Huffman codes : an example of greedy algorithms.

Encoding data on arbitrary alphabets in binary.

Setup:

Σ - alphabet

- Think of the english alphabet or a subset of it.
- OR $\{0,1\}$ - - - $\{0,1,2,\ldots,9\}$

Binary code for Σ is a way of writing each symbol of Σ as a distinct string over $\{0,1\}$

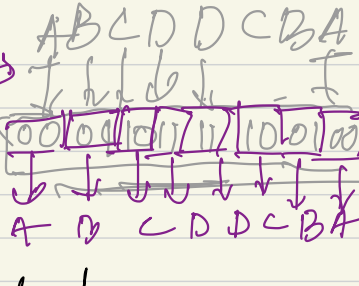
- A binary code for Σ gives us a natural way of encoding strings over Σ as binary strings.
  Necessary for storing inherently non-binary data
- Desirable from such an encoding
  → no collisions → should be able to 'decode'
  → efficiency (encoding, decoding complexity, length of encoding)

E.g:

1) $\Sigma := \{A, B, C, D\}$

Associate a 2-bit string to each symbol.

A : 00
B : 01
C : 10
D : 11



2) Σ — 26 symbols. $= \{A, B, \ldots, Z\}$ $|\Sigma| = 26$

$- 2^5 \geq 26$

Q. How do we encode and decode → natural requirement.

Note: - the length of encoding of each symbol is the same.

- very natural solution

But: can also think of variable length binary code.

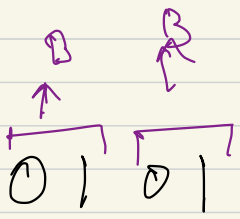
Why?

Might help with 'efficiency'.

	C_1	C_2
A	00	0
B	01	01
C	10	10
D	11	1

① Efficiency of C_2 over C_1

AAADAIDIDBC ✓



01 → B/AD.

② what about decoding.

C_1 :

C_2 :

Want such codes that are uniquely decodable,

- can always do fixed length encoding.
- but can we do better.

Prefix free codes.

$\forall\ \alpha, \beta \in \Sigma$, $Enc(\alpha)$ is not a prefix of $Enc(\beta)$ and vice versa.

- C_2 — not prefix free
- All fixed length codes are prefix free.

- How does prefix freeness help?

- In many cases, not all alphabet symbols in Σ are equally likely.
- Makes sense to encode more frequent symbols by shorter codewords
  → variable length encoding appears more efficient.

Ex:

Advantages of variable length prefix free coding.

Alphabet	Frequency	C_1	C_2
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

Note prefix free nes

$S \in \Sigma^n$, satisfies freq. requirements

$|Enc_{C_1}(S)| \leq 2n.$

$|Enc_{C_2}(S)| =$

$= 1 \times 0.6n + 2 \times 0.25n + 3 \times 0.1n + 3 \times 0.05n$

$= 1.55n < 2n.$ ✓

Suppose in a document, symbols $\{A, B, C, D\}$ appear with the aforementioned freq.

What is the length of the encoding under C_1, C_2?

wtd average of lengths of encodings of symbols in Σ:

Question of interest.

Input: Non negative frequencies p_a for each symbol a of an alphabet Σ of size $n \geq 2$.

Output: A prefix free binary code with min avg encoding length under these frequencies.

$$\text{Avg Enc Length} := \sum_{a \in \Sigma} p_a \cdot (\#\text{bits in Enc}(a))$$

How do we know their frequencies?

Domain knowledge - linguists
- Biologists for Biological data

Here, we assume that the freq are available to us.

- How do we start?
- What is a trivial solution?
- We need to somehow be able to argue optimality.

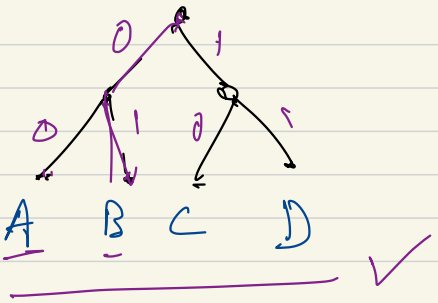
→ prefix free condition makes it hard to think about.

- Try to reformulate this problem as a question about labeled trees
- Will make it slightly easier to think about

Codes vs Trees.

Ex!

