COL 351 : Analysis & Design of Algorithms

Lecture 15

Greedy Algorithms II :

Job Scheduling and Huffman Coding

Aug 28, 2024 | Rohit Vaish

Greedy Algorithms

Do what looks best right now and hope everything works out at the end

JOB SCHEDULING

JOB SCHEDULING

input : a set of n jobs with positive lengths $l_1, l_2, \ldots, l_n$ and positive weights $w_1, w_2, \ldots, w_n$

output : a job schedule (or sequence) that minimizes the sum of weighted completion times $\sum_{j=1}^{n} w_j \cdot C_j$.

JOB SCHEDULING

input : a set of n jobs with positive lengths $l_1, l_2, \ldots, l_n$ and positive weights $w_1, w_2, \ldots, w_n$

output: a job schedule (or sequence) that minimizes the sum of weighted completion times $\sum_{j=1}^{n} w_j \cdot C_j$.

Brute force : $O(n!)$ ☹

COMPLETION TIMES

The completion time C_j of job j =

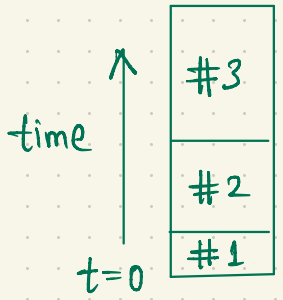
sum of job lengths up to and including j.

COMPLETION TIMES

The completion time C_j of job j =

sum of job lengths up to and including j.

E.g., $l_1 = 1$, $l_2 = 2$, $l_3 = 3$



$C_1 = 1$ $C_2 = 3$ $C_3 = 6$

OBJECTIVE FUNCTION

goal : minimize the weighted sum of completion times

$$\min \sum_{j=1}^{n} w_j \cdot C_j$$

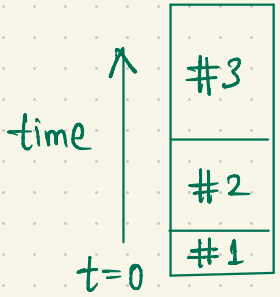
OBJECTIVE FUNCTION

goal : minimize the weighted sum of completion times

$$\min \sum_{j=1}^{n} w_j \cdot C_j$$

E.g., 3 jobs , lengths $l_1 = 1$, $l_2 = 2$, $l_3 = 3$

weights $w_1 = 3$, $w_2 = 2$, $w_3 = 1$



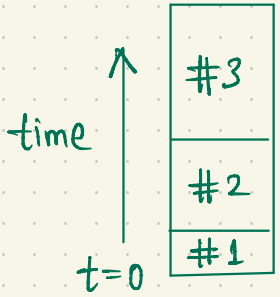
OBJECTIVE FUNCTION

goal : minimize the weighted sum of completion times

$$\min \sum_{j=1}^{n} w_j \cdot C_j$$

E.g., 3 jobs, lengths $l_1 = 1$, $l_2 = 2$, $l_3 = 3$

weights $w_1 = 3$, $w_2 = 2$, $w_3 = 1$



$$\sum_{j=1}^{n} w_j \cdot C_j = \underset{w_1}{3} \times \overset{C_1}{1} + \underset{w_2}{2} \times \overset{C_2}{3} + \underset{w_3}{1} \times \overset{C_3}{6}$$

$$= 15$$

SPECIAL CASES

goal : min $\sum_{j=1}^{n} w_j \cdot C_j$

(I) All lengths equal : schedule heavier job earlier

(II) All weights equal : schedule shorter job earlier

GENERAL INPUTS

Heavier jobs may not be shorter —— conflicting advice.

GENERAL INPUTS

Heavier jobs may not be shorter —— conflicting advice.

Idea: Use a score function

* increasing in weight
* decreasing in length

GENERAL INPUTS

Heavier jobs may not be shorter —— conflicting advice.

Idea: Use a score function

* increasing in weight
* decreasing in length

Proposal 1 : schedule in decreasing order of $w_j - l_j$

Proposal 2 : schedule in decreasing order of w_j / l_j.

GENERAL INPUTS

Heavier jobs may not be shorter —— conflicting advice.

Idea: Use a score function

* increasing in weight
* decreasing in length

Proposal 1: schedule in decreasing order of $w_j - l_j$ (suboptimal)

Proposal 2: schedule in decreasing order of w_j / l_j.

CORRECTNESS OF ORDER-BY-RATIO

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: (by an exchange argument)

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: (by an exchange argument)

* by contradiction: without ties

* by massaging: with ties

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: (by an exchange argument)

* by contradiction: without ties

* by massaging: with ties

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Two assumptions:

1. jobs are indexed in decreasing order of ratios

$$\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}.$$

2. no ties between ratios

$$\frac{w_i}{\ell_i} \neq \frac{w_j}{\ell_j} \quad \text{whenever } i \neq j$$

CORRECTNESS OF ORDER-BY-RATIO

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : Greedy schedule : σ Optimal schedule : $\sigma^* \neq \sigma$

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule : σ Optimal schedule : $\sigma^* \neq \sigma$

consecutive inversion

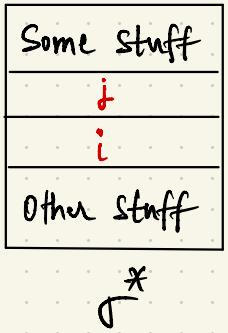
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Greedy schedule: σ Optimal schedule: $\sigma^* \neq \sigma$

consecutive inversion

$i > j$



σ^*

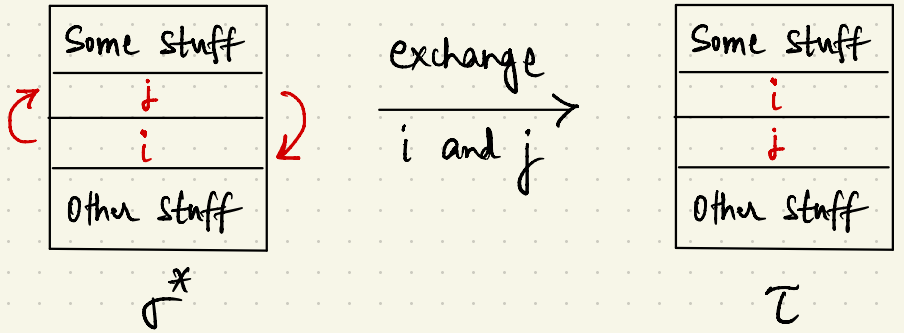
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule: σ Optimal schedule: $\sigma^* \neq \sigma$

consecutive inversion

$i > j$



σ^* τ

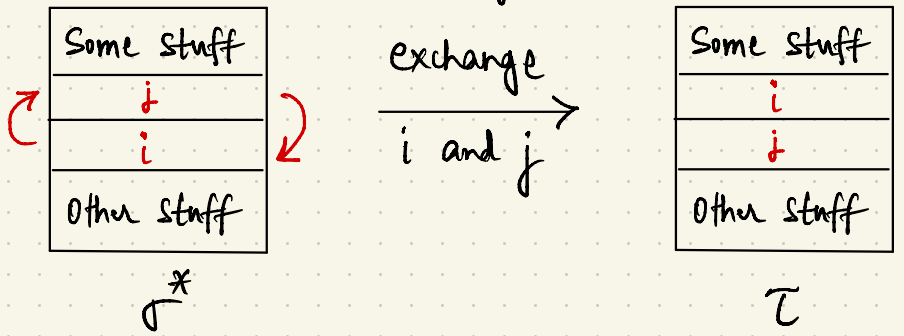
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule: σ Optimal schedule: $\sigma^* \neq \sigma$

consecutive inversion

$$i > j \Rightarrow \frac{w_i}{l_i} < \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j < w_j \cdot l_i$$



