COL 351 : Analysis & Design of Algorithms

Lecture 18

Greedy Algorithms II : Huffman's algorithm

Sept 04, 2024 | Rohit Vaish

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

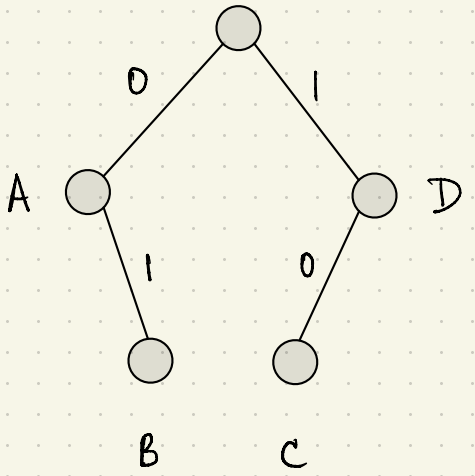
$$\sum_{i \in \Sigma} p_i \cdot \text{number of bits used to encode } i$$

BINARY CODES ⟷ LABELED BINARY TREES

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

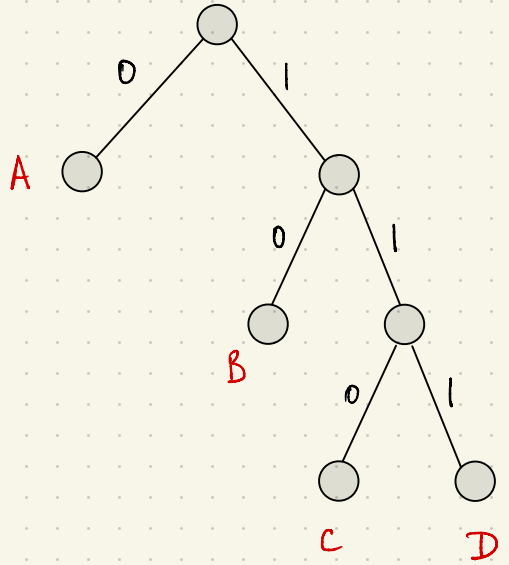
Symbol	variable length
A	0
B	01
C	10
D	1

⟷



PREFIX-FREE CODES AS TREES

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

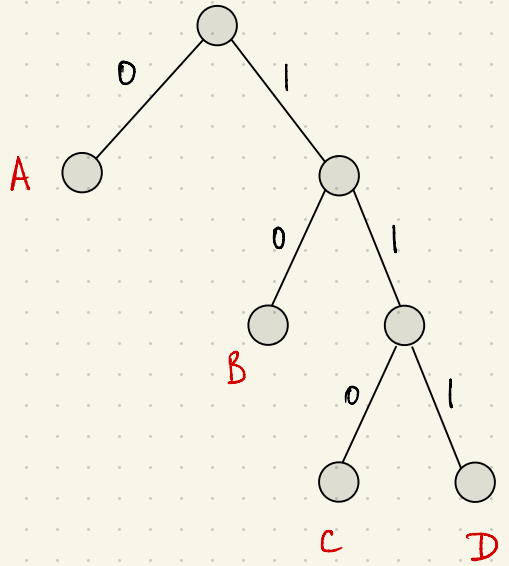


PREFIX-FREE CODES AS TREES

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

* Encoding length of symbol $i \in \Sigma$

  = depth of leaf labeled i in the tree

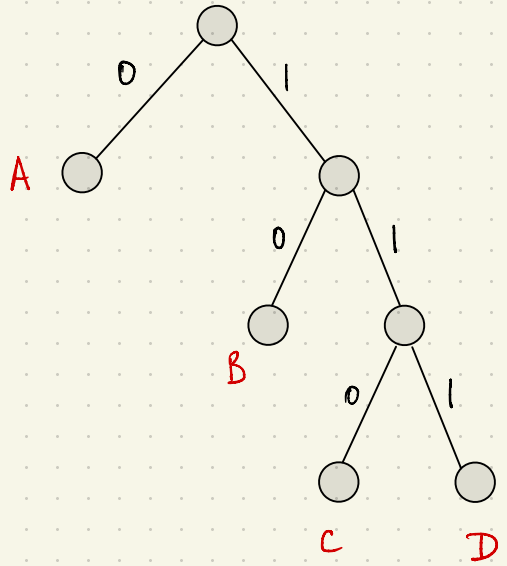


PREFIX-FREE CODES AS TREES

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

* 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.



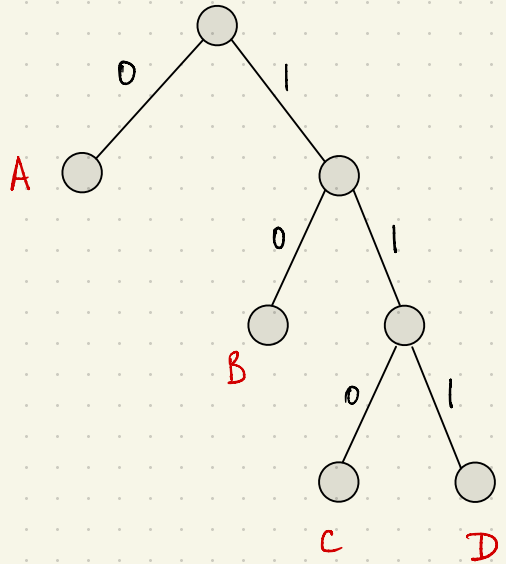
PREFIX-FREE CODES AS TREES

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

* 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 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

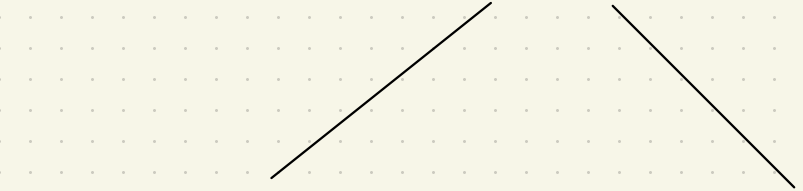
input: a nonnegative frequency p_i for each symbol i of alphabet Σ of size $|\Sigma| = n \geq 2$

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

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

BUILDING A TREE

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



Top down

(divide and conquer)

Bottom up

(greedy)

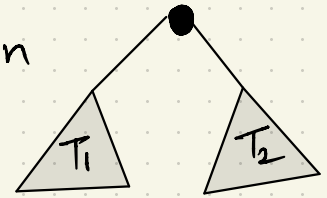
TOP DOWN APPROACH

[Shannon - Fano encoding]

TOP DOWN APPROACH

[Shannon - Fano encoding]

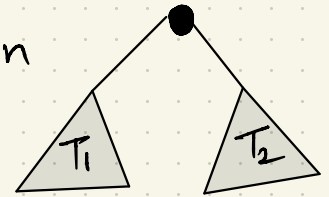
* Divide and conquer
* partition Σ into Σ_1, Σ_2 each with $\approx 50\%$ of total frequency
* recursively compute T_1 for Σ_1, T_2 for Σ_2, return



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



* Suboptimal ☹ [Tutorial sheet 6]

HUFFMAN'S GREEDY ALGORITHM

HUFFMAN'S GREEDY ALGORITHM

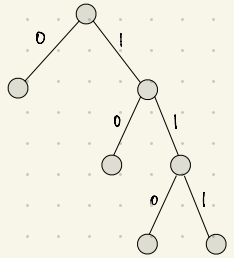
leveraging partial knowledge of optimal solution

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree
(only at leaves)

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)

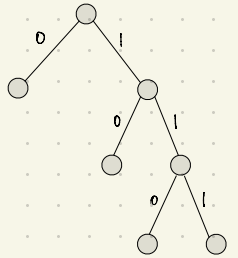
Want
- an optimal Σ tree T^*
- a labeling of T^*

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)



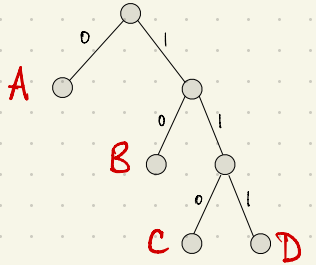
Want
- an optimal $\sum$ tree T^*
  - if we knew this
- a labeling of T^*

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)



Want

- an optimal Σ tree T^* — if we knew this
- a labeling of T^* — Can we determine this?



LABELING AN OPTIMAL SOLUTION

Lemma: Let u and v be leaves of T^* such that $\text{depth}(u) < \text{depth}(v)$. Further, suppose in the labeling of T^* corresponding to an optimal prefix code, u is labeled $a \in \Sigma$ and v is labeled $b \in \Sigma$. Then, $p_a \geq p_b$.

LABELING AN OPTIMAL SOLUTION

Lemma: Let u and v be leaves of T^* such that $\text{depth}(u) < \text{depth}(v)$. Further, suppose in the labeling of T^* corresponding to an optimal prefix code, u is labeled $a \in \Sigma$ and v is labeled $b \in \Sigma$. Then, $p_a \geq p_b$.

Proof: Exercise

Hint: Apply exchange argument to $L(T^*) := \sum_{i \in \Sigma} p_i \cdot$ depth of leaf labeled i in T^*.

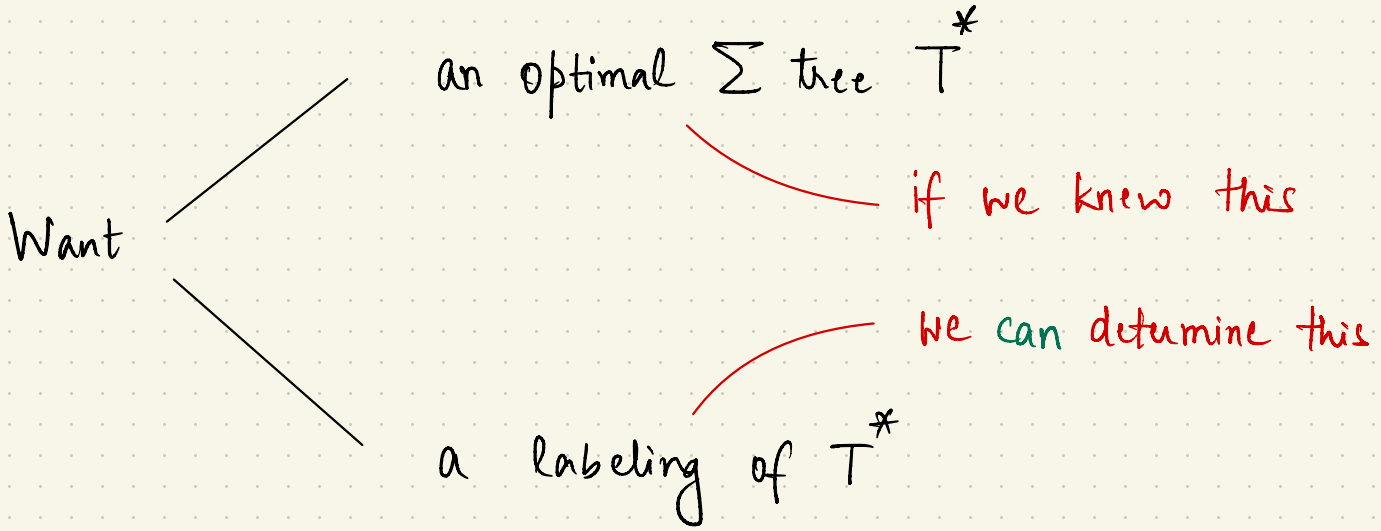
LABELING AN OPTIMAL SOLUTION

Lemma: Let u and v be leaves of T^* such that $\text{depth}(u) < \text{depth}(v)$. Further, suppose in the labeling of T^* corresponding to an optimal prefix code, u is labeled $a \in \Sigma$ and v is labeled $b \in \Sigma$. Then, $p_a \geq p_b$.

Labeling algorithm:

- Label depth 1 leaves (if any) with the highest-frequency symbols
- Label depth 2 leaves (if any) with the next-highest-frequency symbols
- and so on ...

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)



Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)

Want
- an optimal Σ tree T^* — how to determine this?
- a labeling of T^*

Recall : prefix-free code $\longleftrightarrow$ labeled binary tree (only at leaves)

Want
- an optimal Σ tree T^*
  - how to determine this?
  - brute force : exponentially-many binary trees ☹
- a labeling of T^*

Recall labeling algorithm (given T^*):

- Label depth 1 leaves (if any) with the highest-frequency symbols
- Label depth 2 leaves (if any) with the next-highest-frequency symbols
- and so on ...

Recall labeling algorithm (given T^*):

- Label depth 1 leaves (if any) with the highest-frequency symbols
- Label depth 2 leaves (if any) with the next-highest-frequency symbols
- and so on ...

Lowest frequency symbols are assigned to the deepest leaves!

Recall labeling algorithm (given T^*):

- Label depth 1 leaves (if any) with the highest-frequency symbols
- Label depth 2 leaves (if any) with the next-highest-frequency symbols
- and so on ...

Lowest frequency symbols are assigned to the deepest leaves!