A	00
B	01
C	10
D	11



A binary tree rep. the encoding:

Every internal vertex has a left edge labeled 0
— Right edge —— 1

If the label on path from root to a vertex v equals Enc(x) for some $x \in \Sigma$, then v is labeled by x.

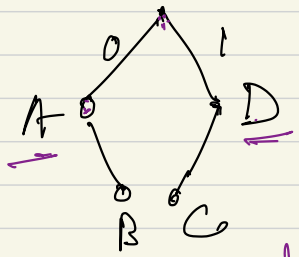
Ex 2

A 0

B 0 1

C 1 0

D 1



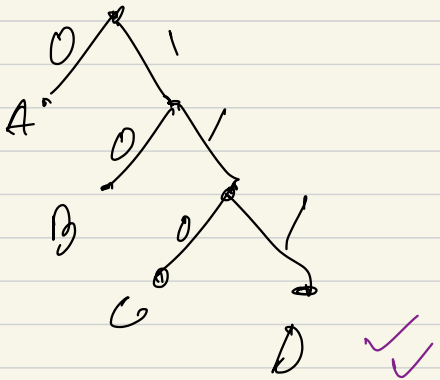
(3)

A 0

B 1 0

C 1 1 0

D 1 1 1



[Labeled bin trees of this type.
vs
codes.]

A Labeled tree from a given code.

① l := max length of $Enc(a)$ for any $a \in \Sigma$

② Take a full binary tree of depth l.

(3) For every internal vertex, label the edge to its left child by 0
edge to right child by 1.

(4) Encoding of each $a \in \Sigma$ defines a path from root to a vertex v. Label v by a.

(5) Delete unlabeled leaves till none remains.
(i.e all leaves labled)

(6) can also delete edge labels. (follow the protocol left→0 right→1.

A code from a given labelled tree.
↳ Σ.

- For every $a \in \Sigma$, follow the path from the root to the vertex labelled a.
- The labels of the edges on this path (concatenated) equals $Enc(a)$.

Note: - can also just drop the edge labels.
- treat left as 0, right as 1

Properties of Codes as properties of trees.

C T

① Length of $Enc(a)$ = depth of the vertex labelled a
(# edges from the root to a on the path)

② Given frequencies p_a for $a \in \Sigma$. ($\sum_{a\in\Sigma} p_a \cdot |Enc(a)|$)

avg length of encoding under the code C

$$= \sum_{a\in\Sigma} p_a \cdot (\text{depth of vertex labelled } a).$$

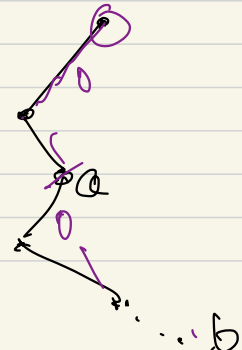
③ Prefix freeness:

Enc(a) is a prefix of Enc(b)

iff

vertex with label a in T is

an ancestor of the vertex with label b



Lemma. Code C is prefix free

iff

for every $a \in \Sigma$, the vertex of T labeled a is a leaf.

(T is pruned - no unlabeled leaves)

Pf. $\Rightarrow$ Easy

$\Leftarrow$

Exercise

Restating our question in terms of trees.

Defn:

For alphabet Σ,

Σ-tree: binary tree with leaves labeled in 1-1 correspondence with symbols in Σ.

(can treat edge labels implicitly)

Prefix free codes for Σ $\Leftrightarrow$ Σ-trees.

Average depth of a Σ-tree

T : Σ-tree

$\{p_a\}_{a \in \Sigma}$: frequencies

$L(T, p)$ ≡ average leaf depth in T

$$= \sum_{a \in \Sigma} p_a \cdot (\text{depth of the leaf labeled } a \text{ in } T)$$

← Avg encoding length of the corresponding code.

Task:

Input: $\{p_a\}_{a \in \Sigma}$ for some alphabet Σ.

Output: Σ-tree with min $L(T, p)$.

Huffman: - efficient algo. for this problem (1951)
- term paper for a course
- corresponding codes are called Huffman codes

Huffman's algo.

① Start with n-isolated vertices each labelled by one alphabet symbol in Σ. ⇒ view this as a forest

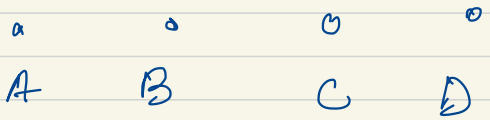
② Iteratively, merge the trees in this forest two at a time based on Huffman's criterion.

