Huffman Code

- Data Compression
  - Lossy and Lossless Compression
  - Adaptive and non-Adaptive Compression
  - Fixed and Variable length Codes
- Prefix Free Codes
  - Binary Tree Representation
  - Goodness Measure
- Generic Greedy Algorithm
- Huffman Code
- Optimality and Implementation

Imdad ullah Khan

Data Compression

Fixed Length Binary Code

- Fixed num of bits for each symbol ▷ e.g. ASCII and Unicode

Variable Length Binary Code

- Variable num of bits for each symbol ▷ uses fewer bits for frequent symbols

Prefix free code

- no code is a prefix of another

If a code is prefix free, then it is uniquely decodable

Prefix free code as Binary Tree

- Prefix free code can be represented by a rooted binary tree
- Leaves are labeled with characters and edges with bits
- The bits along the path from root a leaf is code of the symbol

Compression from Codes (Binary Trees)

- Alphabet $\Sigma = \{a_1, \ldots, a_n\}$
- A document $D \in \mathcal{D}$
- f: frequency distribution, $f(a_i)$: freq. of a_i in D
- C: a compression scheme with code given as a tree
- $B_C(D) = B(D)$: number of bits to encode D with C
- $C(a_i)$: the code for a_i and $len(C(a_i))$: is its length
- $L(a_i) = len(C(a_i))$= length of root to leaf (a_i) path, $depth(a_i)$

Total number of bits needed to encode the document D is

$$B(D) = \sum_{a_i \in \Sigma} f(a_i) L(a_i) = \sum_{i=1}^{n} f(a_i) \cdot \text{ [depth of } a_i \text{ in } T]$$

Problem Formulation

Input: Given an alphabet Σ and a frequency distribution $f : \Sigma \to \mathbb{Z}$

Output: A prefix free code C with minimum $\sum_{i=1}^{n} f(a_i) \cdot$ [depth of a_i in T], where T is the tree representation of C

Equivalently

Input: A document D

Output: A prefix free code C with minimum $B(D)$

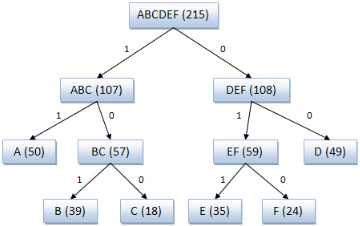
Equivalence follows from the fact that Σ and f can be computed with a single scan of D

Fano-Shannon Code

A greedy divide and conquer approach

- Split Σ into Σ_1 and Σ_2, with *(roughly)* equal frequency distributions
- Recursively compute T_1 for (Σ_1, f_1) and T_2 for (Σ_2, f_2)
- Pre-pend each code in T_1 and T_2 with 0 and 1 resp.

   $\triangleright$ Correspond to merging T_1 and T_2 by making them left and right subtrees of a new root node



source:https://stackoverflow.com

Example where it produces a suboptimal code?

Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

Make every symbol a_i a tree T_{a_i}

for $i = 1$ to $n - 1$ **do**

  Select two trees T_x and T_y

  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node

return the only remaining tree T

Greedy Algorithm

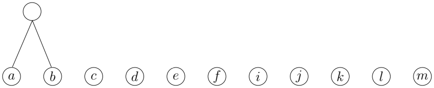
Algorithm Generic Algorithm ($\mathcal{D}$)

Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two trees T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T

Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

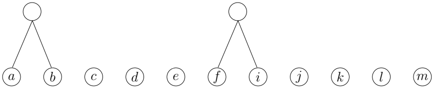
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

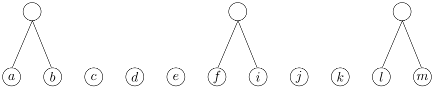
Algorithm Generic Algorithm ($\mathcal{D}$)

Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Algorithm Generic Algorithm ($\mathcal{D}$)

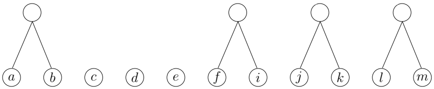
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n-1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

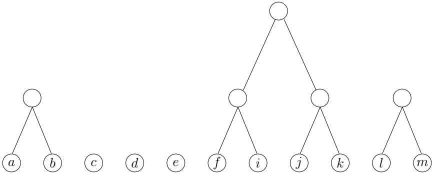
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

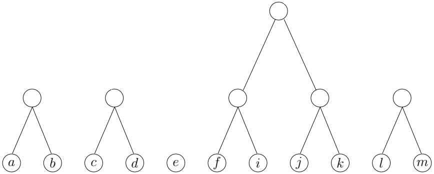
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

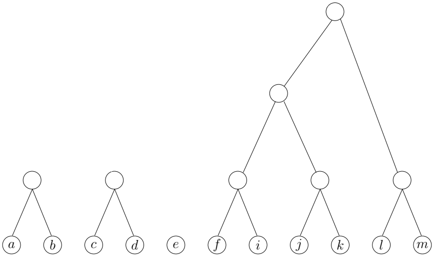
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

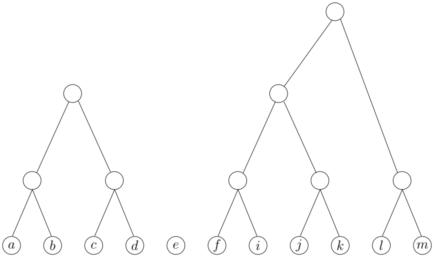
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

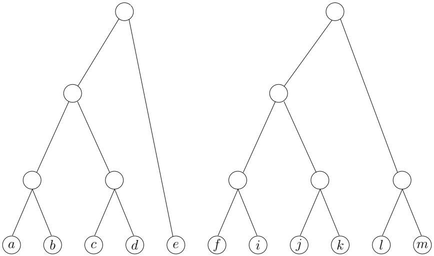
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

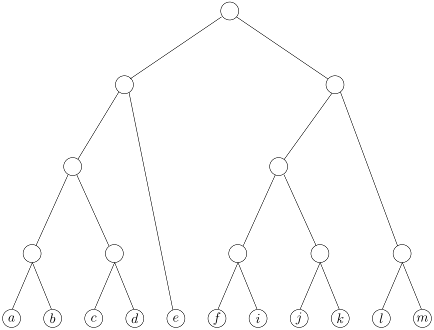
Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T



Greedy Algorithm

Algorithm Generic Algorithm ($\mathcal{D}$)

Make every symbol a_i a tree T_{a_i}
for $i = 1$ to $n - 1$ **do**
  Select two tree T_x and T_y
  $\textsc{Merge}(T_x, T_y)$ ▷ Make them left/right child of a new node
return the only remaining tree T

Clearly constructs a prefix free code

▷ Symbols always and only remain at leaves

Which two subtrees to merge?

- Have to take into account the frequency distribution
- Merging two trees increases code lengths of leaves therein by one
- Code length of symbol is No. of merges its tree undergo
- **Would like frequent symbols go through few merges**