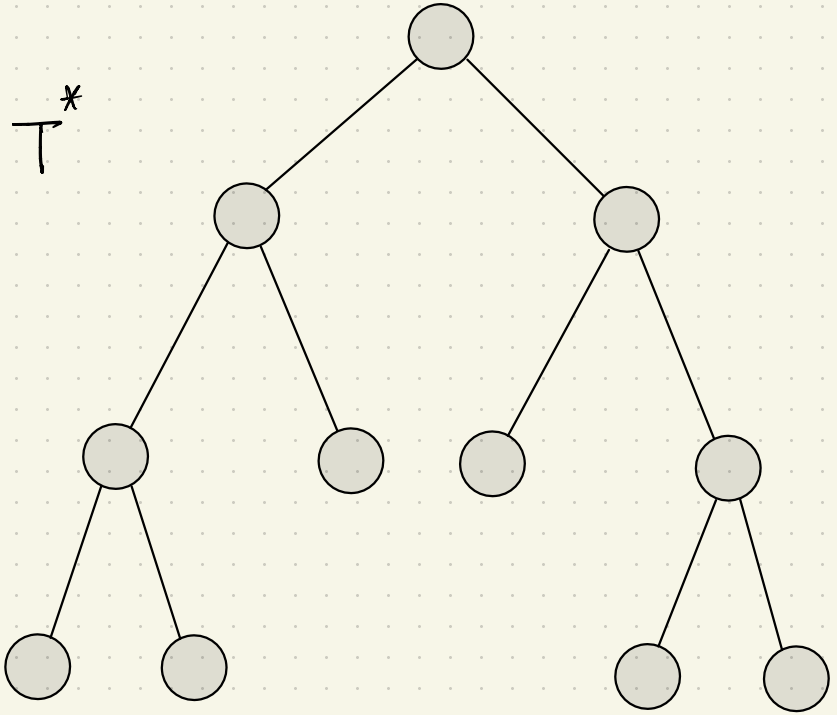
Recall labeling algorithm (given T^*):

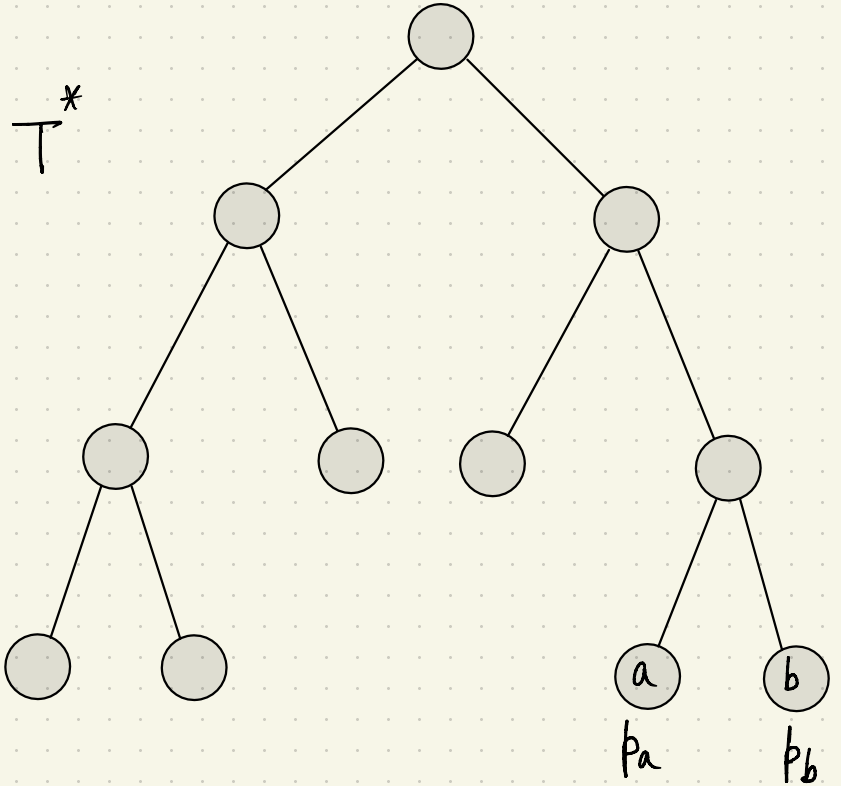
- Label depth 1 leaves (if any) with the highest-frequency symbols
- Label depth 2 leaves (if any) with the next-highest-frequency symbols
- and so on ...

Lowest frequency symbols are assigned to the deepest leaves!

Check: If u is a leaf at maximum depth, then u's sibling is also a leaf at maximum depth.

Corollary: There is an optimal prefix code for input (Σ, p) with corresponding Σ-tree T^* in which the two lowest-frequency symbols are assigned to sibling leaves at maximum depth.

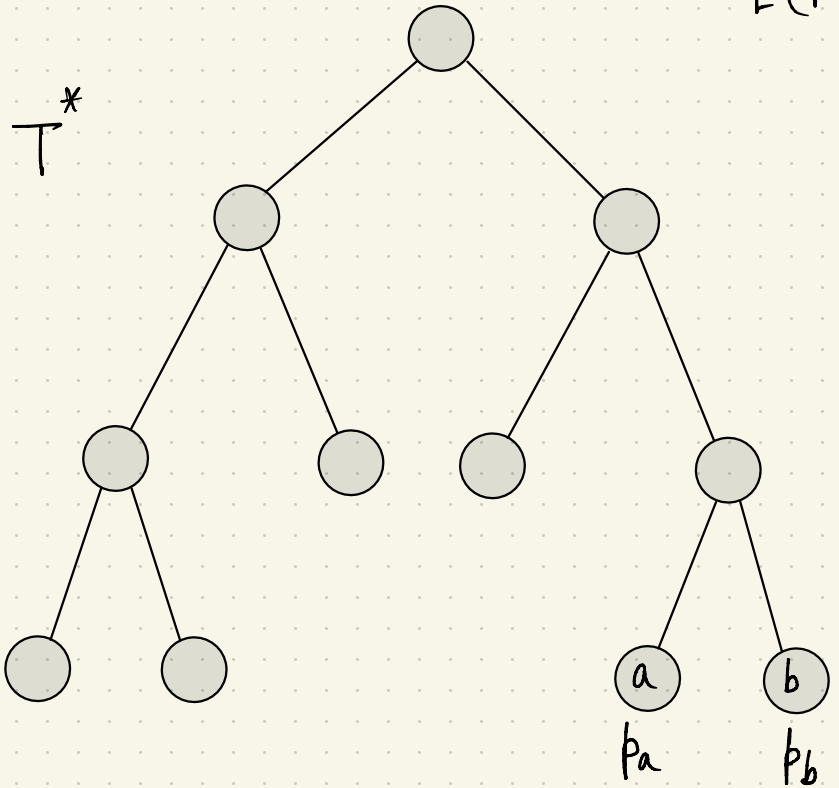

T^*



Two lowest-frequency symbols

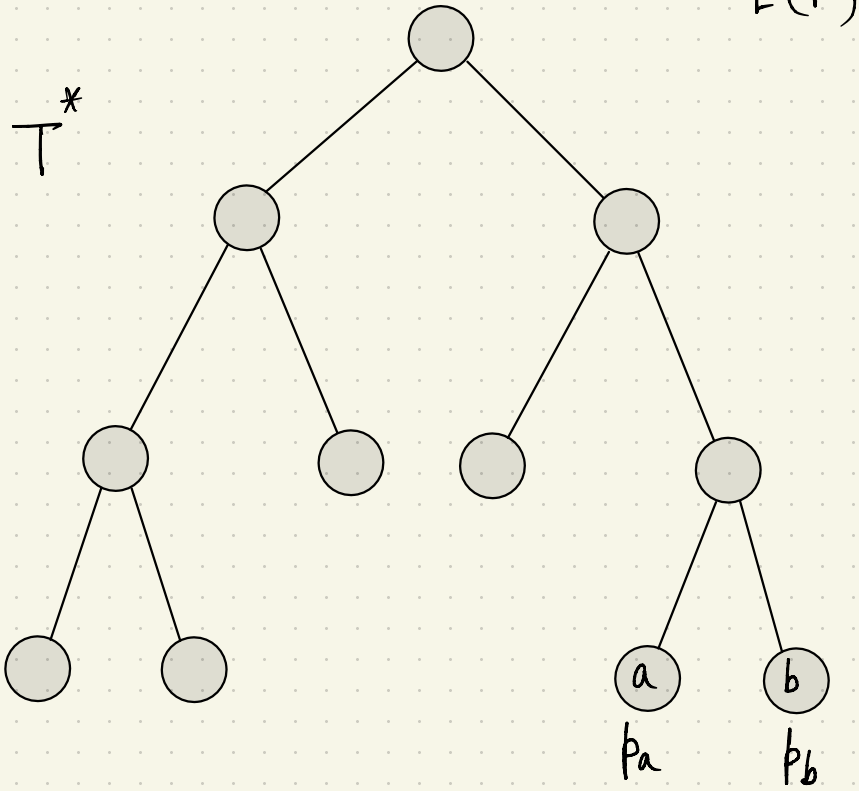
("safely" locked in")

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



Two lowest-frequency symbols

("safely" locked in")

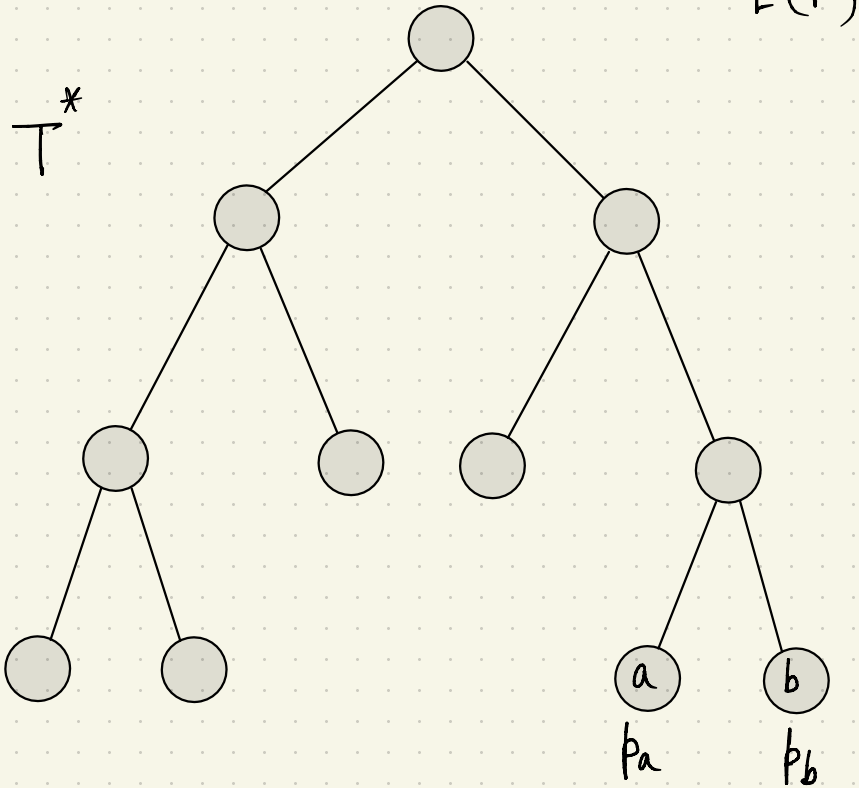


$$L(T^*) = \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i$$

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

Two lowest-frequency symbols

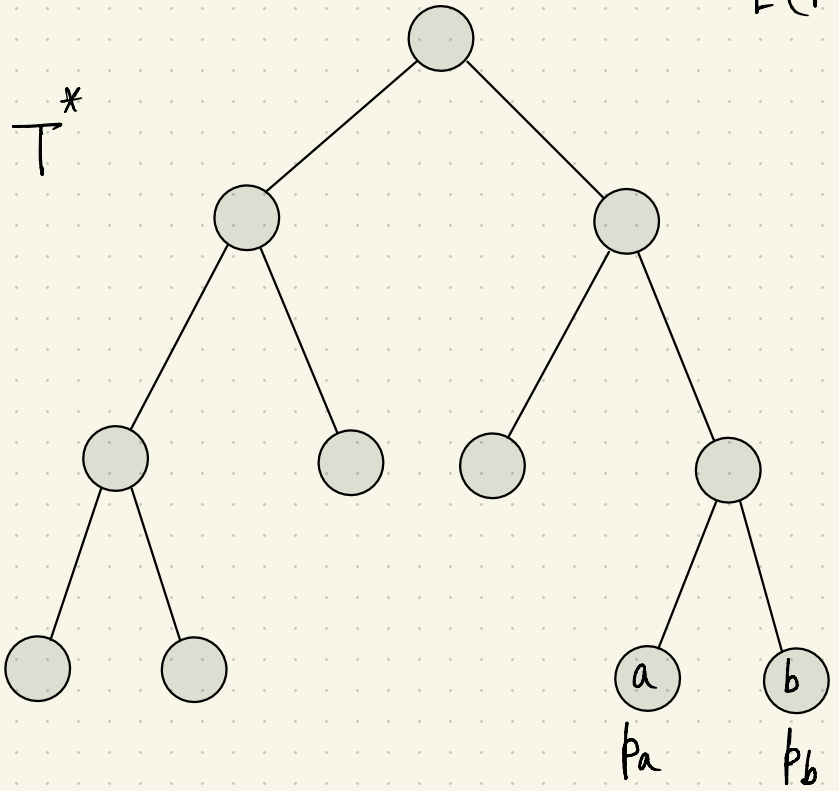
("safely" locked in")



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

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

Two lowest-frequency symbols ("safely locked in")



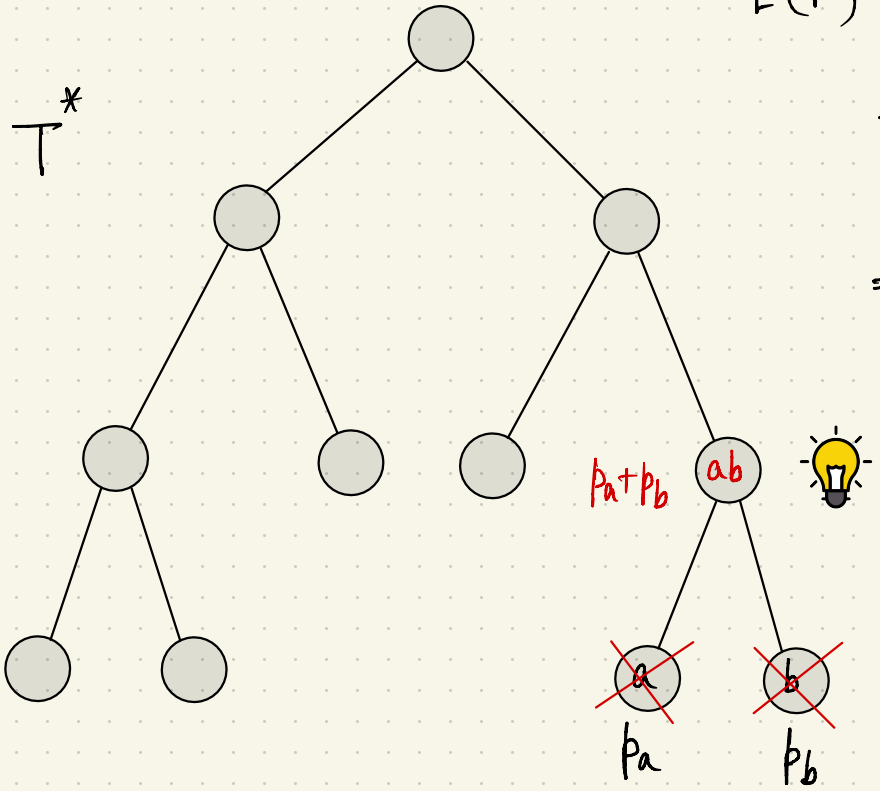
$$L(T^*) = \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i$$

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

Two lowest-frequency symbols

("safely" locked in")

