shannon - Fano encoding]

* Divide and conquer
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency
* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



e.g., $\Sigma = \{ a \quad b \quad c \quad d \quad e \}$

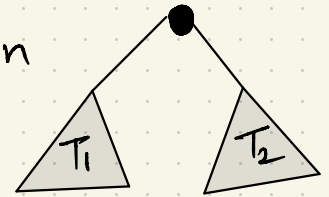
9% 41% 35% 5% 10%



TOP DOWN APPROACH

[Shannon - Fano encoding]

* Divide and conquer
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency
* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



* Suboptimal ☹ [Tutorial sheet 6]