HUFFMAN CODING AND HUFFMAN TREE

Coding:

- Reducing strings over arbitrary alphabet Σ_o to strings over a fixed alphabet Σ_c to standardize machine operations ($|\Sigma_c| < |\Sigma_o|$).
  - Binary representation of both operands and operators in machine instructions in computers.
- It must be possible to uniquely decode a code-string (string over Σ_c) to a source-string (string over Σ_o).
  - Not all code-string need to correspond to a source-string.
- Both the coding and decoding should be efficient.

Word: A finite non-empty string over an alphabet (Σ_o or Σ_c).

Simple Coding Mechanism:

- code(a_i) = a non-empty string over Σ_c, for $a_i \in \Sigma_o$.
  code($a_1 a_2 \cdots a_n$) = code(a_1).code(a_2)$\cdots$code(a_n).

Example. $\Sigma_0 = \{A, B, C, D, E\}$ and $\Sigma_c = \{0, 1\}$.

A	*B*	*C*	*D*	*E*		Prefix-property
000	001	010	011	100	code(*AAB*) = 000. 000. 001; easy to decode	yes
0	01	001	0001	00001	code(*C*) = code(*AB*) = 001; not always possible to uniquely decode	no
1	01	001	0001	00001	prefix-free code	yes
1	10	100	1000	10000	not prefix-free code	no

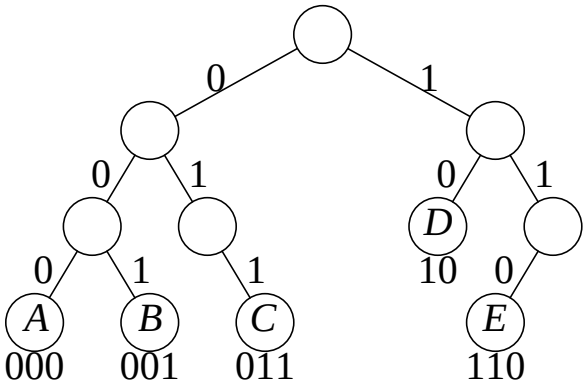
PREFIX-FREE CODE

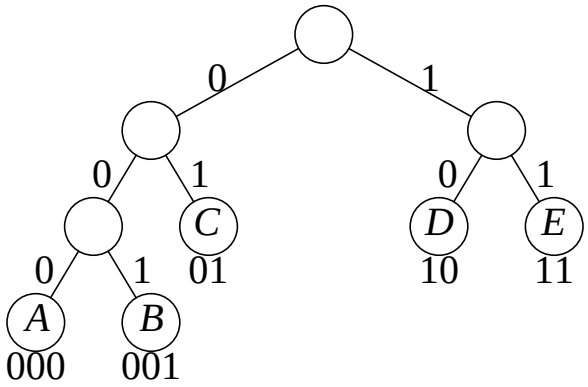
Definition:

- No code(a_i) is a prefix of another code(a_j).
  In particular, code(a_i) ≠ code(a_j) for $a_i \neq a_j$.

Binary-tree representation of prefix-free binary code:

- 0 = label(left branch) and 1 = label(right branch).





C and *E* have shorter code-word;
each non-terminal node has 2 children.

Advantage:

- One can decode the symbols from left to right, i.e., as they are received.
- A sufficient condition for left-to-right unique decoding is the *prefix* property.

Question:

- ? How can we keep prefix-free property and assign shorter codes to some of the symbols {*A*, *B*, ···, *E*}?
- ? What should we do if the symbols in Σ_o occur with probabilities $p(A) = 0.1 = p(B)$, $p(C) = 0.3$, $p(D) = p(E) = 0.25$?