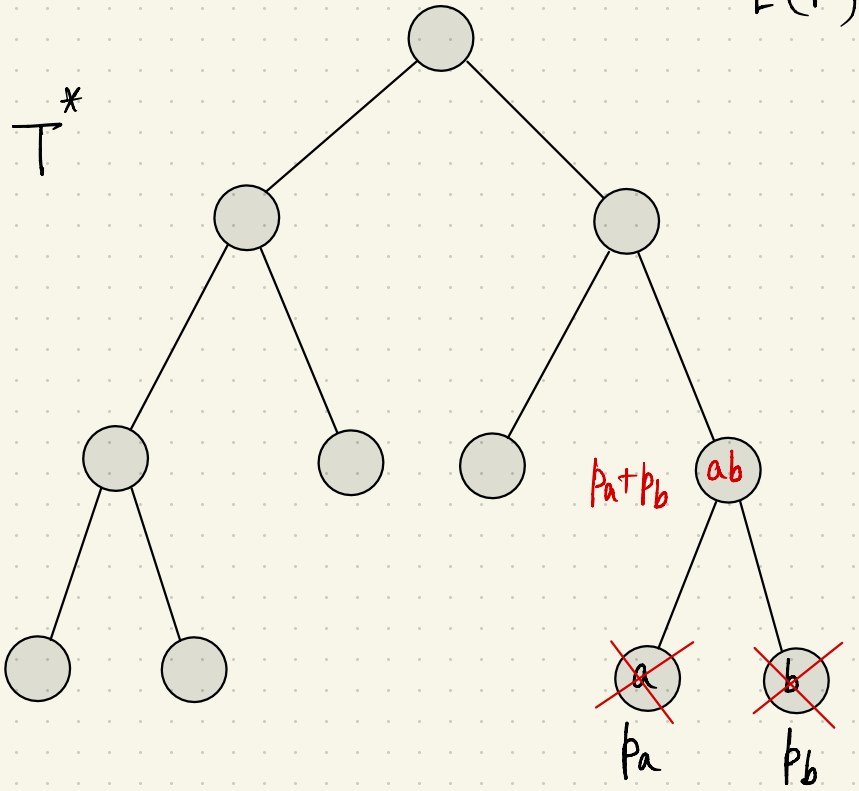


$$L(T^*) = \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i$$

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

Two lowest-frequency symbols ("safely" locked in")



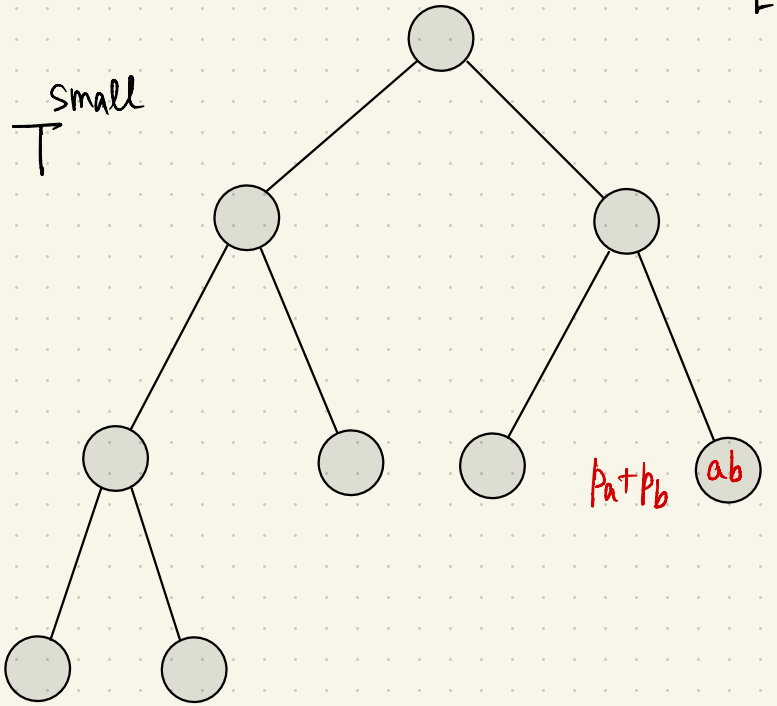
$$L(T^*) = \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i$$

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

$$= p_{ab} \cdot \text{depth}(ab) + p_a + p_b + \text{rest}$$

Two lowest-frequency symbols ("safely locked in")



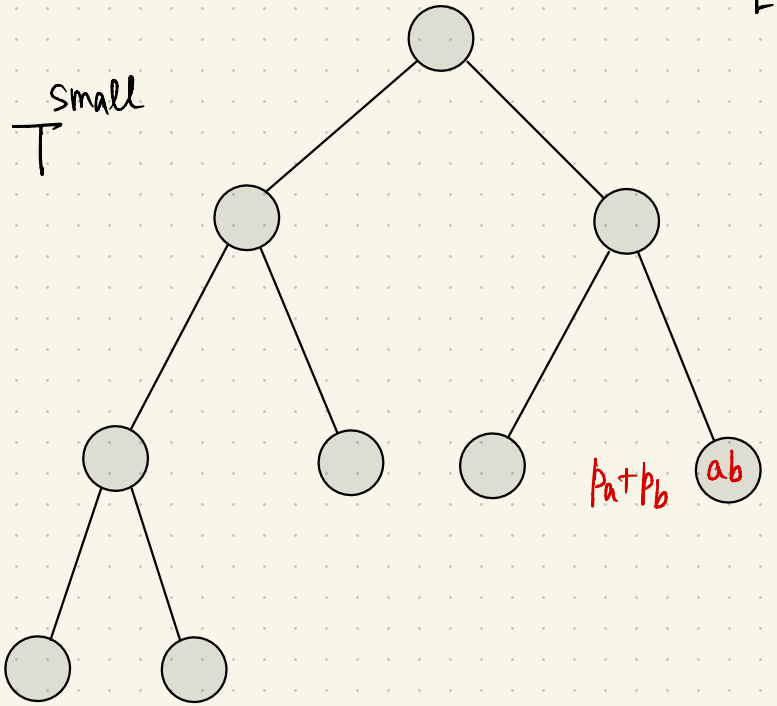
$$L(T^*) = \sum_{i \in \Sigma} p_i \cdot \text{depth of leaf labeled } i$$

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

$$= p_{ab} \cdot \text{depth}(ab) + p_a + p_b + \text{rest}$$

$$= L(T^{small}) + p_a + p_b$$



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

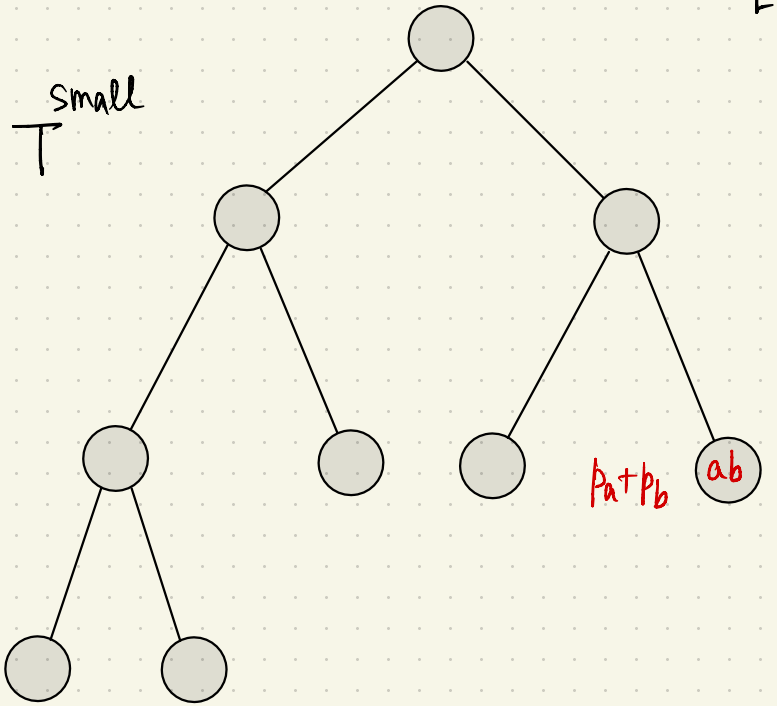
$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

$$= p_{ab} \cdot \text{depth}(ab) + p_a + p_b + \text{rest}$$

$$= L(T^{\text{small}}) + p_a + p_b$$

fixed



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

$$= p_a \cdot \text{depth}(a) + p_b \cdot \text{depth}(b) + \text{rest}$$

$$= (p_a + p_b) \cdot \text{depth}(a) + \text{rest}$$

$$= p_{ab} \cdot \text{depth}(ab) + p_a + p_b + \text{rest}$$

$$= L(T^{\text{small}}) + p_a + p_b$$

fixed

recurse!

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

  return T' after replacing leaf labeled z with unlabeled node and two children labeled x and y.

HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

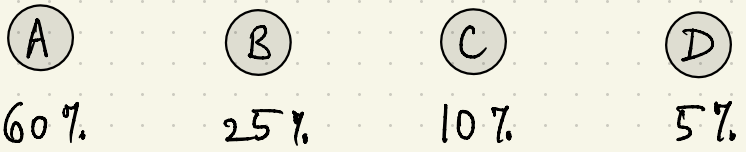
  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

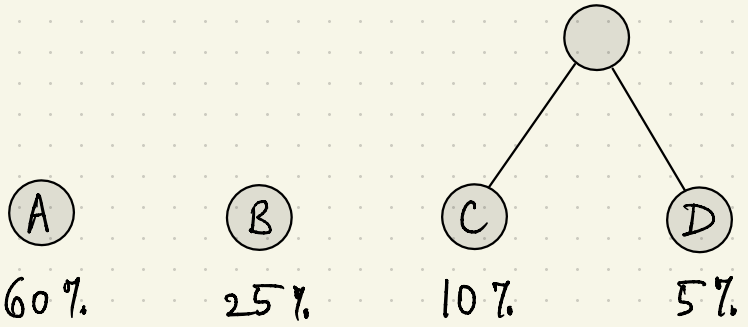
  Recursively construct tree T' for (Σ', p)

  return T' after replacing leaf labeled z with unlabeled node and two children labeled x and y.

