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

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

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?

Huffman Coding

- Have to take into account the frequency distribution
- Merging two trees increases code lengths of leaves therein by one
- Code length of a symbol is the number of merges its tree undergoes
- **Would like frequent symbols go through few merges**

1. Huffman Coding (greedily) chooses two symbols **x** and **y** with lowest frequencies (min and second min)
2. Inserts a new meta-symbol **z** for the merged tree
3. Delete x and y and their frequencies
4. $f(z) \leftarrow f(x) + f(y)$
5. Repeat on the reduced set of symbols

Algorithm Huffman-Tree ($\mathcal{S}$)

```
for x ∈ S do
    MAKE-NODE(x)                        ▷ x is both symbol and pointer
for i = 1 to n − 1 do
    x ← FINDMIN(S)                      ▷ find the symbol with minimum freq.
    S ← S \ {x}
    y ← FINDMIN(S)
    S ← S \ {y}
    MAKENODE(z)
    z · freq ← x · freq + y · freq
    S ← S ∪ {z}
return the only node in S
```

Proof of Optimality: Greedy Choice

The greedy choice property: An optimal code can be constructed by making a locally optimal (greedy) choice for a subproblem

Lemma

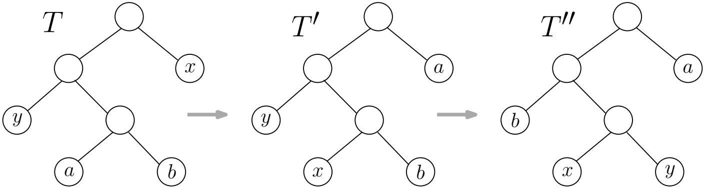
Let x and y be the least and second least frequent symbols in $\mathcal{S}$. Then there exists an optimal prefix free code scheme where the codes for x and y have the same length and differ only in the last bit

In some optimal tree, such x and y are siblings

Proof of Optimality: Greedy Choice

The two least frequent symbols x and y are siblings in an optimal tree

- **Proof:** Let T be an optimal tree
- Let a and b be two deepest sibling leaves in T
- Let $f(a) \leq f(b) \implies f(x) \leq f(a)$ and $f(y) \leq f(b)$



$$B(T') = B(T) - f(x)L(x) - f(a)L(a) + f(x)L(a) + f(a)L(x)$$
$$= -[f(x) - f(a)]\, L(x) + [f(x) - f(a)]\, L(a) \leq B(T)$$

Similarly $B(T'') \leq B(T')$

Proof of Optimality: Optimal Substructure

Let x and y be the two least frequent symbols in S. Let $z \notin S$ be a new symbol with $f'(z) = f(x) + f(y)$ and $S' = S \setminus \{x, y\} \cup \{z\}$

Suppose T' is an optimal tree for $[S', f'(\cdot)]$. Make T by replacing the leaf z in T' by an internal node with x and y as two children

Then T is optimal tree for $[S, f(\cdot)]$

$$\begin{aligned} B(T) &= B(T') - f(z)L'(z) + [f(x) + f(y)][L'(z) + 1] \\ &= B(T') - f(z)L'(z) + [f(x) + f(y)]L'(z) + [f(x) + f(y)] \\ &= B(T') + [f(x) + f(y)] \end{aligned}$$

- Assume for a contradiction that T is not optimal
- Then there is a T'' with $B(T'') < B(T)$ but x and y are siblings
- T'' with the parent of x and y as a leaf is better tree than T' for $[S', f'(\cdot)]$
- A contradiction to optimality of T'

Huffman Coding - Runtime

Algorithm 3 Huffman-Tree $(\mathcal{S}, f(\cdot))$

for $x \in \mathcal{S}$ **do**
  $\textsc{MakeNode}(x)$ $\triangleright$ x is both symbol & pointer
for $i = 1$ to $n - 1$ **do**
  $x \leftarrow \textsc{FindMin}(\mathcal{S})$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{x\}$
  $y \leftarrow \textsc{FindMin}(\mathcal{S})$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{y\}$
  $\textsc{MakeNode}(z)$
  $z \cdot freq \leftarrow x \cdot freq + y \cdot freq$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{z\}$
return the only node in $\mathcal{S}$

- $O(n)$ prep
- $n - 1$ iterations
- 2 $\textsc{FindMin}$ in each
- $+O(1)$ per iteration
- Total $O(n^2)$

Huffman Coding - Implementation

- Repeatedly, finding minimum is the bottleneck
- We use a minimum heap to overcome it

Algorithm Huffman-Tree $(\mathcal{S}, f(\cdot))$

$\mathcal{H} \leftarrow \text{INITIALIZE-HEAP}(\mathcal{S}, f(\cdot))$ ▷ min heap keyed by frequencies
for $i = 1$ to $n - 1$ **do**
  $z \leftarrow \text{NewNode}()$
  $x \leftarrow \text{EXTRACT-MIN}(\mathcal{H})$
  $y \leftarrow \text{EXTRACT-MIN}(\mathcal{H})$
  $z \cdot \textit{left} \leftarrow x$
  $z \cdot \textit{right} \leftarrow y$
  $z \cdot \textit{freq} \leftarrow x \cdot \textit{freq} + y \cdot \textit{freq}$
  $\text{INSERT}(\mathcal{H}, z)$
return $\text{EXTRACT-MIN}(\mathcal{H})$ ▷ Return the root of the tree

---

Huffman Coding - Runtime

Running Time:

- Initially n INSERT ops $\triangleright\ O(n\log n)$
- $2n-2$ EXTRACT-MIN ops $\triangleright\ O(n\log n)$
- $n-1$ INSERT ops $\triangleright\ O(n\log n)$

Total runtime $O(n\log n)$