③ Stop when one tree remains.

a) What is the 'merge' operation
b) What is Huffman's criterion

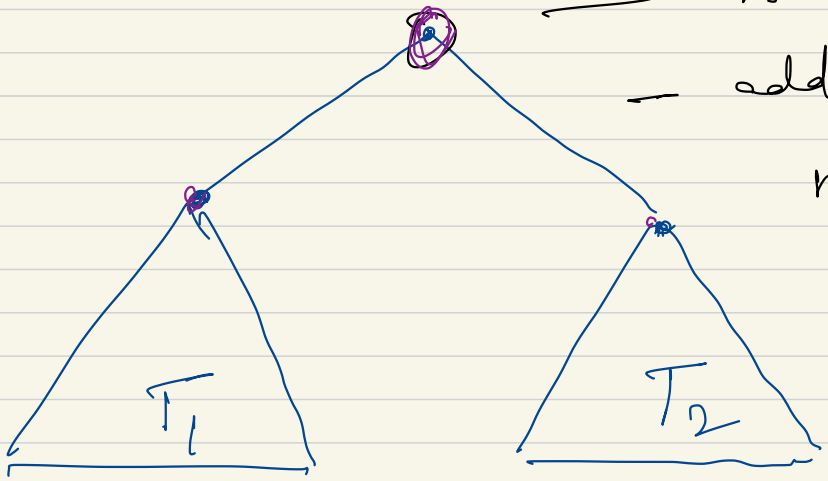
Trees from Merge operations

① start with n-vertices, isolated, labelled by symbols in Σ



② Merge operations

- Two merge any two of trees T_1, T_2
(all trees in this discussion are rooted)



- Add a new root
- add edges to make T_1 and T_2 its two children

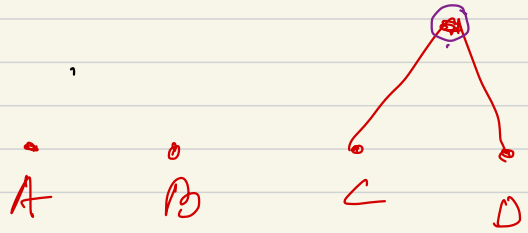
(3) Stop when a single tree remains

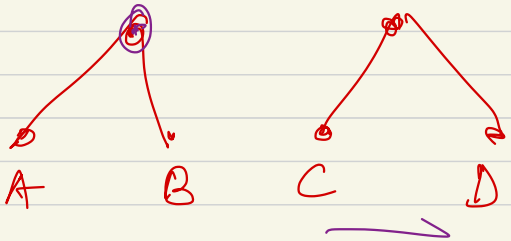
Example:

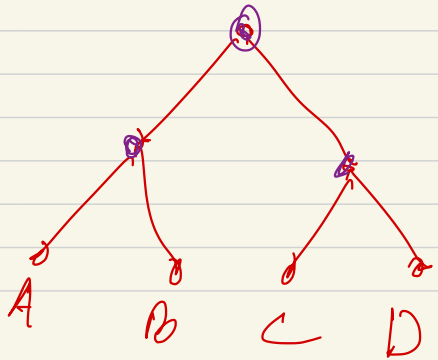
① Merge C, D

② Merge A, B

③ Merge AB, CD





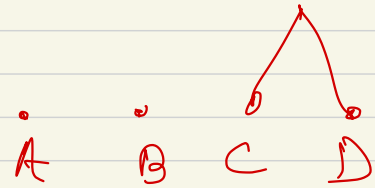


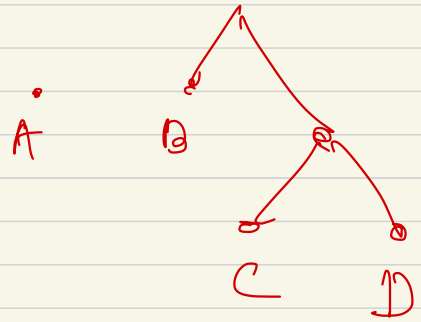
2

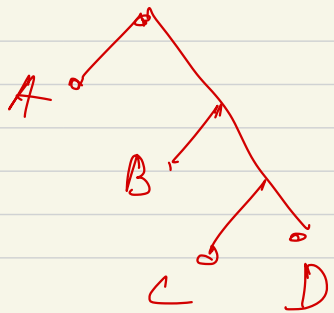
① Merge C, D

② Merge B with CD

③ Merge A with BCD







Huffman's algo.