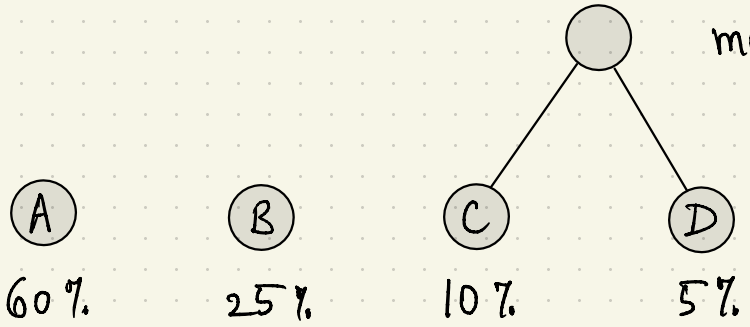
HUFFMAN'S ALGORITHM



HUFFMAN'S ALGORITHM

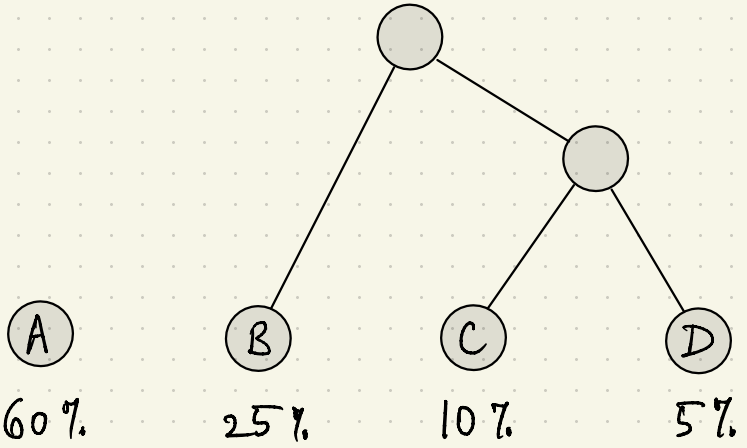


HUFFMAN'S ALGORITHM

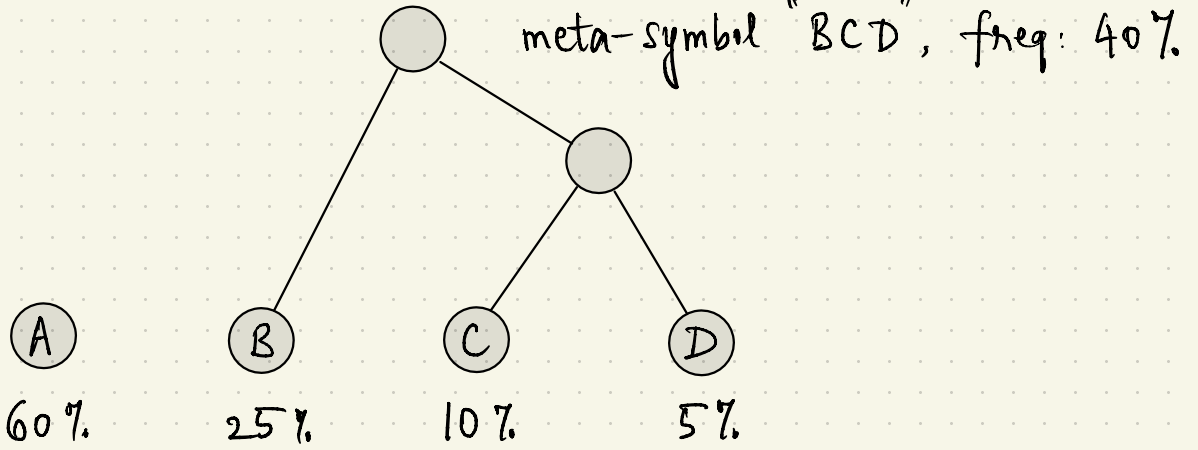


meta-symbol "CD" with frequency = 10+5 = 15%

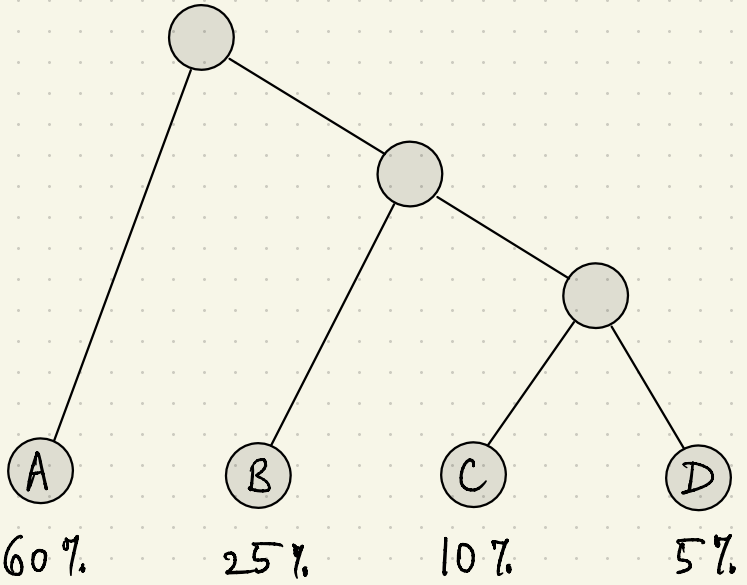
HUFFMAN'S ALGORITHM



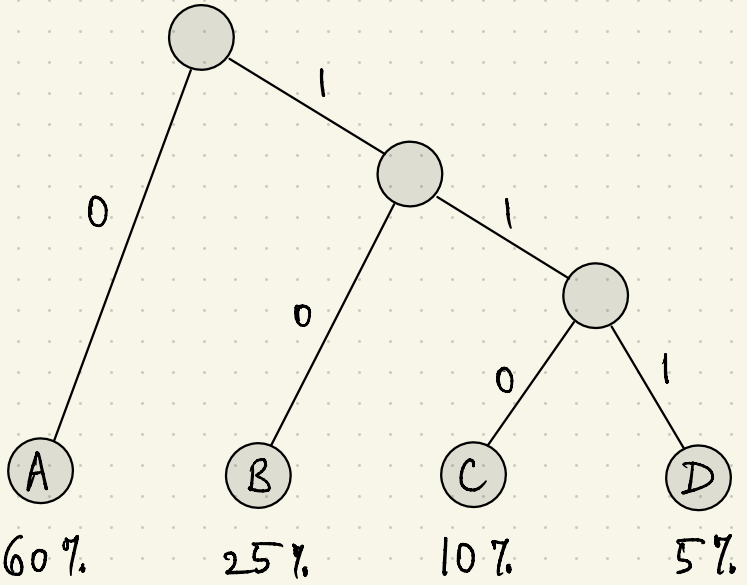
HUFFMAN'S ALGORITHM



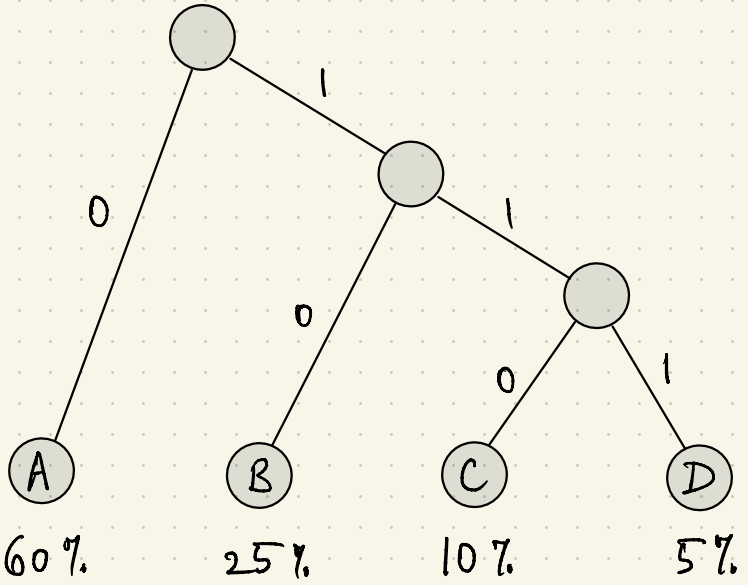
HUFFMAN'S ALGORITHM



HUFFMAN'S ALGORITHM



HUFFMAN'S ALGORITHM



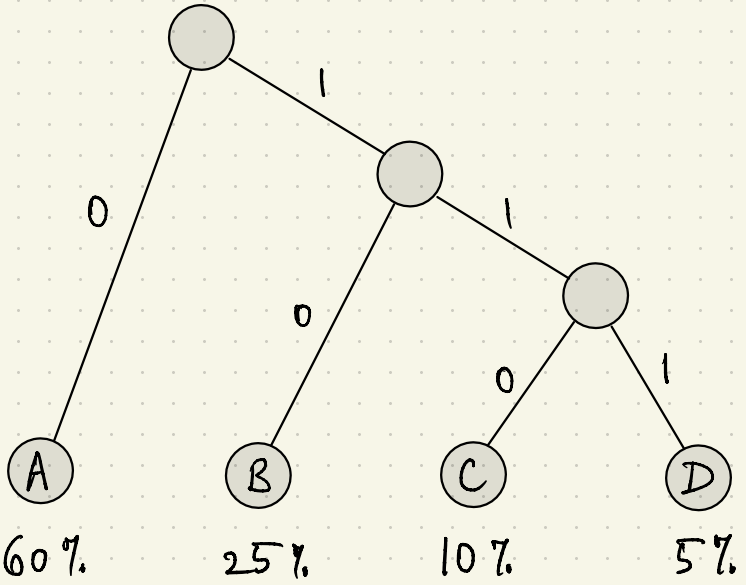
A : 0

B : 01

C : 110

D : 111

HUFFMAN'S ALGORITHM



Huffman code

A : 0

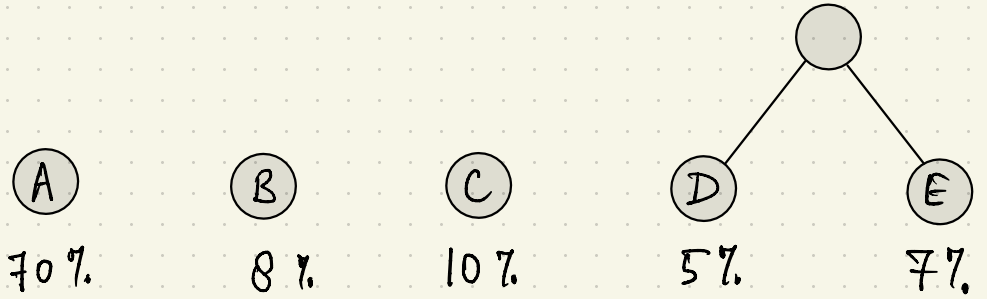
B : 01

C : 110

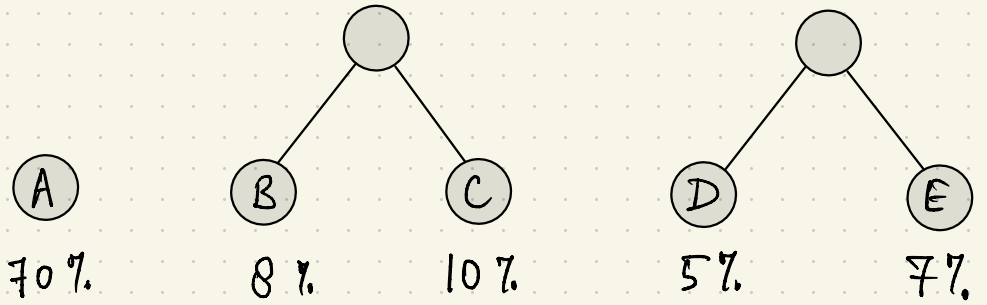
D : 111

HUFFMAN'S ALGORITHM

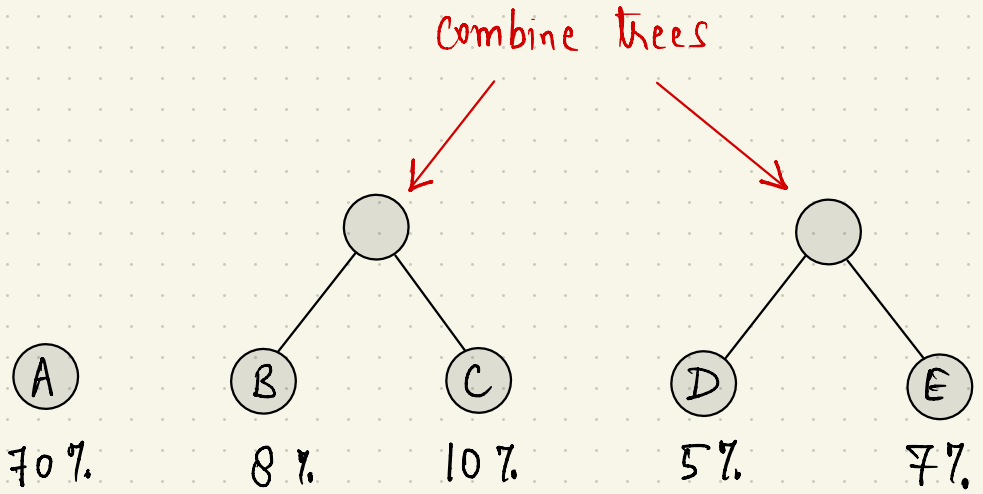
HUFFMAN'S ALGORITHM



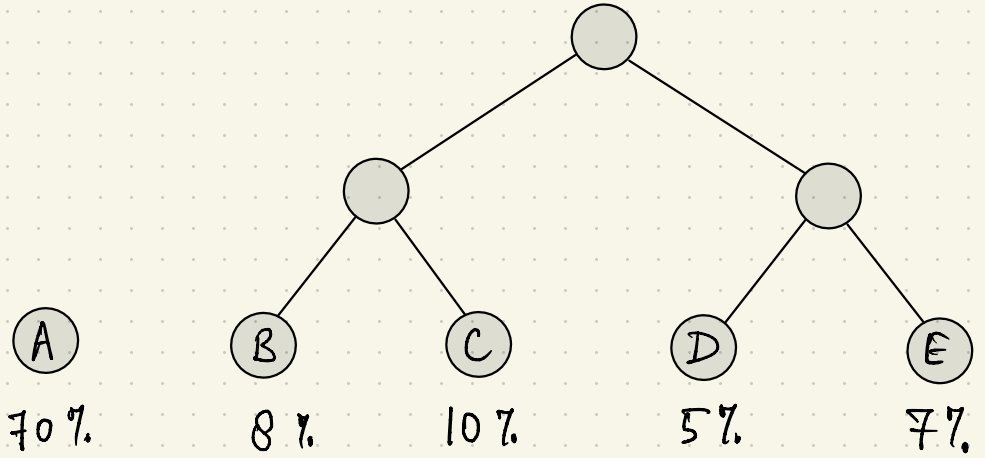
HUFFMAN'S ALGORITHM



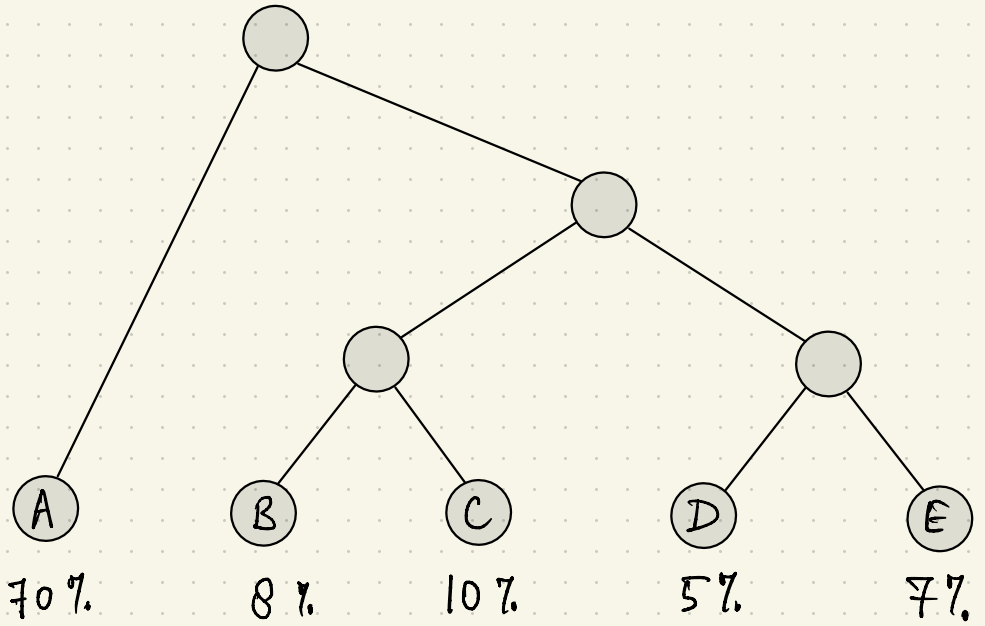
HUFFMAN'S ALGORITHM



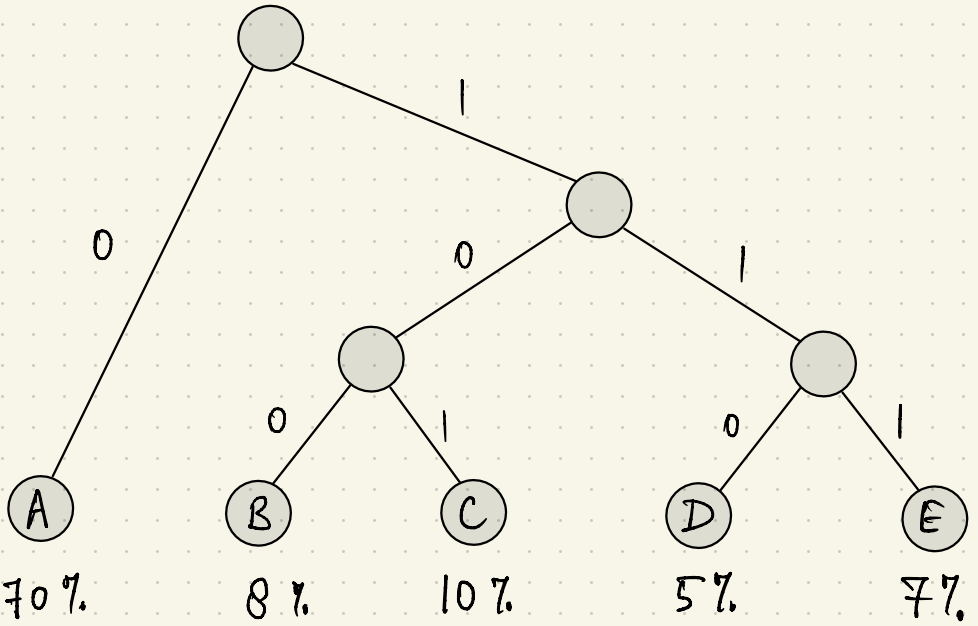
HUFFMAN'S ALGORITHM



HUFFMAN'S ALGORITHM



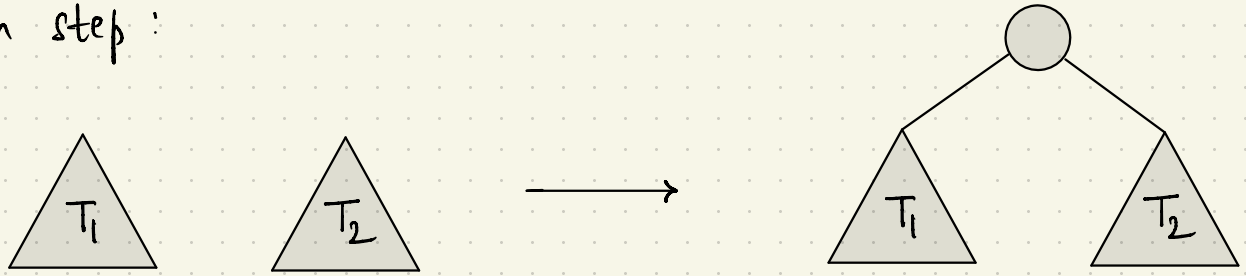
HUFFMAN'S ALGORITHM



A: 0
B: 100
C: 101
D: 110
E: 111

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

At each step:



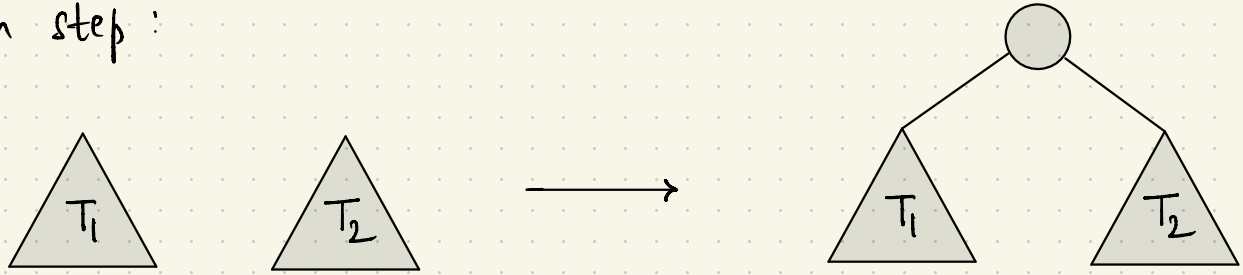
after each merge

👍 # trees in the forest decreases by 1

👎 average depth of leaves increases

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

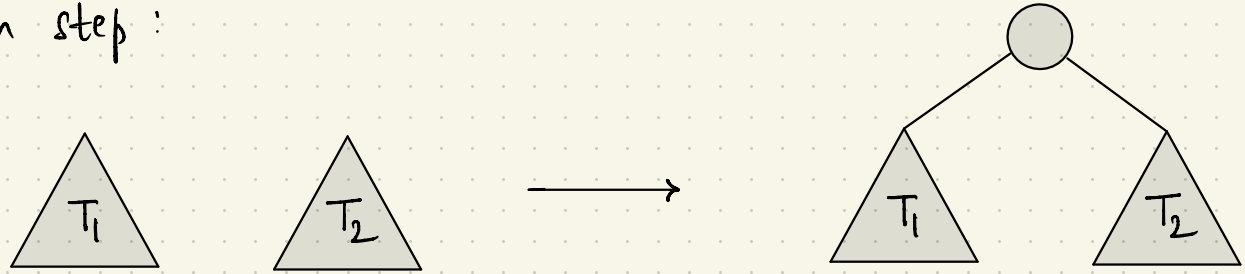
At each step:



$$\text{increase in average leaf-depth} = \sum_{a \in T_1} p_a + \sum_{b \in T_2} p_b$$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

At each step:



$$\text{increase in average leaf-depth} = \sum_{a \in T_1} p_a + \sum_{b \in T_2} p_b$$

Huffman's greedy criterion

Merge the pair of trees that causes minimum possible increase in average leaf depth.