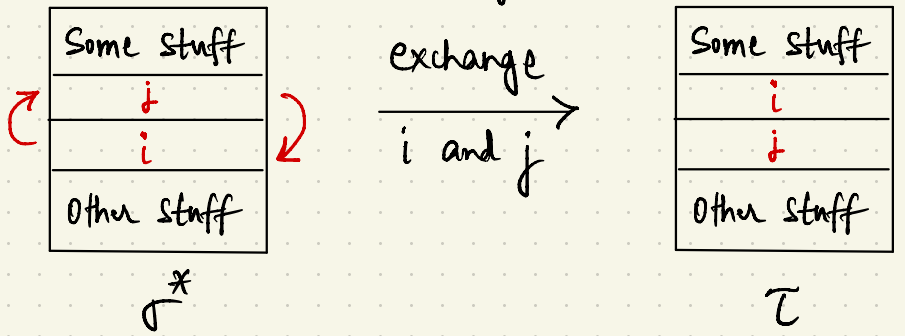
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule : σ Optimal schedule : $\sigma^* \neq \sigma$

consecutive inversion

$$i > j \Rightarrow \frac{w_i}{l_i} < \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j < w_j \cdot l_i$$



σ^* τ

Objective (τ) =

Objective $(\sigma^*) + w_i l_j - w_j l_i$

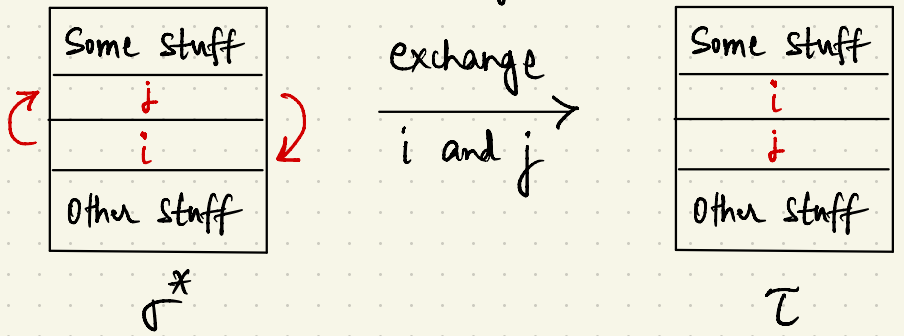
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule : σ Optimal schedule : $\sigma^* \neq \sigma$

consecutive inversion

$$i > j \Rightarrow \frac{w_i}{l_i} < \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j < w_j \cdot l_i$$



σ^* τ

Objective (τ) =

Objective (σ^*) + $\underbrace{w_i l_j - w_j l_i}_{<0}$

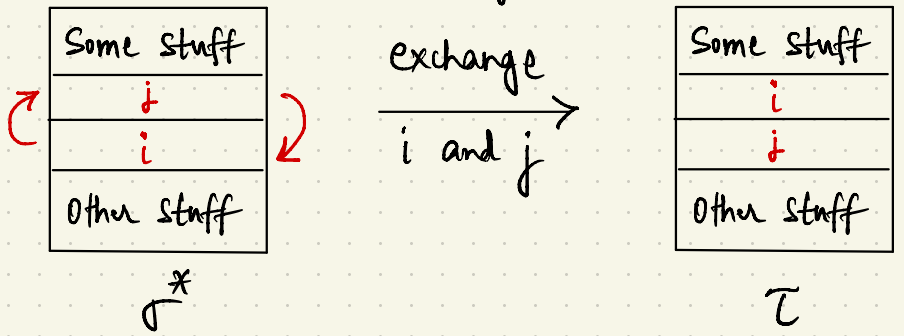
CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Greedy schedule: σ Optimal schedule: $\sigma^* \neq \sigma$

consecutive inversion

$$i > j \Rightarrow \frac{w_i}{l_i} < \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j < w_j \cdot l_i$$



σ^* τ

Objective (τ) =

Objective (σ^*) + $\underbrace{w_i l_j - w_j l_i}_{<0}$

τ is strictly better than σ^*

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Two variants:

* by contradiction: without ties

* by massaging: with ties

CORRECTNESS OF ORDER-BY-RATIO

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Two variants:

* by contradiction: without ties

* by massaging: with ties

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Recall the two assumptions:

(1) jobs are indexed in decreasing order of ratios

$$\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}.$$

(2) no ties between ratios

$$\frac{w_i}{\ell_i} \neq \frac{w_j}{\ell_j} \quad \text{whenever} \quad i \neq j$$

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Recall the two assumptions:

① jobs are indexed in decreasing order of ratios

$$\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}.$$

~~②~~ no ties between ratios

$$\frac{w_i}{\ell_i} \neq \frac{w_j}{\ell_j} \quad \text{whenever} \quad i \neq j$$

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Let σ = greedy schedule, τ = any other schedule (not necessarily optimal)

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: Let σ = greedy schedule, τ = any other schedule (not necessarily optimal)

Will show that σ is at least as good as τ

$\Rightarrow$ greedy schedule is optimal.

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof : Greedy schedule σ is $(1, 2, \cdots, n)$; thus, $\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}$

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof : Greedy schedule σ is $(1, 2, \cdots, n)$; thus, $\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}$.

Consider an arbitrary schedule τ.

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Greedy schedule σ is $(1, 2, \cdots, n)$; thus, $\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}$.

Consider an arbitrary schedule τ.

If $\sigma = \tau$, we're done!

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Greedy schedule σ is $(1, 2, \cdots, n)$; thus, $\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}$.

Consider an arbitrary schedule τ.

If $\sigma = \tau$, we're done!

Else τ contains a consecutive inversion.

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/ℓ_j minimizes the weighted completion time.

Proof: Greedy schedule σ is $(1, 2, \cdots, n)$; thus, $\frac{w_1}{\ell_1} \geq \frac{w_2}{\ell_2} \geq \cdots \geq \frac{w_n}{\ell_n}$.

Consider an arbitrary schedule τ.

If $\sigma = \tau$, we're done!

Else τ contains a consecutive inversion.

↳ consecutive jobs i, j in τ such that $i > j$.

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : Consecutive inversion i, j

$$i > j \Rightarrow \frac{w_i}{l_i} \leq \frac{w_j}{l_j}$$

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : Consecutive inversion i, j

$$i > j \Rightarrow \frac{w_i}{l_i} \leq \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j \leq w_j \cdot l_i .$$

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : Consecutive inversion i, j

$$i > j \Rightarrow \frac{w_i}{l_i} \leq \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j \leq w_j \cdot l_i$$

Observe,

$$\sum_{k=1}^{n} w_k \cdot C_k(\tau^{new}) = \sum_{K=1}^{n} w_k \cdot C_k(\tau) + w_i l_j - w_j l_i$$

objective for new schedule — objective for τ — effect of exchange

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : Consecutive inversion i, j

$$i > j \Rightarrow \frac{w_i}{l_i} \leq \frac{w_j}{l_j} \Rightarrow w_i \cdot l_j \leq w_j \cdot l_i$$

Observe,

$$\sum_{k=1}^{n} w_k \cdot C_k(\tau^{new}) = \sum_{K=1}^{n} w_k \cdot C_k(\tau) + \underbrace{w_i l_j - w_j l_i}_{\leq 0}$$

objective for new schedule — objective for τ — effect of exchange

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : τ^{new} is no worse than τ.

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: τ^{new} is no worse than τ.

After exchange of i and j, number of inversions between σ and τ decreases by 1.

HANDLING TIES

Theorem : Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof : τ^{new} is no worse than τ.

After exchange of i and j, number of inversions between σ and τ decreases by 1.

$\Rightarrow$ After at most ${}^{n}C_{2}$ exchanges, can transform τ to σ.

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: τ^{new} is no worse than τ.

After exchange of i and j, number of inversions between σ and τ decreases by 1.

$\Rightarrow$ After at most ${}^nC_2$ exchanges, can transform τ to σ.

$\Rightarrow$ σ is at least as good as any other schedule

HANDLING TIES

Theorem: Scheduling the jobs in decreasing order of ratios w_j/l_j minimizes the weighted completion time.

Proof: τ^{new} is no worse than τ.