① Start with n-isolated vertices each labelled by one alphabet symbol in Σ.

② Sequence of Merges based on some criterion.

③ Stop when one tree remains.

Crucial missing detail: what is the merging criterion?

Merging increases the depth of all leaves in the two participating trees.

Huffman's criterion:

Merge the two trees that cause the minimum increase in the average leaf depth.

↳ greedy choice at each stage

Increase in average tree depths of the forest:

$T_1, T_2, T_3, T_4, \ldots, T_m$ $\quad \Sigma_1, \Sigma_2, \ldots, \Sigma_m \subseteq \Sigma$

Σ_i = set of leaf labels of T_i

Sum of average tree depths

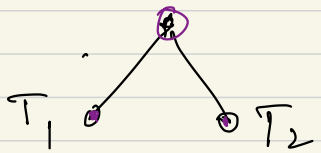
$$= \sum_{i=1}^{m} \left(\sum_{a \in \Sigma_i} p_a \cdot (\text{depth of leaf labeled } a \text{ in } T_i) \right)$$

$(*)$

- Suppose T_1, T_2 are merged.

So, we have T_1', T_3, T_4 - - -

$T_1' \equiv$



Sum of avg tree depths (1)

$$= \sum_{i=3}^{n} \left(\sum_{a \in \Sigma_i} p_a \cdot \text{depth of leaf labelled } a \text{ in } T_i \right)$$

$$+ \sum_{a \in \Sigma_1 \cup \Sigma_2} p_a \cdot \left(\text{depth of leaf labelled } a \text{ in } T_1' \right)$$

$$= \textcircled{1} + \sum_{a \in \Sigma_1} p_a \cdot (1 + \text{depth} - a \text{ in } T_1)$$

$$+ \sum_{a \in \Sigma_2} p_a \cdot (1 + \text{depth} - a \text{ in } T_2)$$

$$\longrightarrow (**)$$

Increase due to merge of T_1, T_2

$$= \sum_{a \in \Sigma_1} p_a + \sum_{a \in \Sigma_2} p_a$$

Intuition:

for every Tree T, assign a weight $p(T) = \sum_{a \in \Sigma(T)} p_a$.

Huffman criterion: Merge trees with the two smallest weights in each step.

Why does this work?

Needs a proof. Not totally trivial. –

Huffman's Algorithm

① for each $a \in \Sigma$

$T_a :=$ tree with one node with label a

$p(T_a) := p_a$

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

② while at ~~least~~ two trees in F

$T_1, T_2 :=$ trees with smallest $p(T_a)$ values in F

$\tilde{T} :=$ Merge T_1, T_2

$p(\tilde{T}) = p(T_1) + p(T_2)$

$F \leftarrow F \setminus \{T_1, T_2\} \cup \{\tilde{T}\}$

(3) Return the unique tree

Example:

0.6	0.25	0.10	0.05
A	B	C	D

Running time?

Correctness

Theorem: For every alphabet Σ and non negative frequencies $\{p_a\}_{a \in \Sigma}$, Huffman's algo outputs a prefix free code with minimum possible average encoding length.

Equivalently Huffman's algorithm outputs a Σ-tree with min possible average tree depth.

Proof:

Induction on $|\Sigma|$

Two main ideas: First some notation.

(1) Let $a, b \in \Sigma$ be symbols s.t p_a, p_b are the two smallest frequencies.

(2) By Huffman Algo: T_a, T_b are merged in the first iteration.

(3) Reg of the other freq. (as long as they are not smaller than p_a, p_b) - output of algo

will have a, b as siblings.

Two steps:

① Prove that there exist optimal Σ-trees where a, b are siblings.

② Prove that the output of Huffman's Algo minimizes avg tree depth for all Σ-trees where a, b are siblings.

Is this sufficient for correctness?

Step 1: T_{ab} := set of all Σ-trees where a, b are siblings.

Want to show:

Lemma 1: $\exists\, T \in T_{ab}$ s.t. avg. leaf depth $L(T, p)$ is min over all Σ-trees.