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

// main loop

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

// main loop

While F has at least two trees

$T_1 := \operatorname{argmin}_{T \in F} P(T)$, $T_2 := \operatorname{argmin}_{T \in F \setminus T_1} P(T)$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

// main loop

While F has at least two trees

$T_1 := \operatorname{argmin}_{T \in F} P(T)$, $T_2 := \operatorname{argmin}_{T \in F | T_1} P(T)$

$T_3 :=$ merge of T_1 and T_2

$P(T_3) := P(T_1) + P(T_2)$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

// main loop

While F has at least two trees

$T_1 := \underset{T \in F}{\operatorname{argmin}}\, P(T)$, $T_2 := \underset{T \in F \setminus T_1}{\operatorname{argmin}}\, P(T)$

$T_3 :=$ merge of T_1 and T_2

$P(T_3) := P(T_1) + P(T_2)$

$F := F \cup T_3 \setminus \{T_1, T_2\}$

AN ITERATIVE VIEW OF HUFFMAN'S ALGORITHM

input: alphabet Σ, frequencies $\{p_a\}_{a \in \Sigma}$

output: Σ-tree with minimum average leaf depth

// initialize

for each $a \in \Sigma$

$T_a :=$ tree with one node labeled "a"

$P(T_a) := p_a$

$F := \{T_a\}_{a \in \Sigma}$ (forest)

// invariant:

$\forall T \in F \quad P(T) := \sum_{a \in T} p_a$

// main loop

While F has at least two trees

$T_1 := \underset{T \in F}{\operatorname{argmin}}\ P(T)$, $T_2 := \underset{T \in F \setminus T_1}{\operatorname{argmin}}\ P(T)$

$T_3 :=$ merge of T_1 and T_2

$P(T_3) := P(T_1) + P(T_2)$

$F := F \cup T_3 \setminus \{T_1, T_2\}$

return unique tree in F.

Running time of Huffman's algorithm

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters — $O(n)$

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters — O(n)

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p) — n-1 calls

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters — $O(n)$

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p) — $n-1$ calls

$O(n^2)$ overall

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

$O(\log n)$ with heap

~~$O(n)$~~

$n-1$ calls

$O(n^2)$ overall

RUNNING TIME OF HUFFMAN'S ALGORITHM

if $|\Sigma| = 2$

  encode one letter using 0 and other using 1

else

  $x, y \leftarrow$ two lowest-frequency letters

  Create new alphabet Σ' by merging x and y into z with $p_z := p_x + p_y$.

  Recursively construct tree T' for (Σ', p)

$O(\log n)$ with heap

~~$O(n)$~~

$n-1$ calls

$O(n \log n)$

~~$O(n^2)$~~ overall

Correctness of Huffman's algorithm

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: (by strong induction on $|\Sigma| = n$)

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: (by strong induction on $|\Sigma| = n$)

Base case:

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: (by strong induction on $|\Sigma| = n$)

Base case: $n = 2$

Huffman uses one bit per symbol — optimal!

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Induction step: Huffman is optimal for all Σ with $|\Sigma| \leq n$.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Induction step: Huffman is optimal for all Σ with $|\Sigma| \leq n$.

Consider Σ such that $|\Sigma| = n+1$.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Induction step: Huffman is optimal for all Σ with $|\Sigma| \leq n$.

Consider Σ such that $|\Sigma| = n+1$.

Huffman's algo merges $x, y \in \Sigma$ into meta-symbol z.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: *Induction step:* Huffman is optimal for all Σ with $|\Sigma| \leq n$.

Consider Σ such that $|\Sigma| = n+1$.

Huffman's algo merges $x, y \in \Sigma$ into meta-symbol z.

Let Σ' be the new alphabet. Thus, $|\Sigma'| = n$.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Induction step: Huffman is optimal for all Σ with $|\Sigma| \leq n$.

Consider Σ such that $|\Sigma| = n+1$.

Huffman's algo merges $x, y \in \Sigma$ into meta-symbol z.

Let Σ' be the new alphabet. Thus, $|\Sigma'| = n$.

Let T' be algorithm's output for $\Sigma' \Rightarrow T'$ is optimal!

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \rightarrow T'$

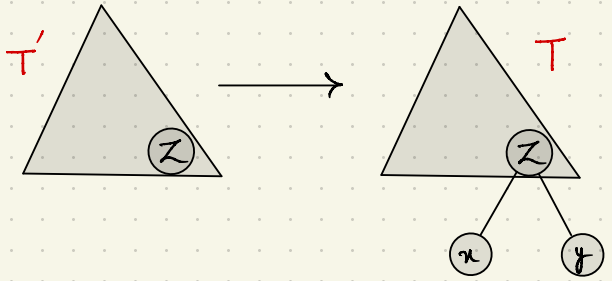
" " " " $\Sigma \rightarrow T$

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \rightarrow T'$

" " " " $\Sigma \rightarrow T$



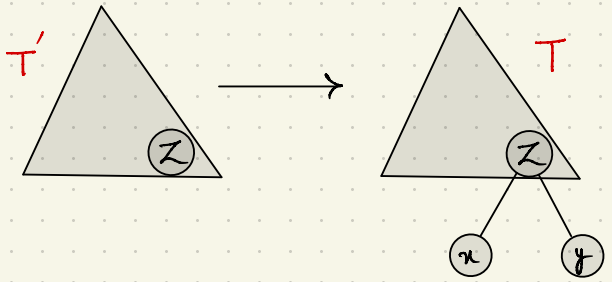
CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$

$$L(T') = L(T) - (p_x + p_y) \quad \text{(exercise)}$$



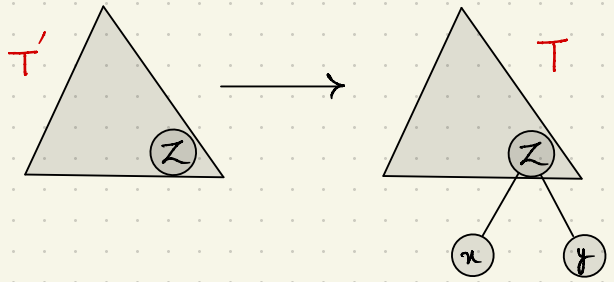
CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$

$$L(T') = L(T) - (p_x + p_y) \quad \text{(exercise)}$$



If T not optimal, then $L(T) > L(S)$ for some tree S

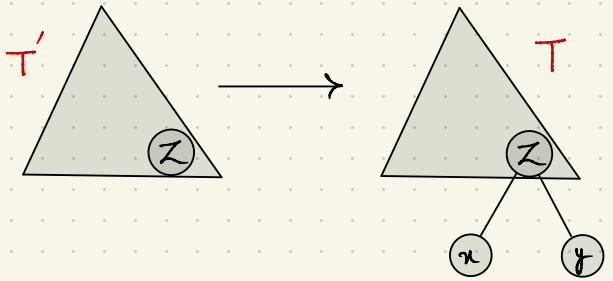
CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$

$$L(T') = L(T) - (p_x + p_y) \quad \text{(exercise)}$$



If T not optimal, then $L(T) > L(S)$ for some tree S

Corollary $\Rightarrow$ x and y are siblings in S.

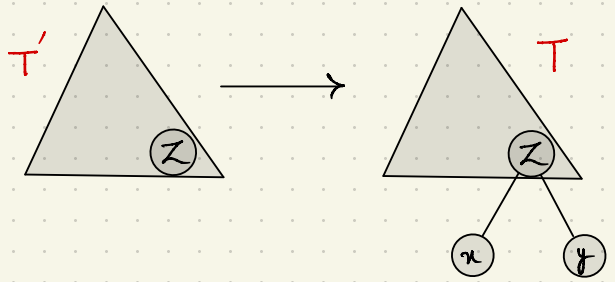
CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$

$L(T') = L(T) - (p_x + p_y)$ (exercise)



If T not optimal, then $L(T) > L(S)$ for some tree S

Corollary $\Rightarrow$ x and y are siblings in S.

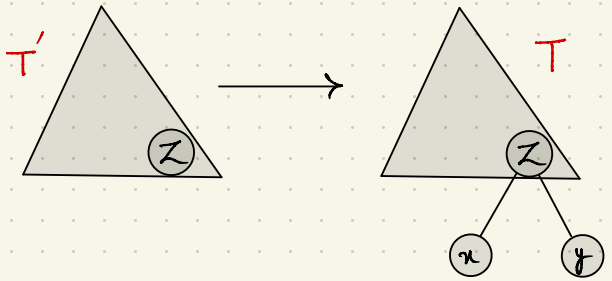
Create S' by "merging" x and y.

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$



$L(T') = L(T) - (p_x + p_y)$ (exercise)

If T not optimal, then $L(T) > L(S)$ for some tree S

Corollary $\Rightarrow$ x and y are siblings in S.

Create S' by "merging" x and y.

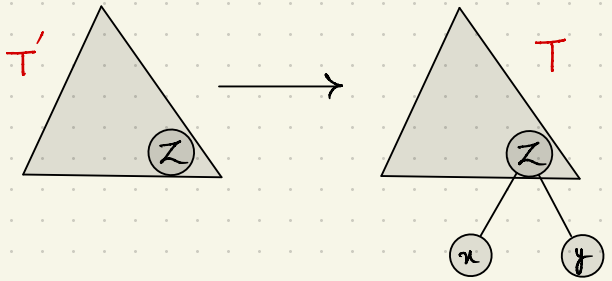
Then, $L(S') < L(T')$. Contradiction!

CORRECTNESS OF HUFFMAN'S ALGORITHM

Theorem: For every alphabet Σ and nonnegative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algorithm returns a prefix-free code with minimum possible average encoding length.

Proof: Algorithm's output for $\Sigma' \longrightarrow T'$

" " " " $\Sigma \longrightarrow T$



$L(T') = L(T) - (p_x + p_y)$ (exercise)

If T not optimal, then $L(T) > L(S)$ for some tree S

Corollary $\Rightarrow$ x and y are siblings in S.

Create S' by "merging" x and y.

Then, $L(S') < L(T')$. Contradiction! ∎

SUMMARY

Goal : Prefix-free code with minimum average encoding length

Key ideas : (1) Reimagining codes as trees

(2) Two lowest-frequency symbols are "locked in" as deepest leaves.

(3) "Merging" into meta-symbol facilitates recursion

Minimum Spanning Trees

SPANNING TREES

SPANNING TREES

A spanning tree of a connected graph

SPANNING TREES

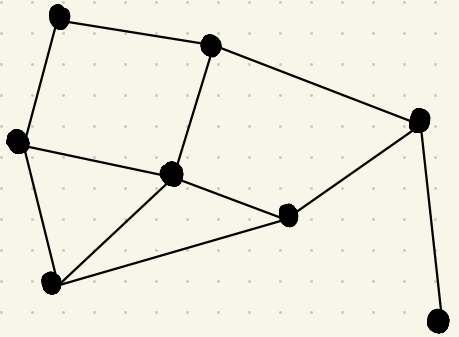
A spanning tree of a connected graph is a subgraph that is itself a tree and

SPANNING TREES

A spanning tree of a connected graph is a subgraph that is itself a tree and whose vertex set is the same as that of the original graph.

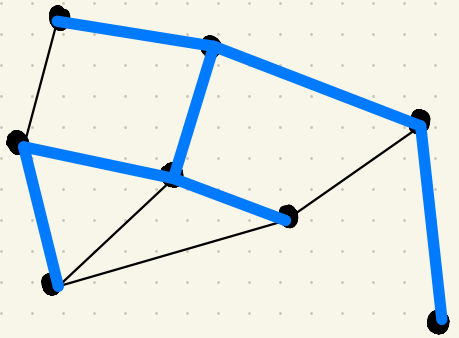
SPANNING TREES

A spanning tree of a connected graph is a subgraph that is itself a tree and whose vertex set is the same as that of the original graph.



SPANNING TREES

A spanning tree of a connected graph is a subgraph that is itself a tree and whose vertex set is the same as that of the original graph.



Theorem: Every connected graph has a spanning tree.

Theorem: Every connected graph has a spanning tree.

Proof: By contradiction.

Suppose, for contradiction, that some connected graph G has no spanning tree.

Theorem: Every connected graph has a spanning tree.

Proof: By contradiction.

Suppose, for contradiction, that some connected graph G has no spanning tree.

Let T be a connected subgraph of G with the same set of vertices and the smallest number of edges possible.

Theorem: Every connected graph has a spanning tree.

Proof: By contradiction.

Suppose, for contradiction, that some connected graph G has no spanning tree.

Let T be a connected subgraph of G with the same set of vertices and the smallest number of edges possible.

↑ T is well-defined because of well ordering principle.

Theorem: Every connected graph has a spanning tree.

Proof: By contradiction.

Suppose, for contradiction, that some connected graph G has no spanning tree.