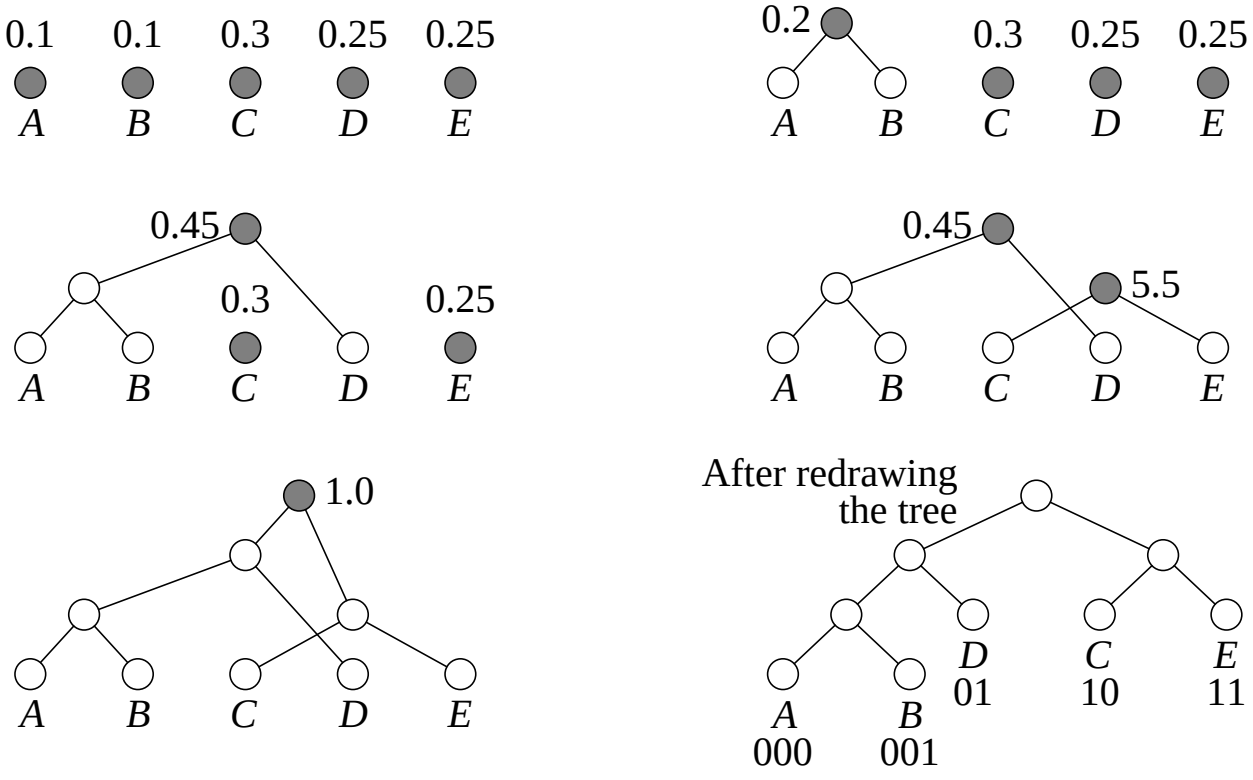
HUFFMAN-TREE

- Binary tree with each non-terminal node having 2 children.
- Gives optimal (min average code-length) prefi x-free binary code to each $a_i \in \Sigma_o$ for a given probabilities $p(a_i) > 0$.

Huffman's Algorithm:

1. Create a terminal node for each $a_i \in \Sigma_o$, with probability $p(a_i)$ and let S = the set of terminal nodes.
2. Select nodes x and y in S with the two smallest probabilities.
3. Replace x and y in S by a node with probability $p(x) + p(y)$. Also, create a node in the tree which is the parent of x and y.
4. Repeat (2)-(3) untill $|S| = 1$.

Example. $\Sigma_0 = \{A, B, \cdots, E\}$ and $p(A) = 0.1 = p(B)$, $p(C) = 0.3$, $p(D) = p(E) = 0.25$. The nodes in S are shown shaded.