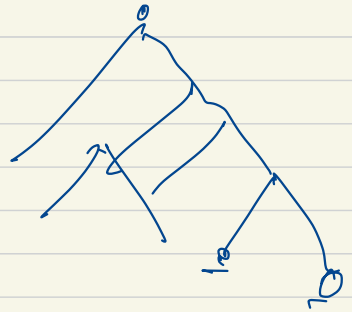
Pf: $\tilde{T}$ — an arbitrary Σ-tree that minimizes avg leaf depth.

Want to argue: there is a $T \in T_{ab}$ s.t.

$$L(T, p) \leq L(\tilde{T}, p).$$

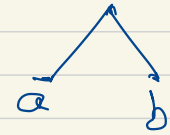
W.l.o.g each vertex in T is either a leaf or has two children. Why?

$\Rightarrow$ $\exists$ 2 leaves with a common parent at the deepest level.



Let x, y be their labels.

Step 2.

- a, b Merge happens first in Huff. algo
- after that algo treats  as a single alphabet symbol ab with freq $p_{ab} = p_a + p_b$.

Formally:

Claim: Output of Huff. Algo on $\Sigma, \{p_\alpha\}_{\alpha \in \Sigma}$ is the same as its output on

$$\tilde{\Sigma} = \Sigma \setminus \{a, b\} \cup \{ab\}$$

$$\{\tilde{p}_\alpha\}_{\alpha \in \tilde{\Sigma}}$$

where, $\tilde{p}_a = p_a$ for $a \neq a, b$

$$\tilde{p}_{ab} = p_a + p_b.$$

Proof of claim:

In fact:

$\tilde{\Sigma}$-trees $\longleftrightarrow$ $T_{ab} = \{\Sigma\text{-trees where } a, b \text{ are siblings}\}$

bijection β

Why?

Lemma: Output of H-algo on Σ, ϕ is $\beta(\tilde{T})$ where $\tilde{T}$ is the output of the algo with input $\tilde{\Sigma}, \tilde{\phi}$.

Further properties:

Lemma: Let $T \to \Sigma$-tree in T_{ab}

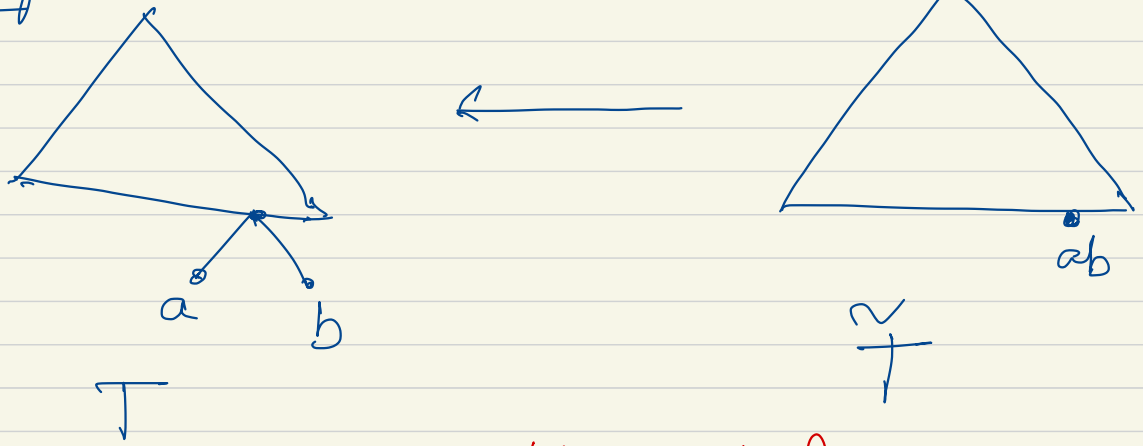
$$\tilde{T} = \beta^{-1}(T).$$

Then,

$$L(T, \rho) = L(\tilde{T}, \tilde{\rho}) + (\rho_a + \rho_b).$$

- Independent of the tree T.
- just depends on a, b.

Proof:



Description of β.

Corollary:

$\tilde{\Sigma}$-tree T^* minimizes $L(\tilde{T}, \tilde{p})$ over all $\tilde{\Sigma}$-trees iff

$T = \beta(T^*)$ minimizes $L(T, p)$ over all Σ-trees in Tab.

Are we done?