Let T be a connected subgraph of G with the same set of vertices and the smallest number of edges possible.

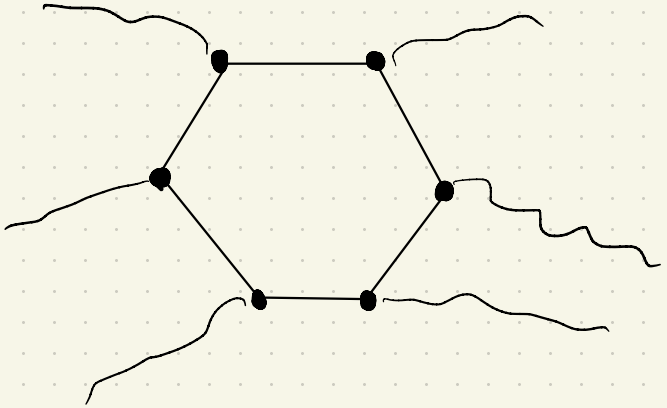
Then, by contradiction assumption, T is NOT a tree.

Theorem: Every connected graph has a spanning tree.

Proof: Therefore, T must have a cycle.

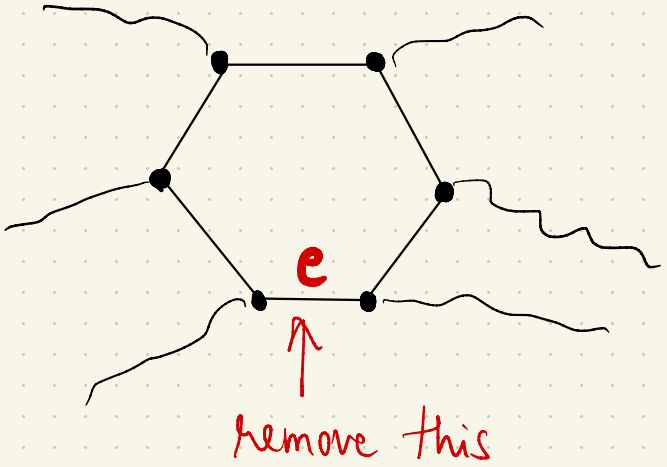
Theorem: Every connected graph has a spanning tree.

Proof: Therefore, T must have a cycle.



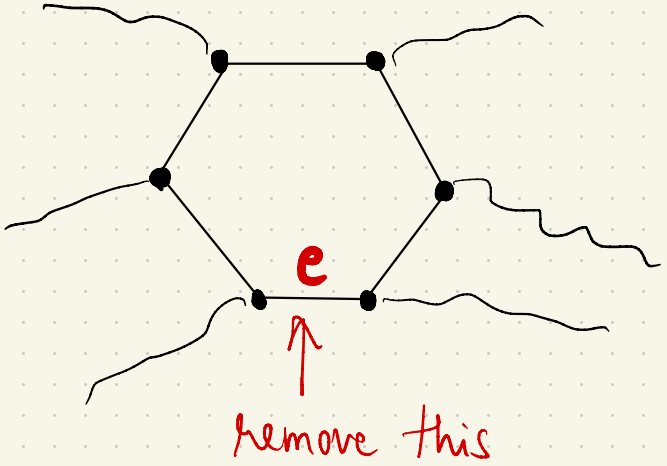
Theorem: Every connected graph has a spanning tree.

Proof: Therefore, T must have a cycle.



Theorem: Every connected graph has a spanning tree.

Proof: Therefore, T must have a cycle.

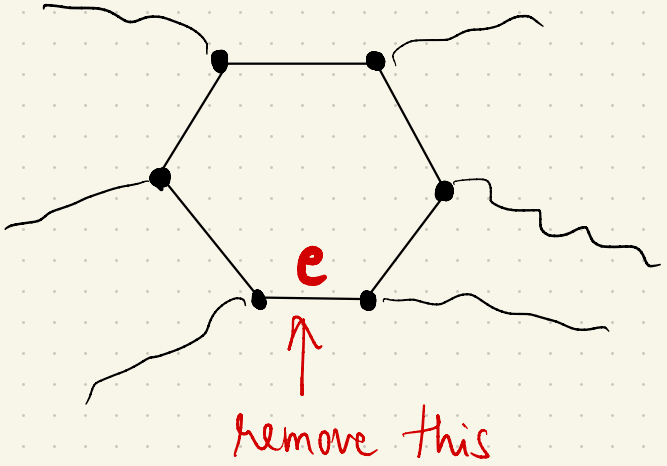


Case I:

does not contain e.

Theorem: Every connected graph has a spanning tree.

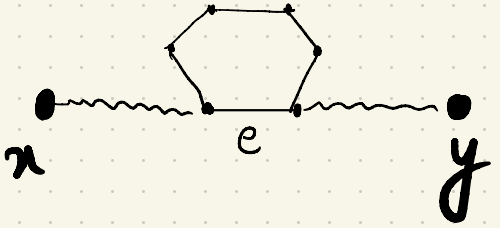
Proof: Therefore, T must have a cycle.



Case I:

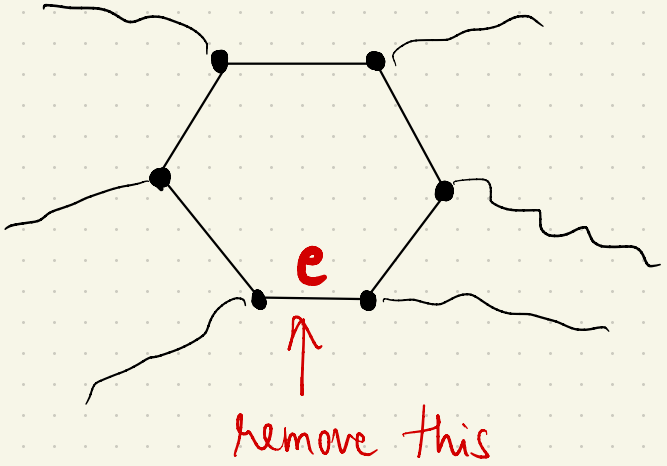
does not contain e.

Case II:



Theorem: Every connected graph has a spanning tree.

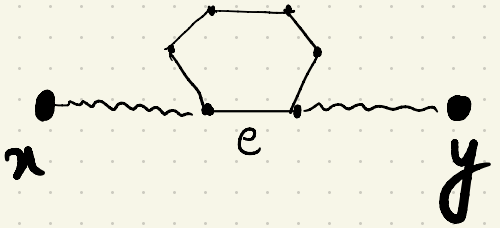
Proof: Therefore, T must have a cycle.



Case I:

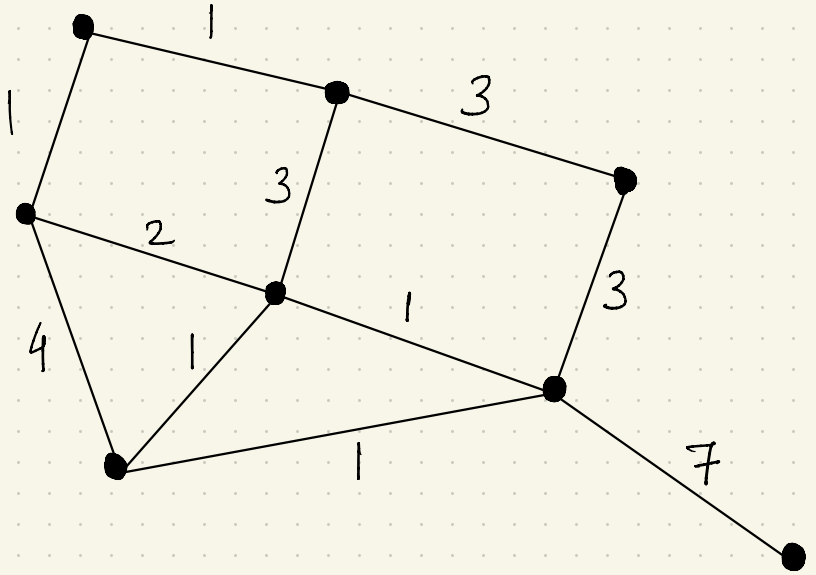
does not contain e.

Case II:

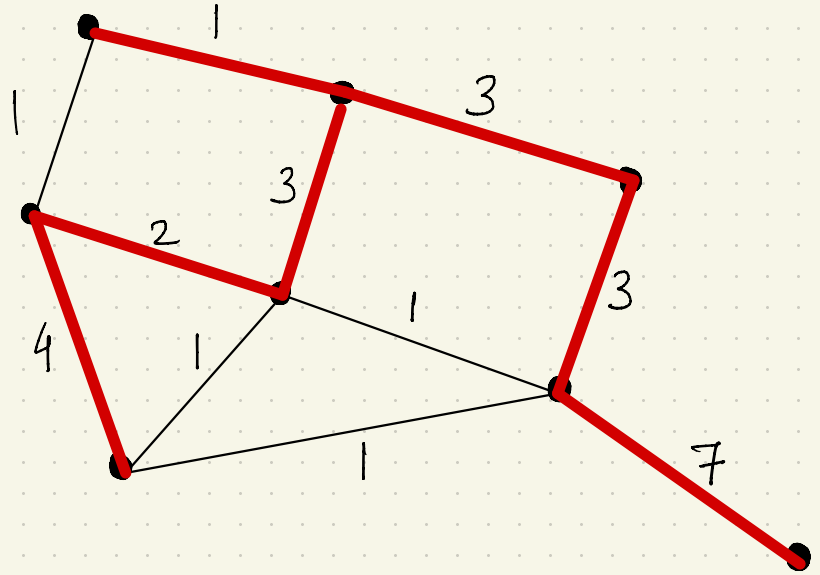


All vertices in T are still connected after removing e.

Minimum weight spanning tree

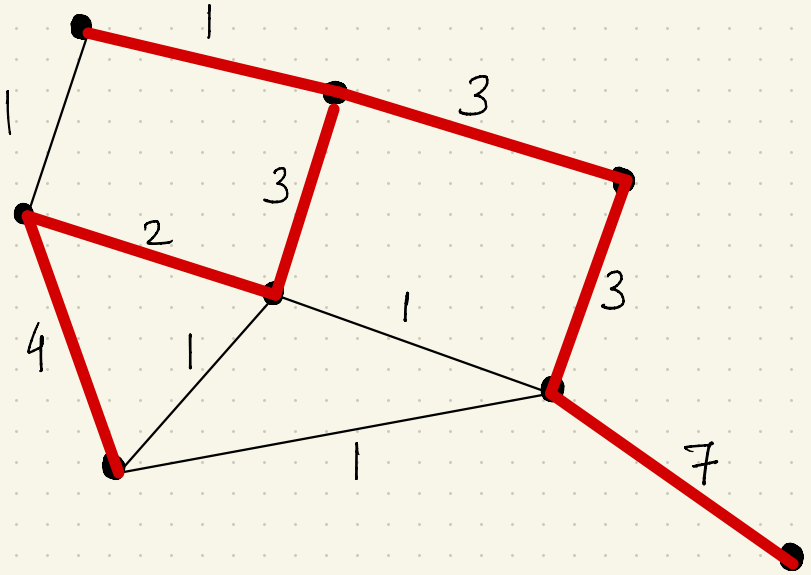


Minimum weight spanning tree



Weight = 1 + 3 + 3 + 3 + 2 + 4 + 7 = 23

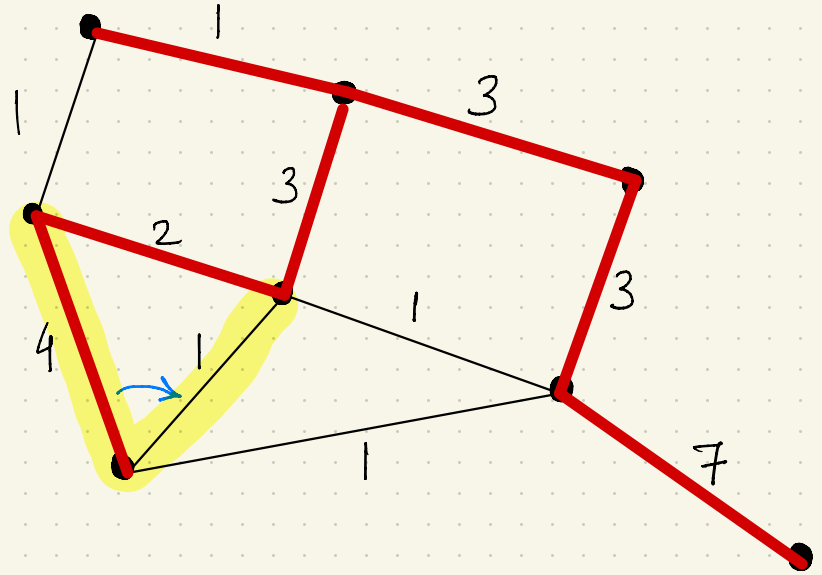
Minimum weight spanning tree



Weight = $1+3+3+3+2+4+7 = 23$

Can we do better?

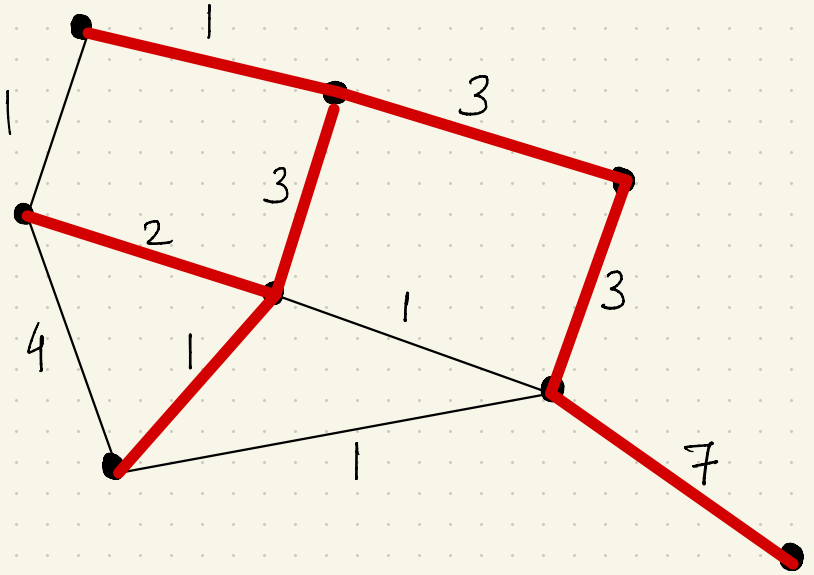
Minimum weight spanning tree



Weight = $1+3+3+3+2+4+7 = 23$

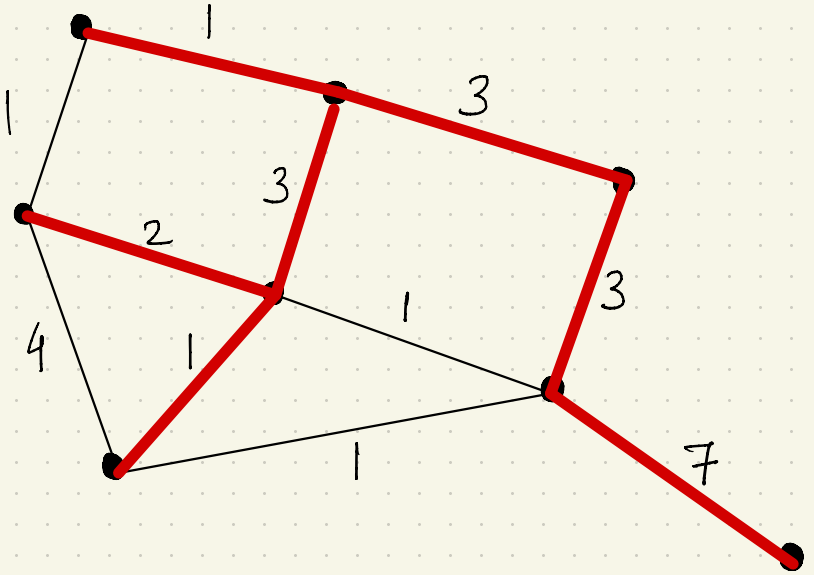
Can we do better?

