CE100 Algorithms and Programming II

Huffman Coding

Author: Asst. Prof. Dr. Uğur CORUH

## CE100 Algorithms and Programming II

## Week-9 (Huffman Coding)

#### Spring Semester, 2021-2022

Download [DOC](ce100-week-9-huffman.tr.md_doc.pdf), [SLIDE](ce100-week-9-huffman.tr.md_slide.pdf), [PPTX](ce100-week-9-huffman.tr.md_slide.pptx)

## Huffman Coding

## Outline

* Heap Data Structure (Review Week-4)
* Heap Sort (Review Week-4)
* Huffman Coding

## **Huffman Codes**

## Huffman Codes for Compression

* Widely used and very effective for data compression
* Savings of 20% - 90% typical
	+ (depending on the characteristics of the data)
* **In summary:** Huffman’s greedy algorithm uses a **table of frequencies** of character occurrences to build up an optimal way of **representing each character as a binary string**.

## Binary String Representation - **Example**

* Consider a data file with:
	+ 100K characters
	+ Each character is one of $\{a,b,c,d,e,f\}$
* Frequency of each character in the file:
	+ **frequency:** $\overset{45K}{\overbrace{a}},\overset{13K}{\overbrace{b}},\overset{12K}{\overbrace{c}},\overset{16K}{\overbrace{d}},\overset{9K}{\overbrace{e}},\overset{5K}{\overbrace{f}}$
* **Binary character code:** Each character is represented by a unique binary string.
* **Intuition:**
	+ Frequent characters $⇔$ shorter codewords
	+ Infrequent characters $⇔$ longer codewords

## Binary String Representation - **Example**

$$\begin{matrix}characters&a&b&c&d&e&f\\frequency&45K&13K&12K&16K&9K&5K\\fixed-length&000&001&010&011&100&101\\variable-length(1)&0&101&100&111&1101&1100\\variable-length(2)&0&10&110&1110&11110&11111\end{matrix}$$

* How many total bits needed for **fixed-length** codewords? $100K×3=300K bits$
* How many total bits needed for **variable-length(1)** codewords? $45K×1+13K×3+12K×3+16K×3+9K×4+5K×4=224K$
* How many total bits needed for **variable-length(2)** codewords? $45K×1+13K×2+12K×3+16K×4+9K×5+5K×5=241K$

## Prefix Codes

* **Prefix codes:** No codeword is also a prefix of some other codeword
* **Example:**

$$\begin{matrix}characters&a&b&c&d&e&f\\codeword&0&101&100&111&1101&1100\end{matrix}$$

* It can be shown that:
	+ Optimal data compression is achievable with a **prefix code**
* In other words, optimality is not lost due to **prefix-code** restriction.

## Prefix Codes: Encoding

$$\begin{matrix}characters&a&b&c&d&e&f\\codeword&0&101&100&111&1101&1100\end{matrix}$$

* **Encoding:** Concatenate the codewords representing each character of the file
* **Example:** Encode file “abc” using the codewords above
	+ $abc⇒0.101.100⇒0101100$
* **Note:** “.” denotes the concatenation operation. It is just for illustration purposes, and does not exist in the encoded string.

## Prefix Codes: Decoding

* Decoding is quite simple with a prefix code
* The first codeword in an encoded file is unambiguous
	+ *because no codeword is a prefix of any other*
* **Decoding algorithm:**
	+ Identify the initial codeword
	+ Translate it back to the original character
	+ Remove it from the encoded file
	+ Repeat the decoding process on the remainder of the encoded file.

## Prefix Codes: Decoding - Example

$$\begin{matrix}characters&a&b&c&d&e&f\\codeword&0&101&100&111&1101&1100\end{matrix}$$

* Example: Decode encoded file $001011101$
	+ $001011101$
	+ $0.01011101$
	+ $0.0.1011101$
	+ $0.0.101.1101$
	+ $0.0.101.1101$
	+ $aabe$

## Prefix Codes

* Convenient representation for the prefix code:
	+ a binary tree whose leaves are the given characters
* Binary codeword for a character is the path from the root to that character in the binary tree
* “$0$” means “**go to the left child**”
* “$1$” means “**go to the right child**”