After exchange of i and j, number of inversions between σ and τ decreases by 1.

$\Rightarrow$ After at most ${}^nC_2$ exchanges, can transform τ to σ.

$\Rightarrow$ σ is at least as good as any other schedule

$\Rightarrow$ σ must be optimal

HUFFMAN CODING

BINARY CODES

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

⋮

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

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

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

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

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

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

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

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 $n!$

one symbol with one bit,
another symbol with two bits,
another symbol with three bits, ...

CODES AS TREES

CODES AS TREES

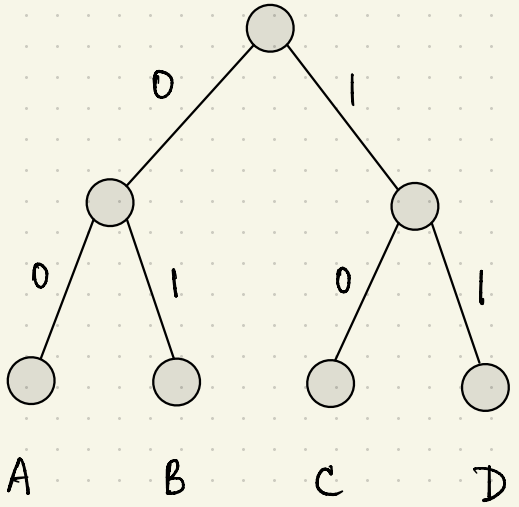
$\Sigma = \{A, B, C, D\}$

symbol	fixed length
A	00
B	01
C	10
D	11

CODES AS TREES

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

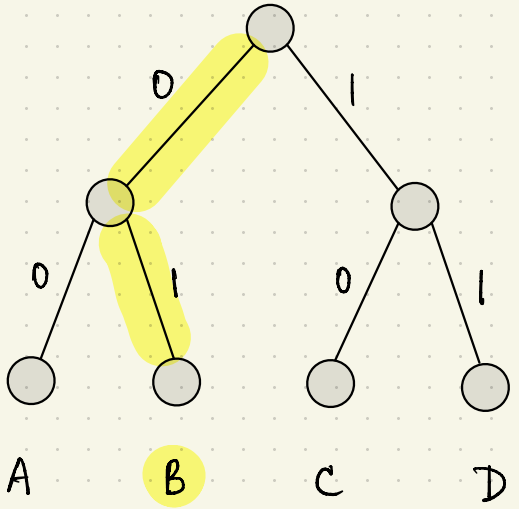
symbol	fixed length
A	00
B	01
C	10
D	11



CODES AS TREES

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

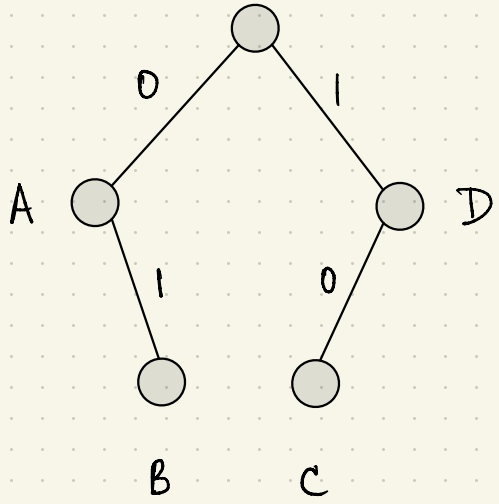
symbol	fixed length
A	00
B	01
C	10
D	11



CODES AS TREES

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

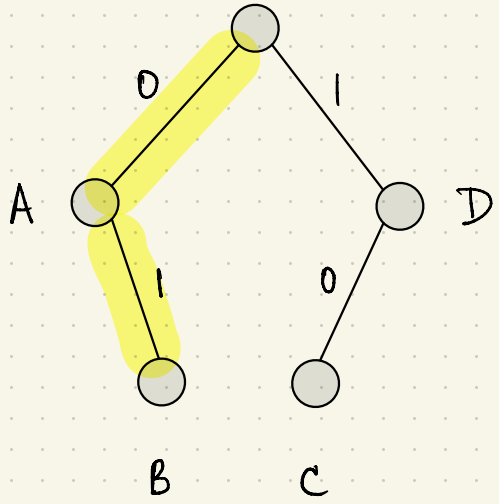
Symbol	variable length
A	0
B	01
C	10
D	1



CODES AS TREES

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

Symbol	variable length
A	0
B	01
C	10
D	1

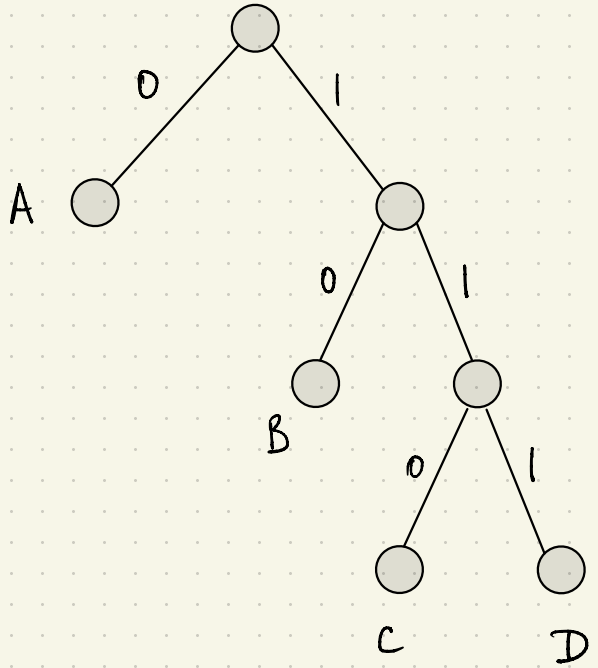


not prefix-free

CODES AS TREES

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

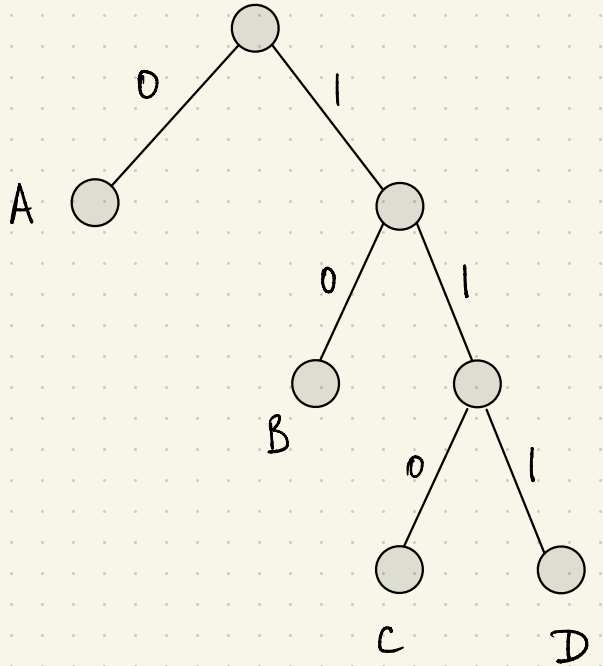
Symbol	variable length
A	0
B	10
C	110
D	111