Minimum weight spanning tree



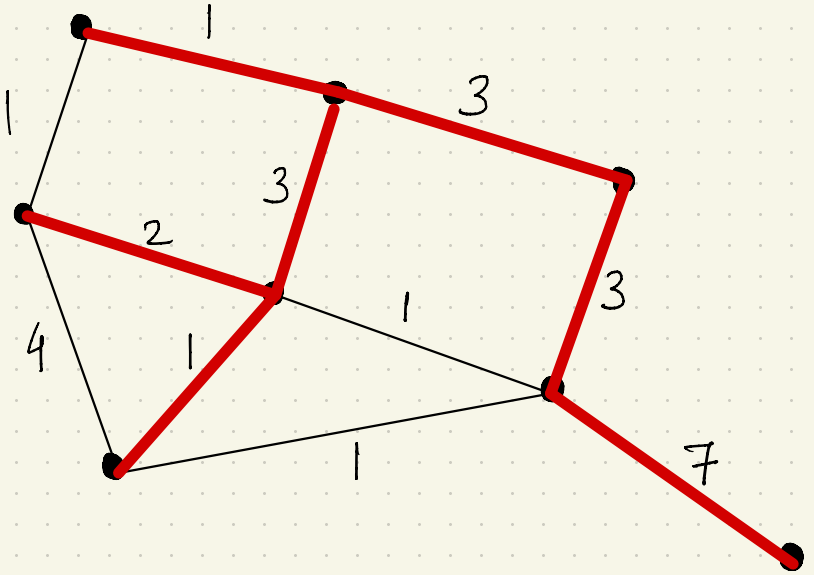
Weight = $1+3+3+3+2+\cancel{4}\ 1+7 = \cancel{23}\ 20$

Minimum weight spanning tree



Weight = 1 + 3 + 3 + 3 + 2 + 1 + 7 = 20

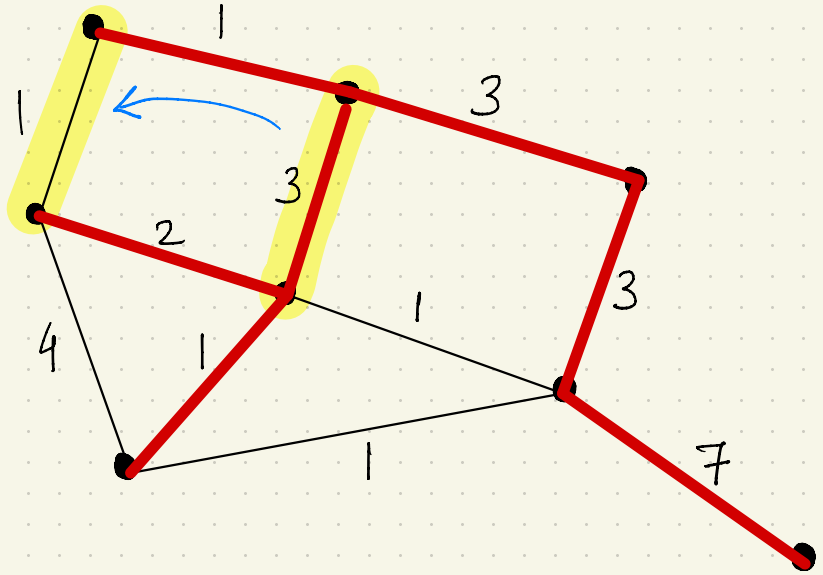
Minimum weight spanning tree



Weight = $1+3+3+3+2+1+7=20$

Can we do better?

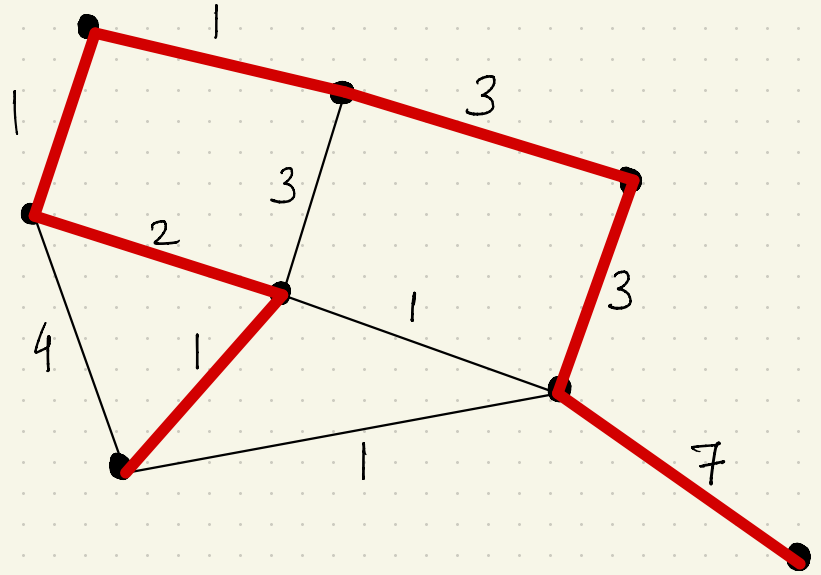
Minimum weight spanning tree



Weight = $1+3+3+3+2+1+7 = 20$

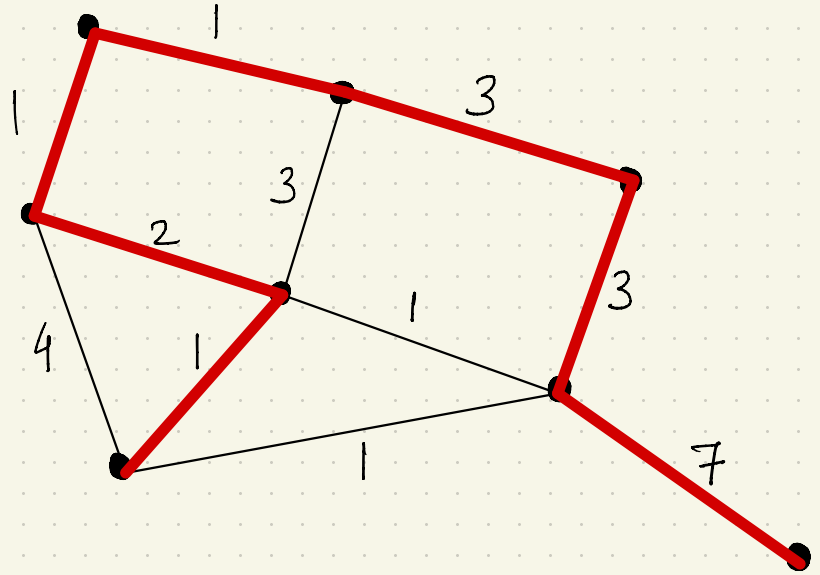
Can we do better?

Minimum weight spanning tree



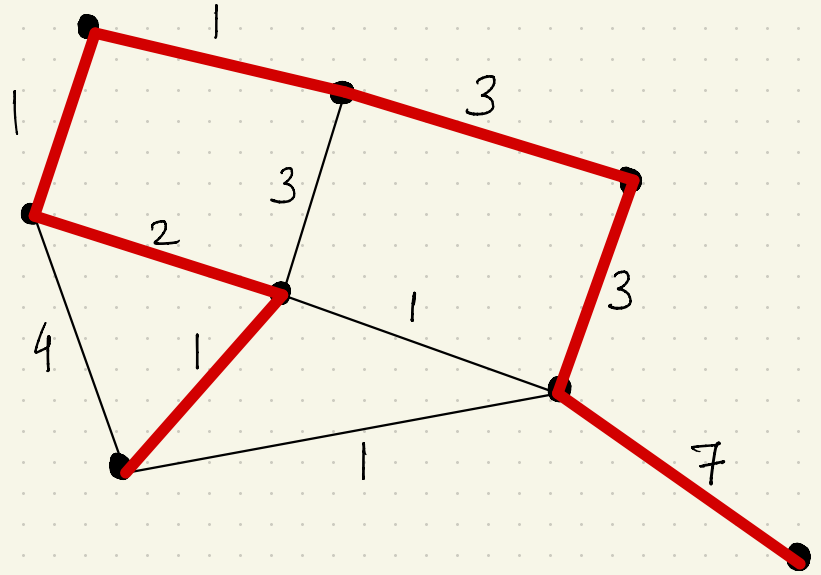
Weight = $1 + \cancel{2}\ 1 + 3 + 3 + 2 + 1 + 7 = \cancel{20}\ 18$

Minimum weight spanning tree



Weight = 1 + 1 + 3 + 3 + 2 + 1 + 7 = 18

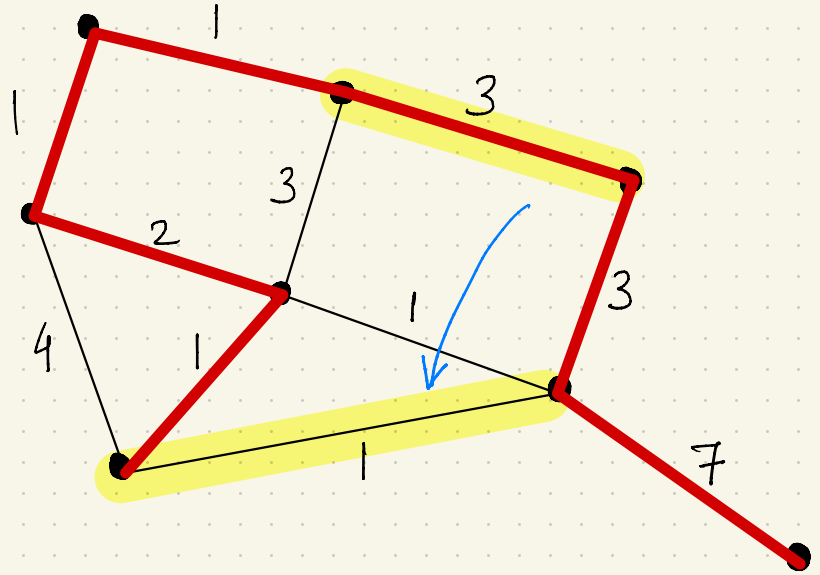
Minimum weight spanning tree



Weight = 1 + 1 + 3 + 3 + 2 + 1 + 7 = 18

Can we do better?

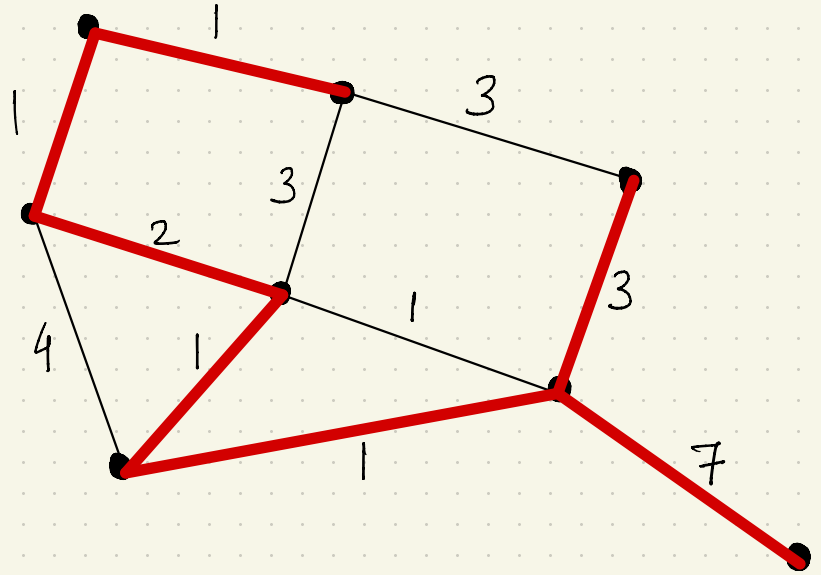
Minimum weight spanning tree



Weight = 1 + 1 + 3 + 3 + 2 + 1 + 7 = 18

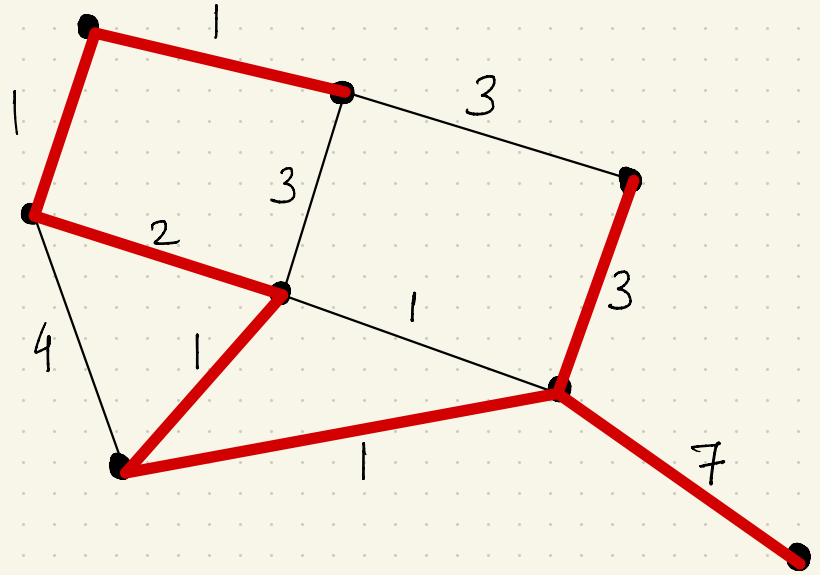
Can we do better?