## Binary Tree Representation of Prefix Codes

* **Weight of an internal node:** sum of weights of the leaves in its subtree
* The binary tree corresponding to the fixed-length code



center h:450px

## Binary Tree Representation of Prefix Codes

* **Weight of an internal node:** sum of weights of the leaves in its subtree
* The binary tree corresponding to the **optimal variable-length** code
* An optimal code for a file is always represented by a **full binary tree**



bg right:40% h:600px

## Full Binary Tree Representation of Prefix Codes

* Consider an **FBT** corresponding to an optimal prefix code
* It has $\left|C\right|$ leaves (external nodes)
* One for each letter of the alphabet where $C$ is the alphabet from which the characters are drawn
* **Lemma:** An **FBT** with $\left|C\right|$ external nodes has exactly $\left|C\right|−1$ internal nodes

## Full Binary Tree Representation of Prefix Codes

* Consider an $FBT$ $T$, corresponding to a prefix code.
* **Notation**:
	+ $f\left(c\right)$: frequency of character c in the file
	+ $d\_{T}\left(c\right)$: depth of $c$’s leaf in the $FBT$ $T$
	+ $B\left(T\right)$: the number of bits required to encode the file
* What is the length of the codeword for $c$?
	+ $d\_{T}\left(c\right)$, same as the depth of $c$ in $T$
* How to compute $B\left(T\right)$, cost of tree $T$?
	+ $B\left(T\right)=\sum\_{c\in C}^{​}f\left(c\right)d\_{T}\left(c\right)$

## Cost Computation - **Example**

$$B\left(T\right)=\sum\_{c\in C}^{​}f\left(c\right)d\_{T}\left(c\right)$$

$$\begin{matrix}B\left(T\right)=&\left(45×1\right)+\left(12×3\right)+\\&\left(13×3\right)+\left(16×3\right)+\\&\left(5×4\right)+\left(9×4\right)\\=&224\end{matrix}$$



bg right:50% h:500px

## Prefix Codes

* **Lemma:** Let each internal node i is labeled with the sum of the weight $w\left(i\right)$ of the leaves in its subtree
* Then

$$B\left(T\right)=\sum\_{c\in C}^{​}f\left(c\right)d\_{T}\left(c\right)=\sum\_{i\in I\_{T}}^{​}w\left(i\right)$$

* *where* $I\_{T}$ *is the set of internal nodes of* $T$
* **Proof:** Consider a leaf node $c$ with $f\left(c\right)$ & $d\_{T}\left(c\right)$
	+ Then, $f\left(c\right)$ appears in the weights of $d\_{T}\left(c\right)$ internal node
	+ along the path from $c$ to the root
	+ Hence, $f\left(c\right)$ appears $d\_{T}\left(c\right)$ times in the above summation

## Cost Computation - **Example**

$$B\left(T\right)=\sum\_{i\in I\_{T}}^{​}w\left(i\right)$$

$$\begin{matrix}B\left(T\right)=&100+55+\\&25+30+14\\=&224\end{matrix}$$



bg right:50% h:500px

## Constructing a Huffman Code

* **Problem Formulation:** For a given character set C, construct an optimal prefix code with the minimum total cost
* **Huffman** invented a **greedy algorithm** that constructs an optimal prefix code called a **Huffman code**
* The greedy algorithm
	+ builds the **FBT** corresponding to the optimal code in a **bottom-up** manner
	+ begins with a set of $\left|C\right|$ leaves
	+ performs a sequence of $\left|C\right|−1$ “**merges**” to create the final tree

## Constructing a Huffman Code

* A **priority queue** $Q$, keyed on $f$, is used to identify the two **least-frequent** objects to merge
* The result of the **merger** of two objects is a **new object**
	+ inserted into the priority queue according to its frequency
	+ which is the sum of the frequencies of the two objects merged

## Constructing a Huffman Code

* Priority queue is implemented as a binary heap
* Initiation of $Q$ ($BUILD-HEAP$): $O\left(n\right)$ time
* $EXTRACT-MIN$ & $INSERT$ take $O\left(lgn\right)$ time on $Q$ with $n$ objects