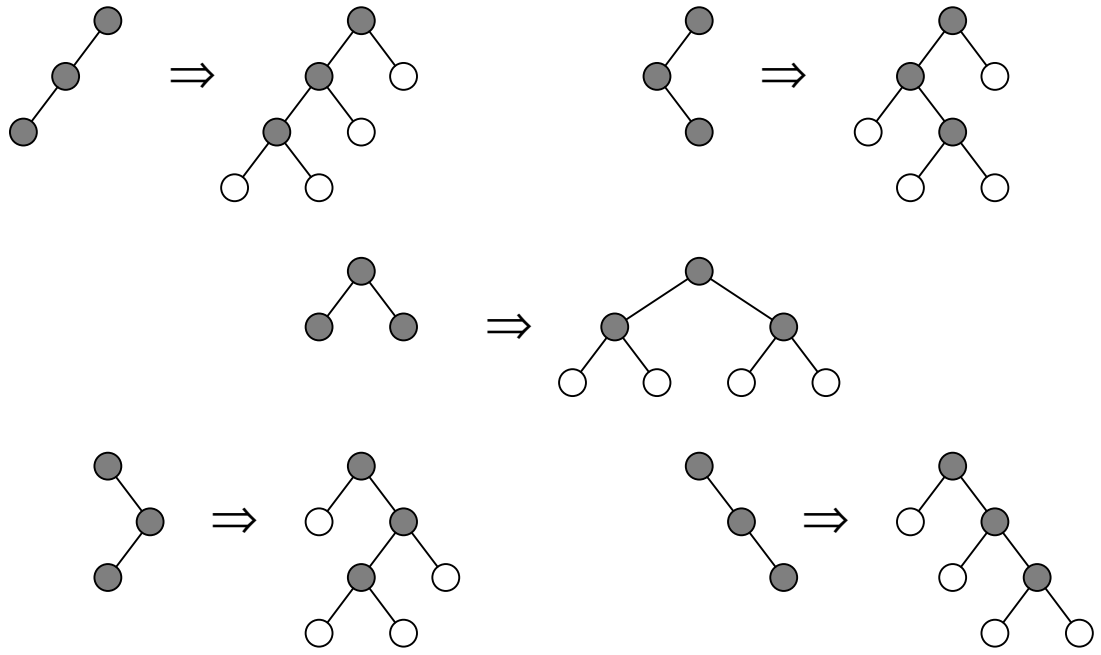


NUMBER OF BINARY TREES

0-2 Binary Tree: Each non-terminal node has 2 children.

- #(Binary trees with N nodes) = $\frac{1}{2N+1}\binom{2N+1}{N}$.
- #(0-2 Binary trees with N terminal nodes)
  = #(Binary trees with $N-1$ nodes) = $\frac{1}{2N-1}\binom{2N-1}{N-1} \geq 2^{N-2}$.

Example. Binary trees with $N-1 = 3$ nodes correspond to 0-2 binary trees with $N = 4$ terminal nodes.



Merits of Huffman's Algorithm:

- It finds the optimal coding in $O(N.\,log\ N)$ time.
- It does so without having to search through $\frac{1}{2N-1}\binom{2N-1}{N-1}$ possible 0-2 binary trees.

DATA-STRUCTURE FOR IMPLEMENTING HUFFMAN'S ALGORITHM

Main Operations:

- Choosing the two nodes with minimum associated probabilities (and creating a parent node, etc).
  - Use heap data-structure for this part.
  - This is done $N-1$ times; total work for this part is $O(N.log\ N)$.
- Addition of each parent node and connecting with the children takes a constant time per node.
  - A total of $N-1$ parent nodes are added, and total time for this $O(N)$.

Complexity: $O(N.log\ N)$.

EXERCISE

1. Consider the codes shows below.

A	*B*	*C*	*D*	*E*
000	001	011	10	110

   (a) Arrange the codes in a binary tree form, with 0 = label(left-branch) and 1 = label(rightbranch).

   (b Does these codes have the prefi x-property? How do you decode the string 10110001000?

   (c) Modify the above code (keeping the prefi x property) so that the new code will have less average length no matter what the probabilities of the symbols are. Show the binary tree for the new code.

   (d) What are the two key properties of the new binary tree (hint: compare with your answer for part (a))?

   (e) Give a suitable probability for the symbols such that prob(A) < prob(B) < prob(C) < prob(D) < prob(E) and the new code in part (c) is optimal (minimum aver. length) for those probabilities.

2. Show the successive heaps in creating the Huffman-Tree for the probabilities $p(A) = 0.1 = p(B)$, $p(C) = 0.3$, $p(D) = 0.14$, $p(E) = 0.12$, and $p(F) = 0.24$.

3. Give some probabilities for the items in $\Sigma_o = \{A, B, \cdots, F\}$ that give the largest possible value for optimal average code length.

4. Argue that for an optimal Huffman-tree, any subtree is optimal (w.r.t to the relative probabilities of its terminal nodes), and also the tree obtained by removing all children and other descendants of a node x gives a tree which is optimal w.r.t to $p(x)$ and the probabilities of its other terminal nodes.