Minimum weight spanning tree



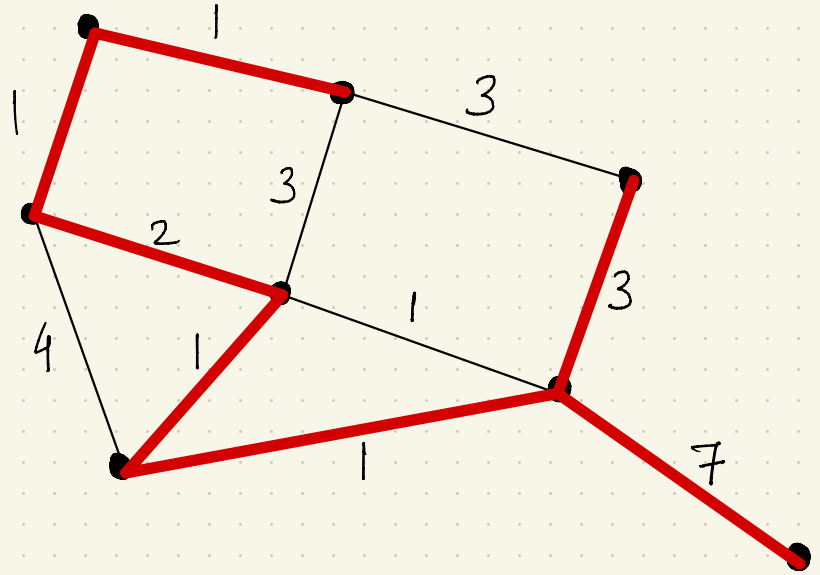
Weight = 1 + 1 + ~~2~~ 1 + 3 + 2 + 1 + 7 = ~~18~~ 16

Minimum weight spanning tree



Weight = 1 + 1 + 1 + 3 + 2 + 1 + 7 = 16

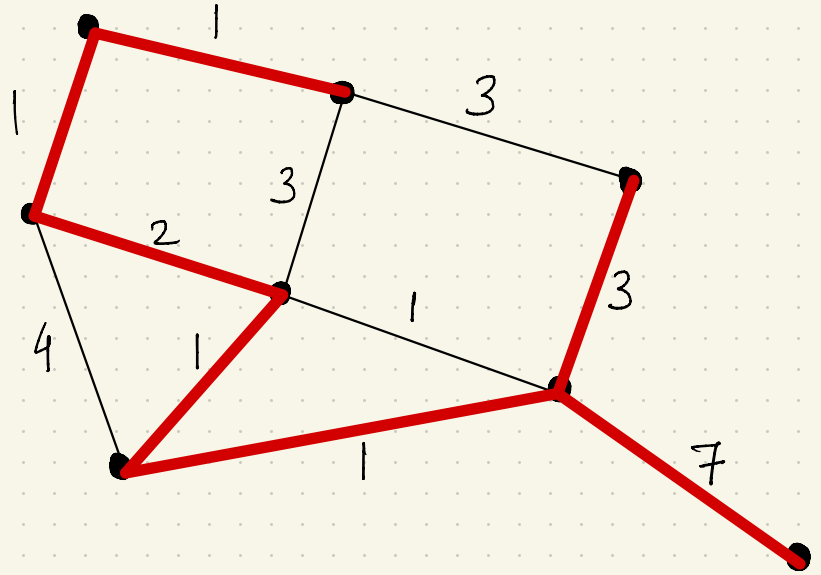
Minimum weight spanning tree



Weight = 1 + 1 + 1 + 3 + 2 + 1 + 7 = 16

Can we do better?

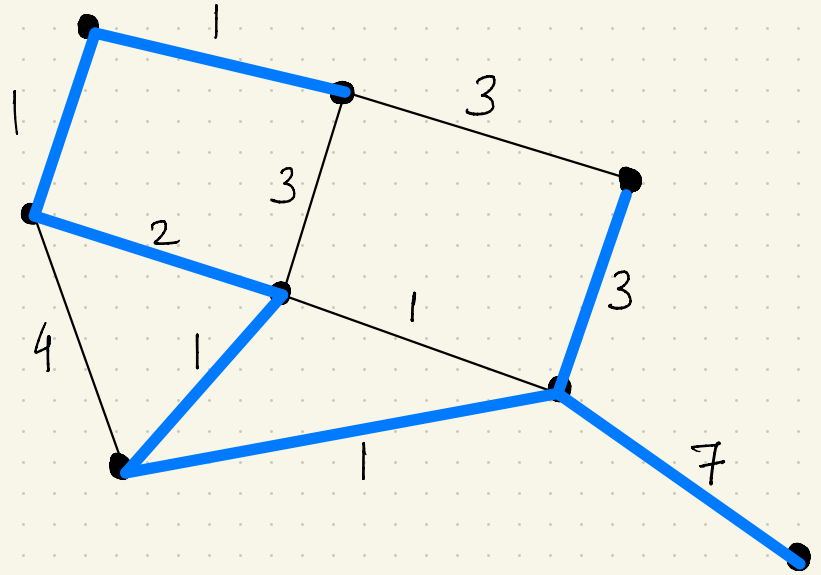
Minimum weight spanning tree



Weight = 1 + 1 + 1 + 3 + 2 + 1 + 7 = 16

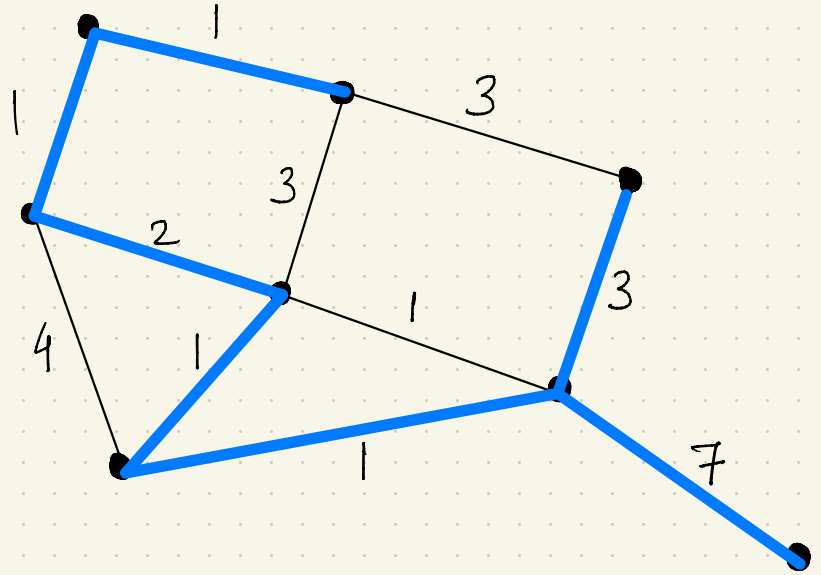
Can we do better? No!

Minimum weight spanning tree



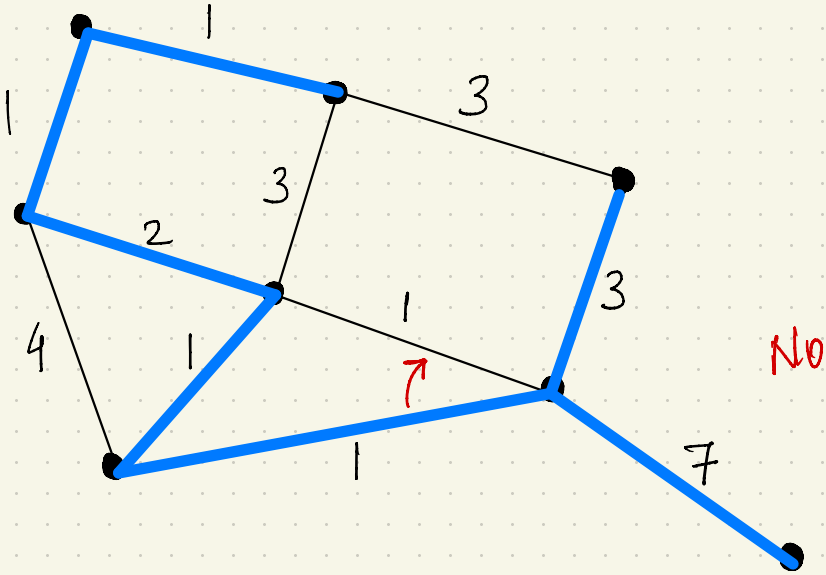
Spanning tree of minimum weight

Minimum weight spanning tree



Weight = 1 + 1 + 1 + 1 + 2 + 3 + 7 = 16

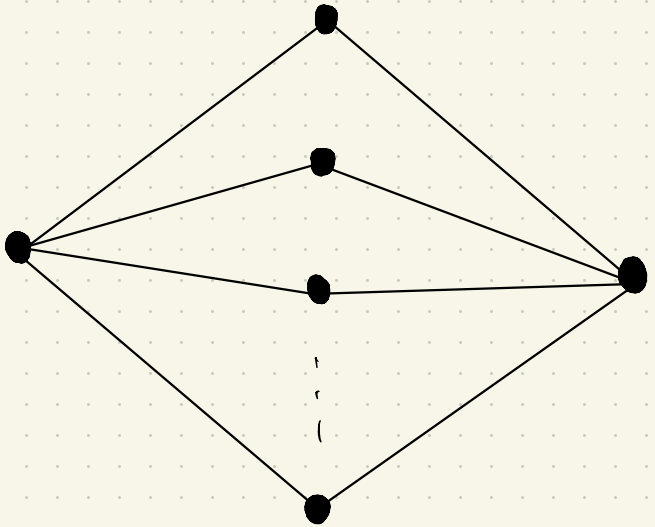
Minimum weight spanning tree



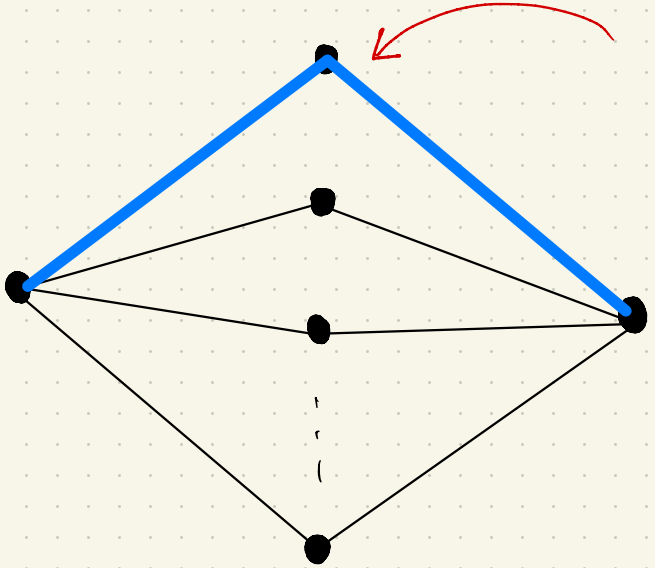
NOT UNIQUE!

Weight = 1 + 1 + 1 + 1 + 2 + 3 + 7 = 16

EXPONENTIALLY MANY SPANNING TREES

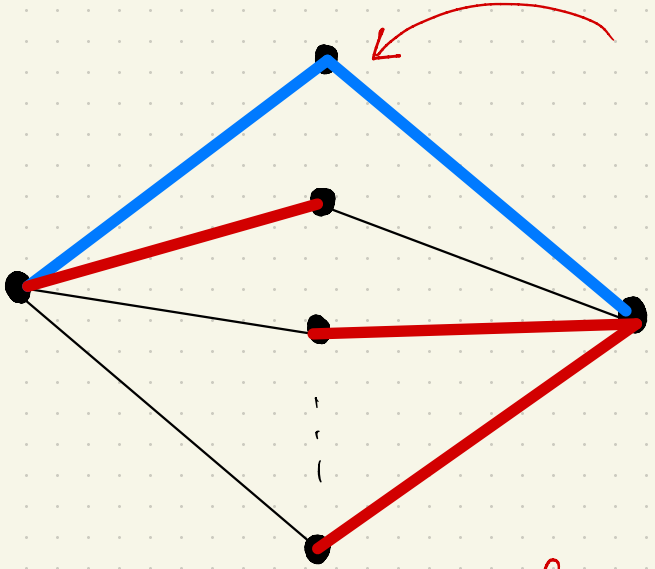


EXPONENTIALLY MANY SPANNING TREES



Some middle node must have both edges in spanning tree

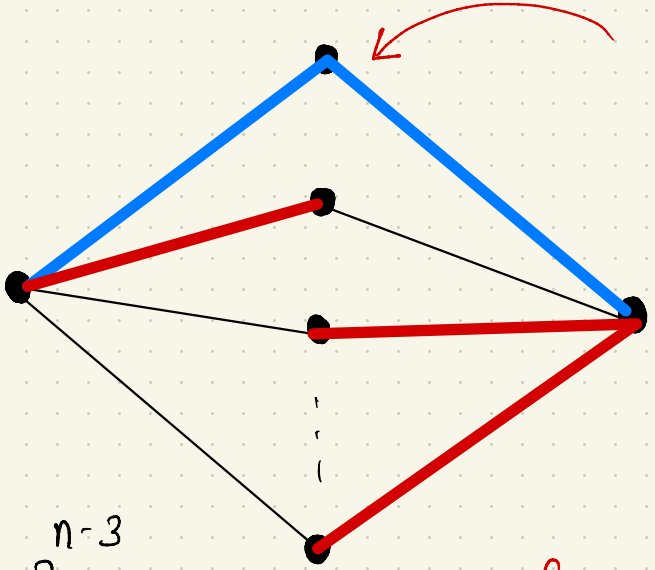
EXPONENTIALLY MANY SPANNING TREES



Some middle node must have both edges in spanning tree

for every other node, pick either left or right edge

EXPONENTIALLY MANY SPANNING TREES

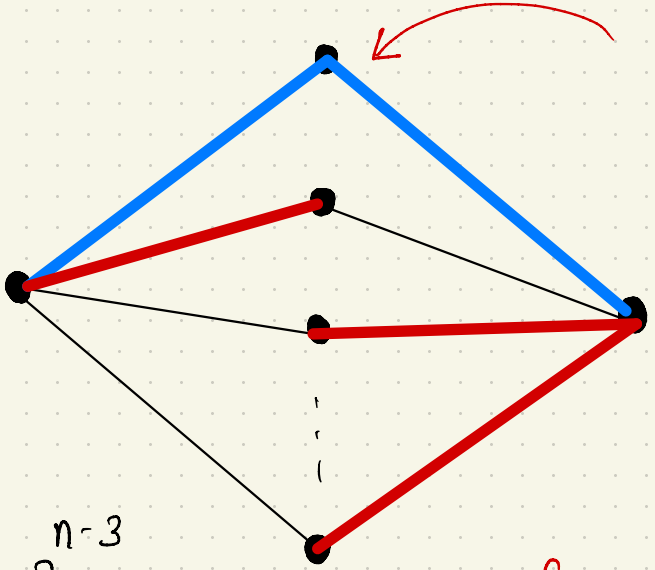


Some middle node must have both edges in spanning tree

$\#$ spanning trees $\geq 2^{n-3}$

for every other node, pick either left or right edge

EXPONENTIALLY MANY SPANNING TREES



Some middle node must have both edges in spanning tree

spanning trees $\geq 2^{n-3}$

for every other node, pick either left or right edge

Cannot hope to find an MST via brute force enumeration