CODES AS TREES

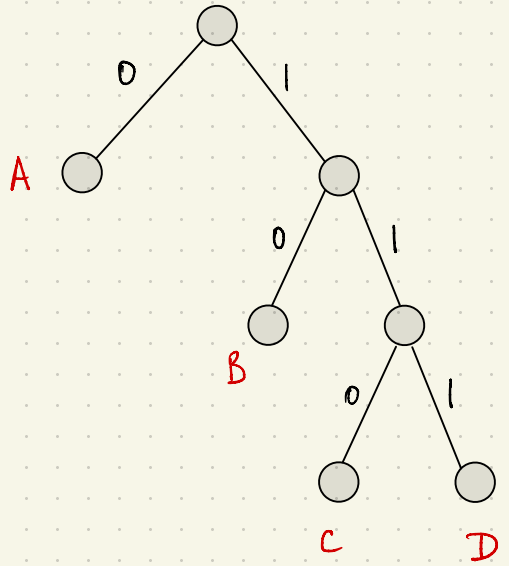
$\Sigma = \{A, B, C, D\}$

Symbol	variable length
A	0
B	10
C	110
D	111



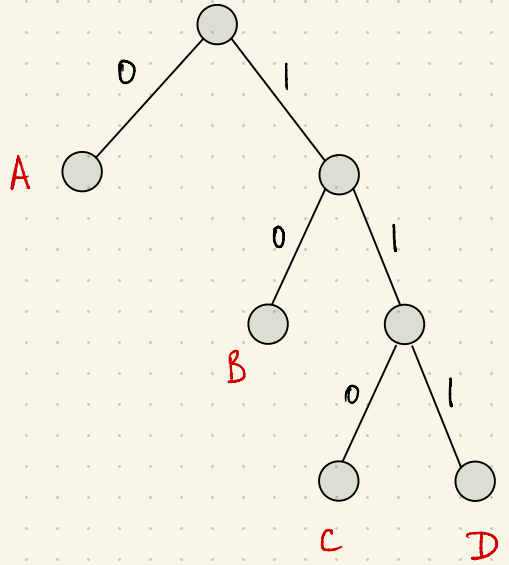
prefix-free

PREFIX-FREE CODES AS TREES



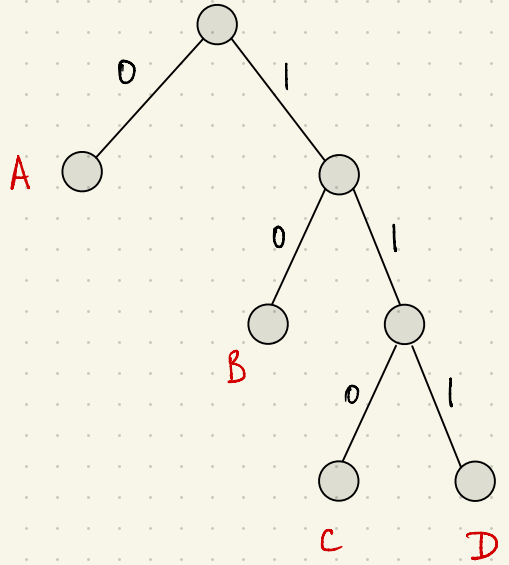
PREFIX-FREE CODES AS TREES

* Left child "0", right child "1"



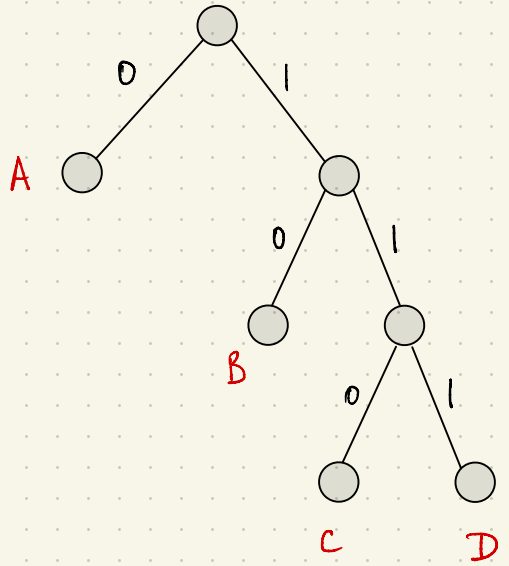
PREFIX-FREE CODES AS TREES

* Left child "0", right child "1"
* for each $i \in \Sigma$, exactly one node labeled "i"



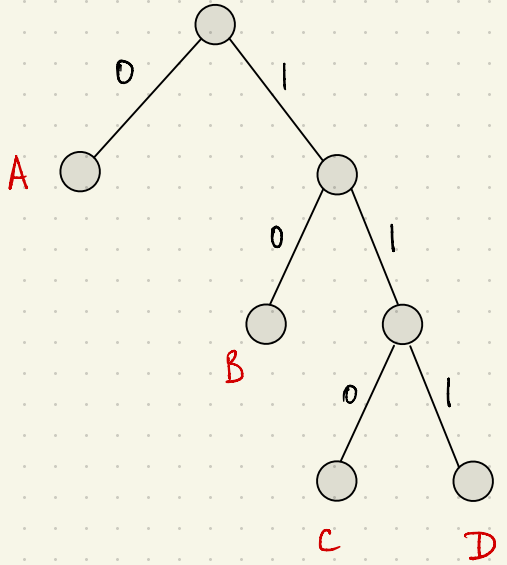
PREFIX-FREE CODES AS TREES

* 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

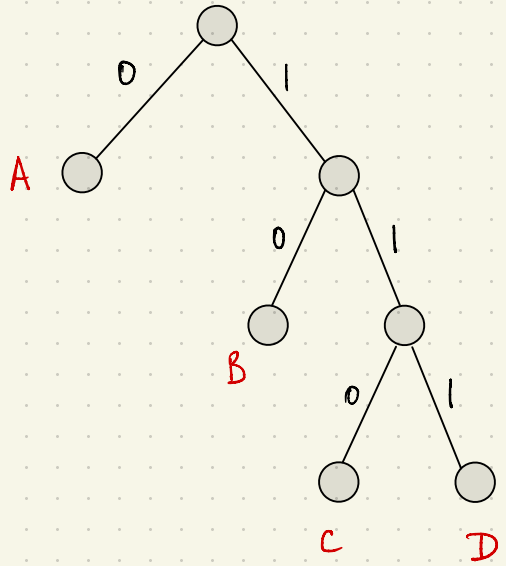
* Left child "0", right child "1"
* for each $i \in \Sigma$, exactly one node labeled "i"
* encoding of $i \in \Sigma$: bits along the path from root to node "i".
* prefix-free : labeled nodes ≡ leaves



PREFIX-FREE CODES AS 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

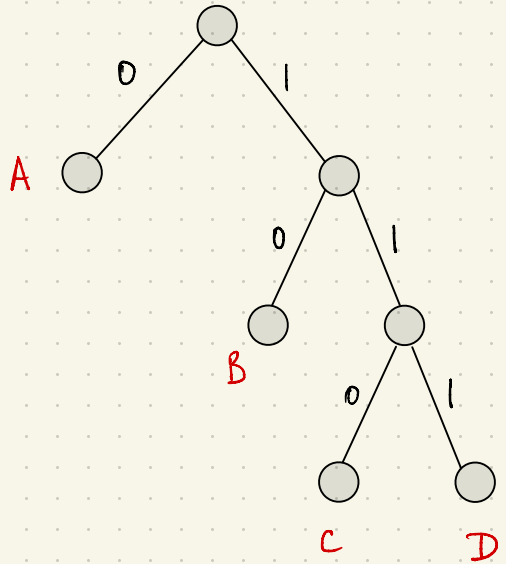


PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

* Decoding is easy!

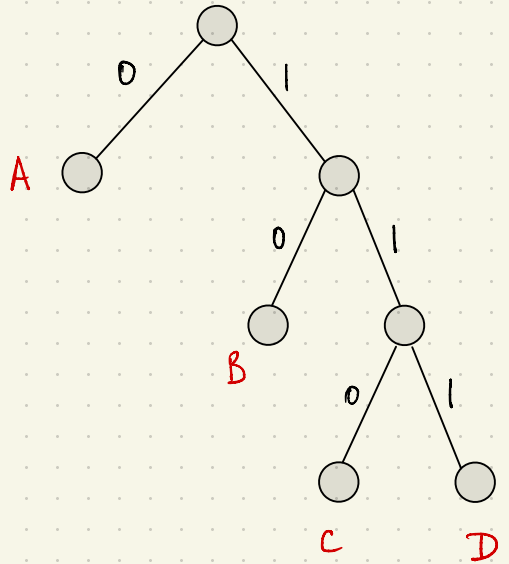


PREFIX-FREE CODES AS TREES

* 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: labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

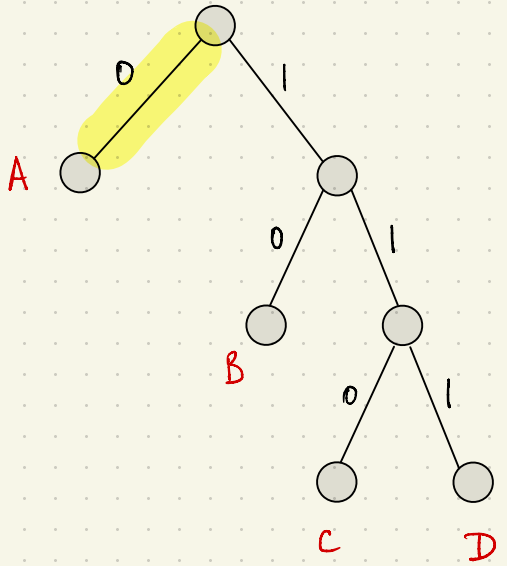


PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

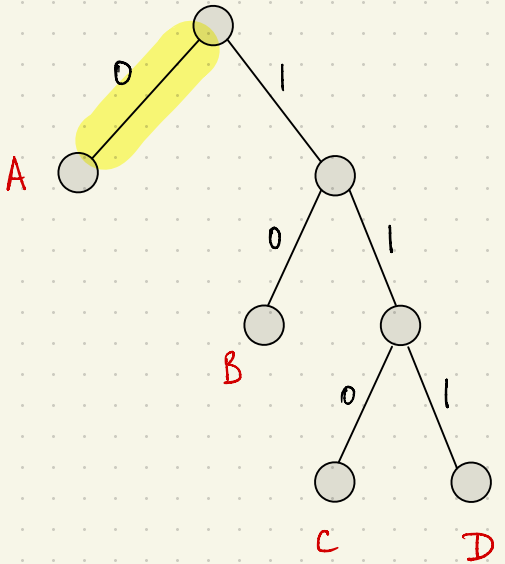


PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

* Decoding is easy! e.g., 0 1 1 0 1 1 1
  (0 = A)



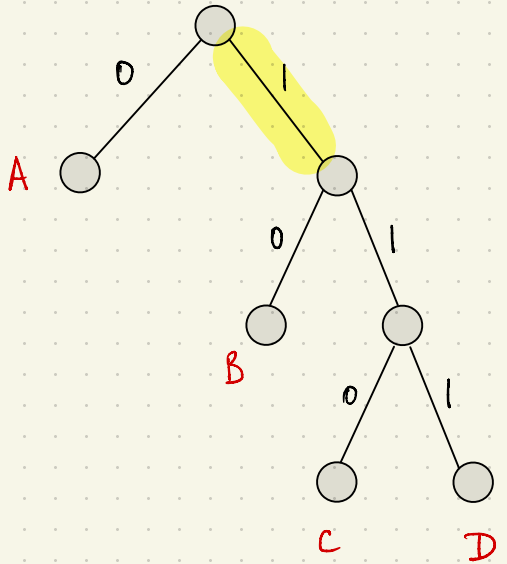
PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A



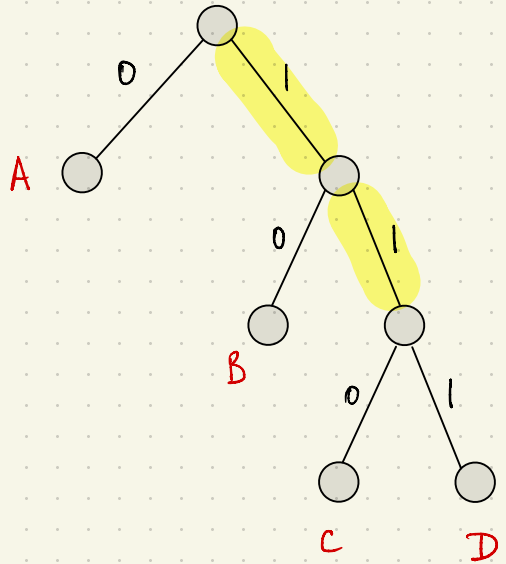
PREFIX-FREE CODES AS TREES

* 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: labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A



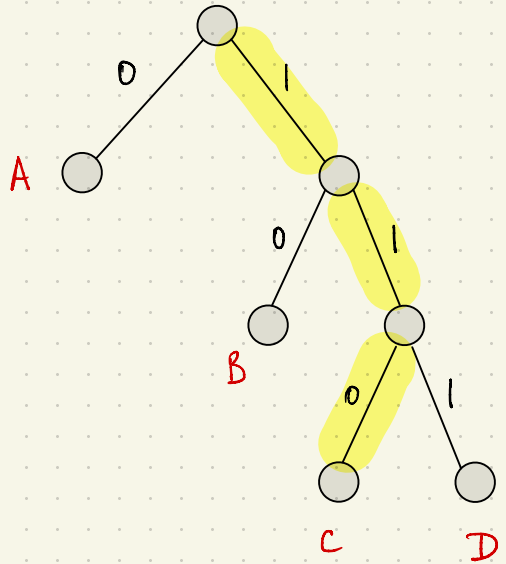
PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A



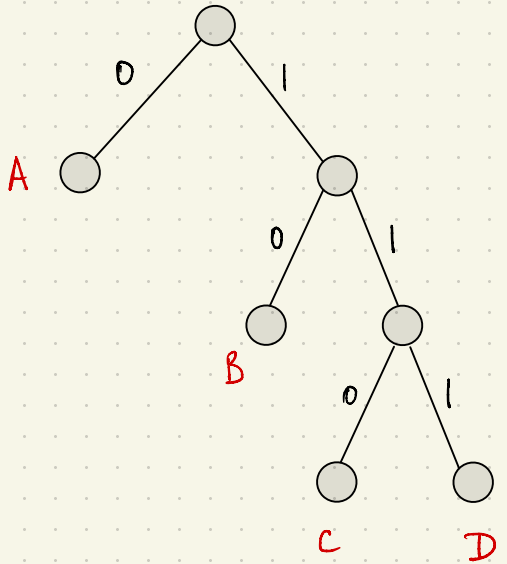
PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A C



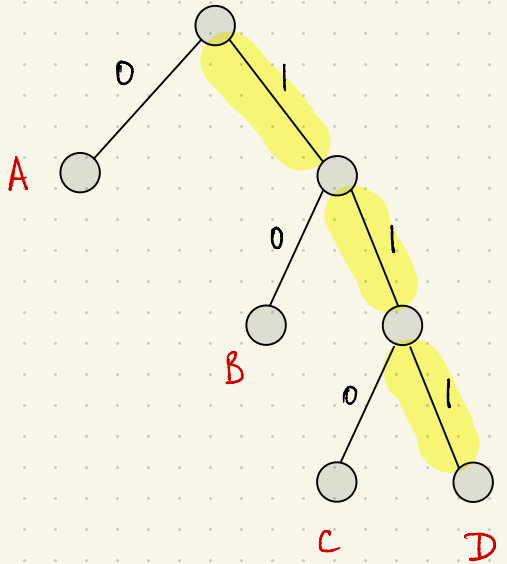
PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A C



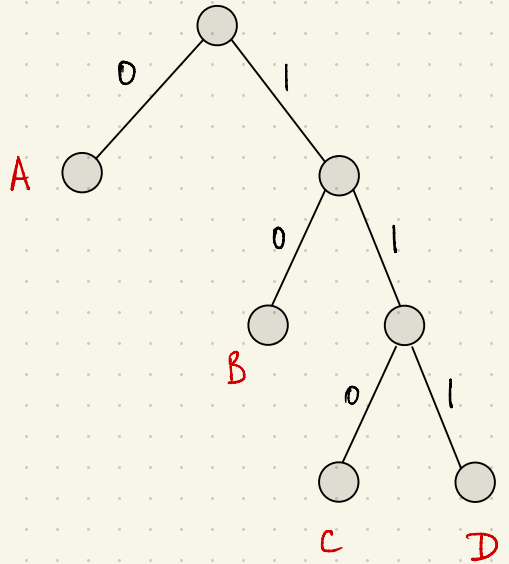
PREFIX-FREE CODES AS TREES

* 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: labeled nodes ≡ leaves

[prefixes ⟷ one node ancestor of another]

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

A C D

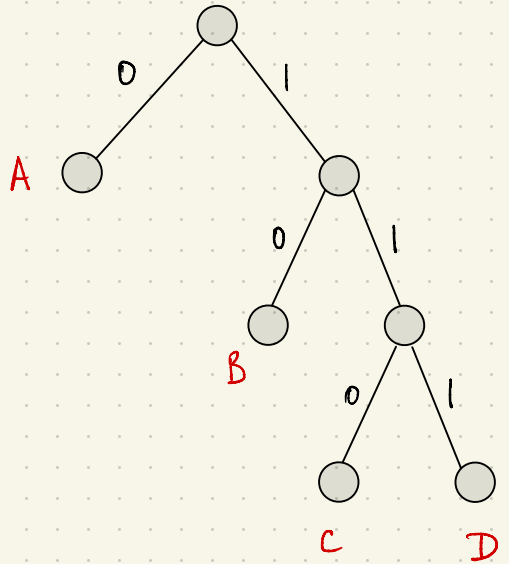


PREFIX-FREE CODES AS TREES

* 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 : labeled nodes ≡ leaves

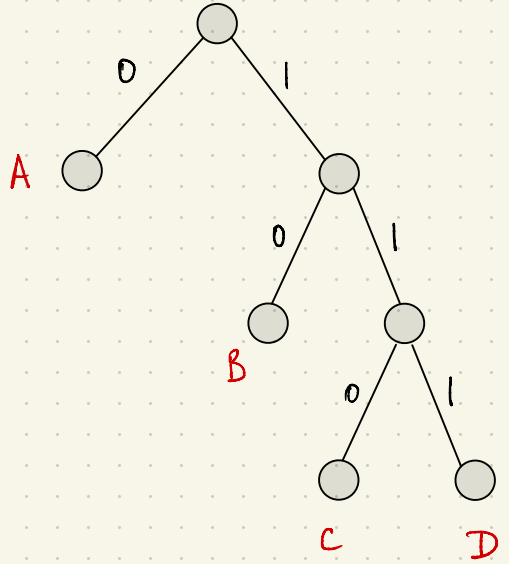
[prefixes ⟷ one node ancestor of another]

* Decoding is easy! e.g., 0 1 1 0 1 1 1 (A C D)
* Encoding length of $i \in \Sigma$ = depth of i in the tree



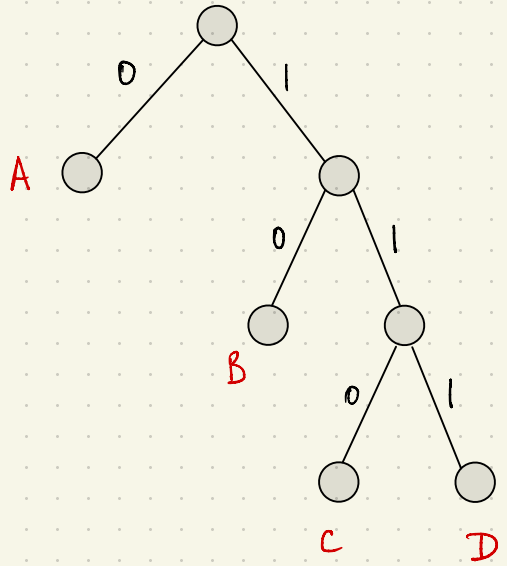
PREFIX-FREE CODES AS TREES

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



PREFIX-FREE CODES AS TREES

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



Note : 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 with minimum possible average leaf depth

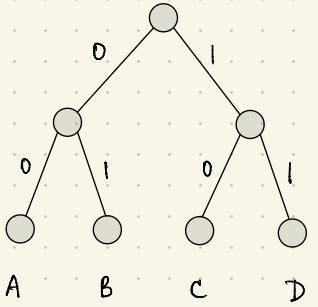
$$L(T) := \sum_{i \in \Sigma} p_i \cdot \text{depth of } i \text{ in } T$$

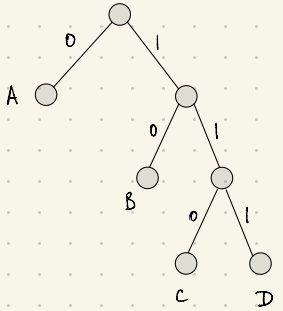
Optimal Prefix-free codes

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

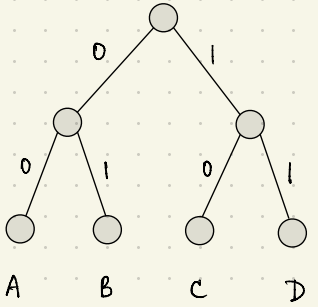
symbol	frequency	fixed length	variable length
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

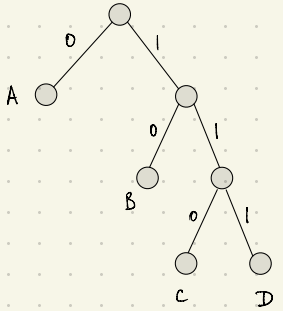




Optimal Prefix-free codes

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

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

HUFFMAN'S GREEDY ALGORITHM

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 approach [aka Fano-Shannon encoding]

BUILDING A TREE

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

Top down approach [aka Fano-Shannon encoding]

- divide and conquer

BUILDING A TREE

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

Top down approach [aka Fano-Shannon encoding]

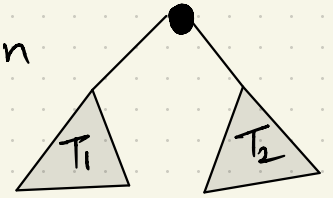
- divide and conquer
- partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency

BUILDING A TREE

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

Top down approach [aka Fano-Shannon 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

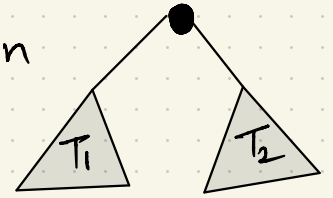


BUILDING A TREE

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

Top down approach [aka Fano-Shannon 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]

BUILDING A TREE

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

Bottom up approach [Huffman encoding]

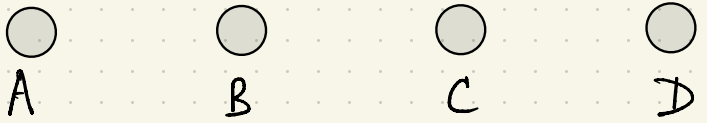
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



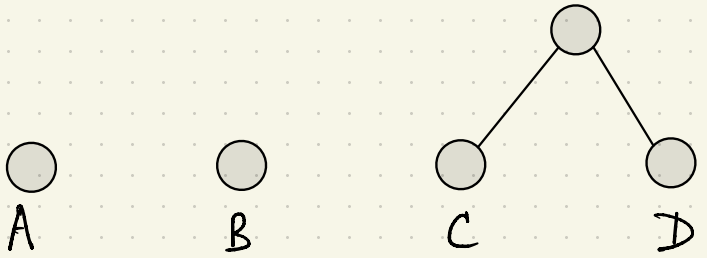
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



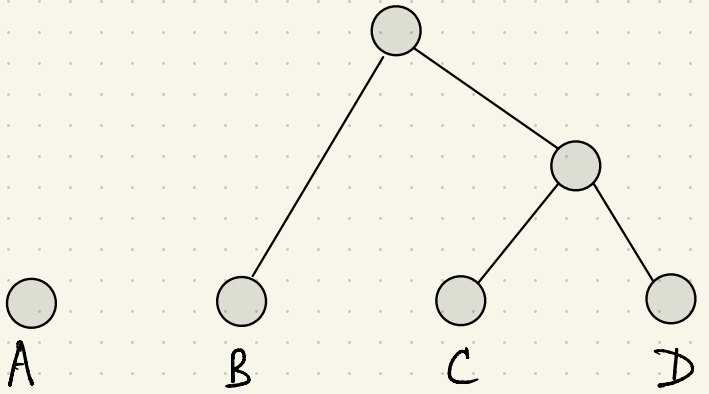
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



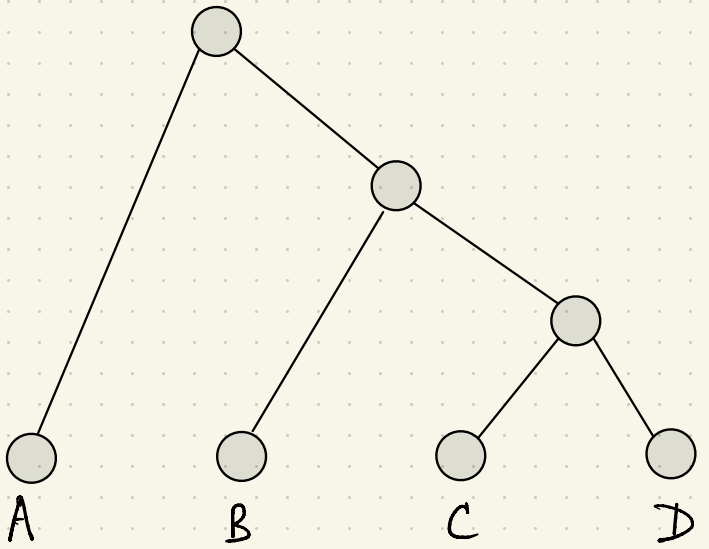
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



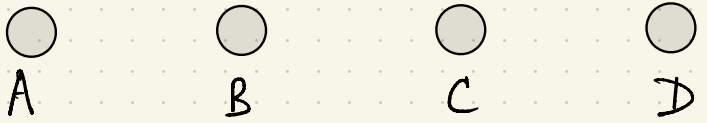
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



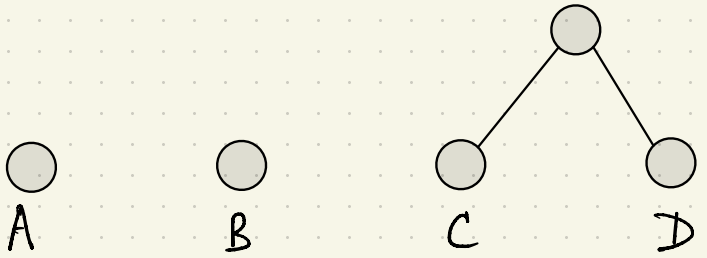
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



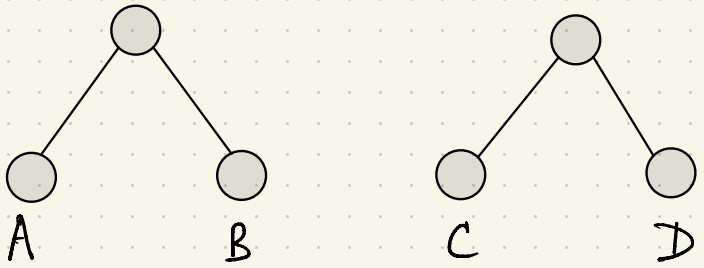
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

  Successive mergers



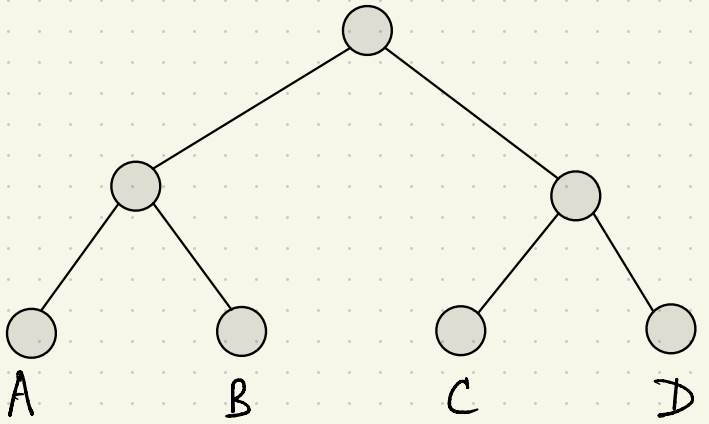
BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- greedy via

Successive mergers



BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- after each merge

👍

👎

BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- after each merge

  👍 # trees in the forest decreases by 1

  👎 average depth of leaves increases

BUILDING A TREE

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

Bottom up approach [Huffman encoding]

- after each merge

  👍 # trees in the forest decreases by 1

  👎 average depth of leaves increases

Which pair of trees is "safe" to merge?

BUILDING A TREE

Which pair of trees is "safe" to merge?

BUILDING A TREE

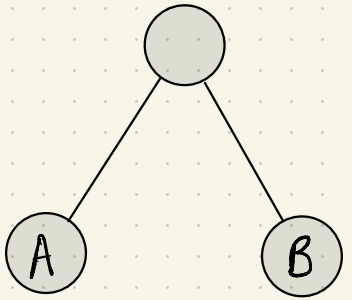
Which pair of trees is "safe" to merge?

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

BUILDING A TREE

Which pair of trees is "safe" to merge?

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



Not much to do

BUILDING A TREE

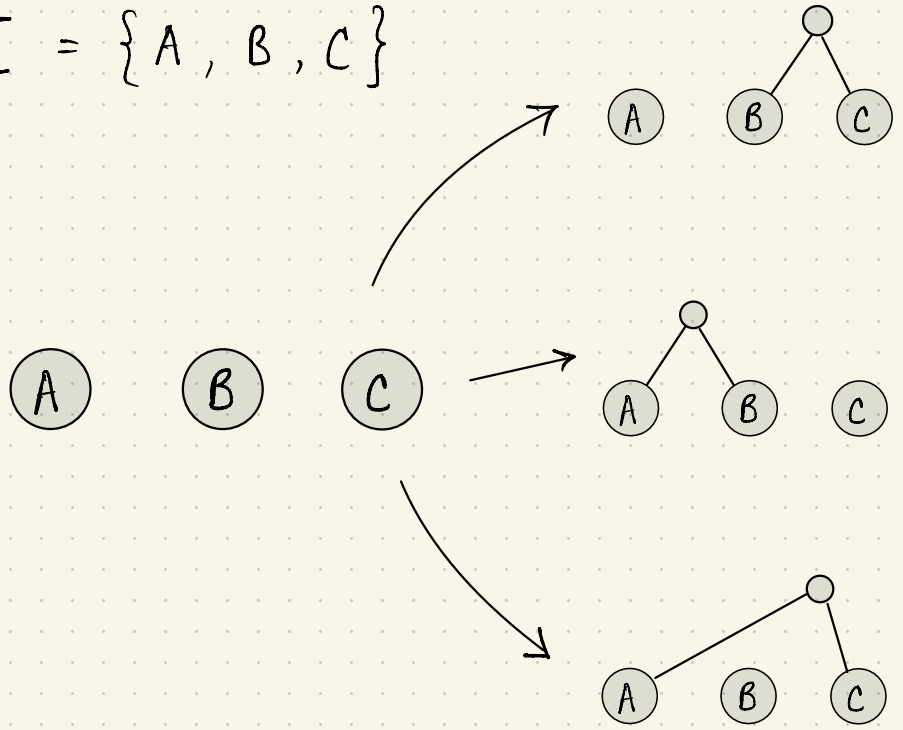
Which pair of trees is "safe" to merge?

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

BUILDING A TREE

Which pair of trees is "safe" to merge?

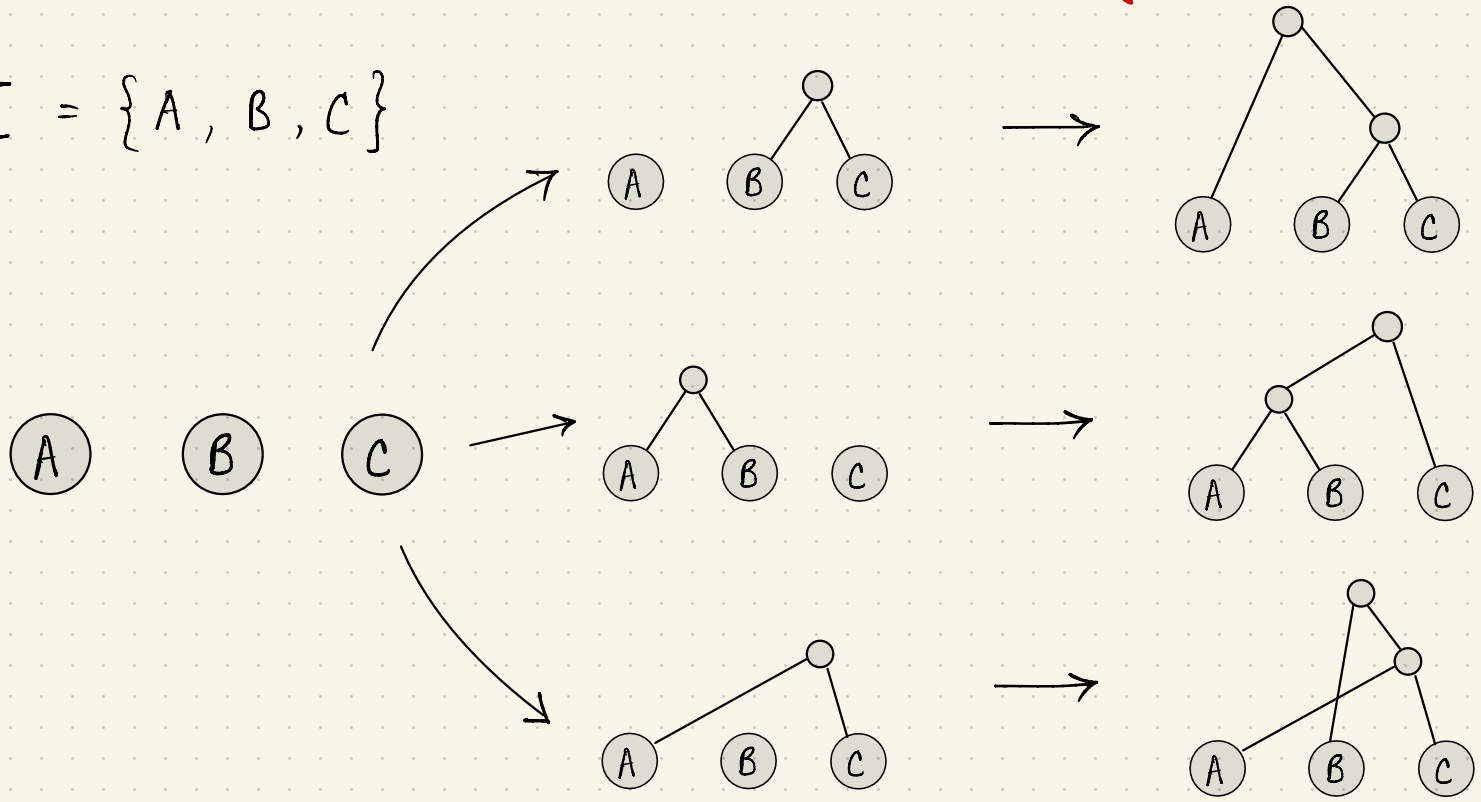
$\Sigma = \{A, B, C\}$



BUILDING A TREE

Which pair of trees is "safe" to merge?

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

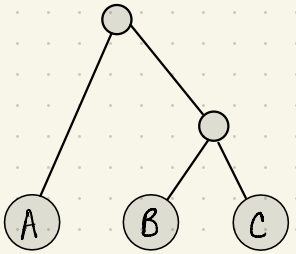


BUILDING A TREE

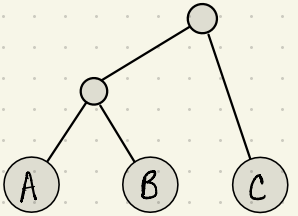
Which pair of trees is "safe" to merge?

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

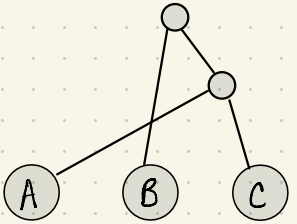
$p_a + 2p_b + 2p_c$



$2p_a + 2p_b + p_c$



$2p_a + p_b + 2p_c$

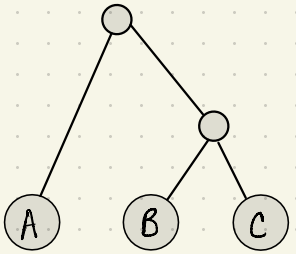


BUILDING A TREE

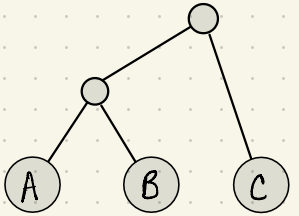
Which pair of trees is "safe" to merge?

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

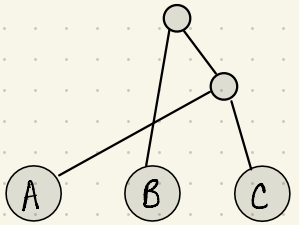
$(p_a + p_b + p_c) + (p_b + p_c)$



$(p_a + p_b + p_c) + (p_a + p_b)$



$(p_a + p_b + p_c) + (p_a + p_c)$

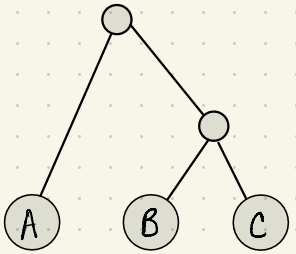


BUILDING A TREE

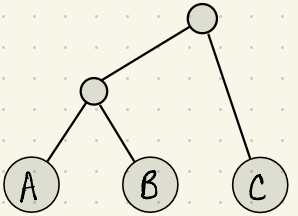
Which pair of trees is "safe" to merge?

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

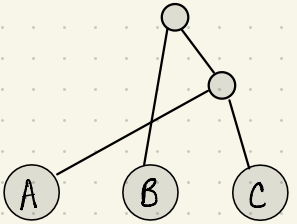
$(p_a + p_b + p_c) + (p_b + p_c)$



$(p_a + p_b + p_c) + (p_a + p_b)$



$(p_a + p_b + p_c) + (p_a + p_c)$

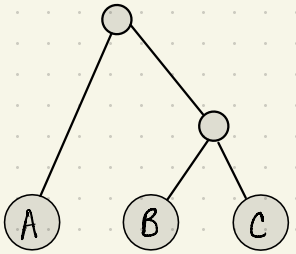


BUILDING A TREE

Which pair of trees is "safe" to merge?

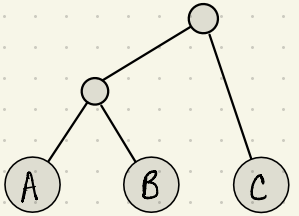
$\Sigma = \{A, B, C\}$

$(p_a + p_b + p_c) + (p_b + p_c)$

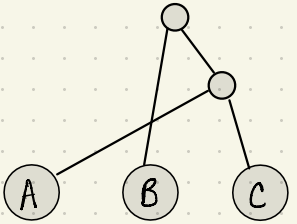


The first merge should involve the two least-frequent symbols

$(p_a + p_b + p_c) + (p_a + p_b)$



$(p_a + p_b + p_c) + (p_a + p_c)$



BUILDING A TREE

Which pair of trees is "safe" to merge?

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

How to merge?