## Constructing a Huffman Code

$$\begin{matrix}&HUFFMAN\left(c\right)\\& n\leftarrow \left|C\right|\\& Q\leftarrow BUILD-HEAP\left(c\right)\\& for i\leftarrow 1 to n−1 do\\&  z\leftarrow ALLOCATE-NODE\left(\right)\\&  x\leftarrow left\left[z\right]\leftarrow EXTRACT-MIN\left(Q\right)\\&  y\leftarrow right\left[z\right]\leftarrow EXTRACT-MIN\left(Q\right)\\&  f\left[z\right]\leftarrow f\left[x\right]\leftarrow f\left[y\right]\\&  INSERT\left(Q,z\right)\\& return EXTRACT-MIN\left(Q\right)⊲one object left in Q\end{matrix}$$

## Constructing a Huffman Code - **Example**

* Start with one leaf node for each character
* The $2$ nodes with the least frequencies: $f\&e$
* Merge $f\&e$ and create an internal node
* Set the internal node frequency to $5+9=14$



center h:300px

## Constructing a Huffman Code - **Example**

* The 2 nodes with least frequencies: $b\&c$



center h:400px

## Constructing a Huffman Code - **Example**



center h:350px

## Constructing a Huffman Code - **Example**



bg right:50% h:350px

## Constructing a Huffman Code - **Example**



bg right:50% h:600px

## Constructing a Huffman Code - **Example**



bg right:50% h:600px

## Correctness Proof of Huffman’s Algorithm

* **We need to prove:**
	+ The greedy choice property
	+ The optimal substructure property
* **What is the greedy step in Huffman’s algorithm?**
	+ *Merging the two characters with the lowest frequencies*
* *We will first prove the greedy choice property*

## Greedy Choice Property

* **Lemma 1:** Let $x\&y$ be two characters in $C$ having the **lowest frequencies**.
* Then, $∃$ an optimal prefix code for $C$ in which the codewords for $x\&y$ have the same length and differ only in the last bit
* **Note:** *If* $x\&y$ *are merged in Huffman’s algorithm, their codewords are guaranteed to have the same length and they will differ only in the last bit*.
	+ *Lemma 1* states that there exists an optimal solution where this is the case.

## Greedy Choice Property - Proof

* Outline of the proof:
	+ Start with an arbitrary optimal solution
	+ Convert it to an optimal solution that satisfies the greedy choice property.
* **Proof:** Let $T$ be an arbitrary optimal solution where:
	+ $b\&c$ are the sibling leaves with the **max depth**
	+ $x\&y$ are the characters with the **lowest frequencies**

## Greedy Choice Property - Proof



bg right:50% h:600px

* Reminder:
	+ $b\&c$ are the nodes with max depth
	+ $x\&y$ are the nodes with min freq.
* Without loss of generality, assume:
	+ $f\left(x\right)\leq f\left(y\right)$
	+ $f\left(b\right)\leq f\left(c\right)$
* Then, it must be the case that:
	+ $f\left(x\right)\leq f\left(b\right)$
	+ $f\left(y\right)\leq f\left(c\right)$

## Greedy Choice Property - Proof

* $T⇒T′$: exchange the positions of the leaves $b\&x$
* $T′⇒T″$: exchange the positions of the leaves $c\&y$



center h:400px

## Greedy Choice Property - Proof



center h:500px

## Greedy Choice Property - Proof

* **Reminder:** Cost of tree $T′$

$$B\left(T\right)=\sum\_{c\in C}^{​}f\left(c\right)d\_{T′}\left(c\right)$$

* How does $B\left(T′\right)$ compare to $B\left(T\right)$?
* **Reminder:** $f\left(x\right)\leq f\left(b\right)$
	+ $d\_{T′}\left(x\right)=d\_{T}\left(b\right)$ and $d\_{T′}\left(b\right)=d\_{T}\left(x\right)$



bg right:50% h:500px

## Greedy Choice Property - Proof

* **Reminder:** $f\left(x\right)\leq f\left(b\right)$
	+ $d\_{T′}\left(x\right)=d\_{T}\left(b\right)$ and $d\_{T′}\left(b\right)=d\_{T}\left(x\right)$
* The difference in cost between $T$ and $T′$:

$$\begin{matrix}B\left(T\right)−B\left(T′\right)=&\sum\_{c\in C}^{​}f\left(c\right)d\_{T}\left(c\right)−\sum\_{c\in C}^{​}f\left(c\right)d\_{T′}\left(c\right)\\&=f\left[x\right]d\_{T}\left(x\right)+f\left[b\right]d\_{T}\left(b\right)−f\left[x\right]d\_{T′}\left(x\right)−f\left[b\right]d\_{T′}\left(b\right)\\&=f\left[x\right]d\_{T}\left(x\right)+f\left[b\right]d\_{T}\left(b\right)−f\left[x\right]d\_{T}\left(x\right)−f\left[b\right]d\_{T}\left(b\right)\\&=f\left[b\right]\left(d\_{T}\left(b\right)+d\_{T}\left(x\right)\right)−f\left[x\right]\left(d\_{T}\left(b\right)−d\_{T}\left(x\right)\right)\\&=\left(f\left[b\right]−f\left[x\right]\right)\left(d\_{T}\left(b\right)+d\_{T}\left(x\right)\right)\end{matrix}$$

## Greedy Choice Property - Proof

$$\begin{matrix}B\left(T\right)−B\left(T′\right)=\left(f\left[b\right]−f\left[x\right]\right)\left(d\_{T}\left(b\right)+d\_{T}\left(x\right)\right)\end{matrix}$$

* Since $f\left[b\right]−f\left[x\right]\geq 0$ and $d\_{T}\left(b\right)\geq d\_{T}\left(x\right)$
	+ therefore $B\left(T′\right)\leq B\left(T\right)$
* In other words, $T′$ is also optimal

## Greedy Choice Property - Proof



center h:450px

## Greedy Choice Property - Proof

* We can similarly show that
* $B\left(T′\right)−B\left(T″\right)\geq 0⇒B\left(T″\right)\leq B\left(T′\right)$
	+ which implies $B\left(T″\right)\leq B\left(T\right)$
* Since $T$ is optimal $⇒B\left(T″\right)=B\left(T\right)⇒T″$ is also optimal
* **Note:** $T″$ contains our greedy choice:
	+ Characters $x\&y$ appear as sibling leaves of max-depth in $T″$
* Hence, the proof for the greedy choice property is complete

## Greedy-Choice Property of Determining an Optimal Code

* **Lemma 1** implies that
	+ process of building an optimal tree
	+ by mergers can begin with the greedy choice of merging
	+ those two characters with the lowest frequency
* We have already proved that $B\left(T\right)=\sum\_{i\in I\_{T}}^{​}w\left(i\right)$ , that is,
	+ the total cost of the tree constructed
	+ is the **sum** of the **costs** of its **mergers** (**internal nodes**) **of all possible mergers**
* At each step **Huffman chooses** the merger that incurs the **least cost**

## Optimal Substructure Property

* Consider an optimal solution $T$ for alphabet $C$. Let $x$ and $y$ be any two sibling leaf nodes in $T$. Let $z$ be the parent node of $x$ and $y$ in $T$.
* Consider the subtree $T′$ where $T′=T–\{x,y\}$.
	+ Here, consider z as a new character, where
		- $f\left[z\right]=f\left[x\right]+f\left[y\right]$
* **Optimal substructure property:** $T′$ must be optimal for the alphabet $C′$, where $C′=C–\{x,y\}∪\{z\}$



bg right:30% h:500px

## Optimal Substructure Property - Proof

Reminder:

$$B\left(T\right)=\sum\_{c\in C}^{​}f\left[c\right]d\_{T}\left(c\right)$$

Try to express $B\left(T\right)$ in terms of $B\left(T′\right)$.

**Note:** All characters in $C′$ have the same depth in $T$ and $T′$.

$$B\left(T\right)=B\left(T′\right)–cost\left(z\right)+cost\left(x\right)+cost\left(y\right)$$



bg right:30% h:500px

## Optimal Substructure Property - Proof

Reminder:

$$B\left(T\right)=\sum\_{c\in C}^{​}f\left[c\right]d\_{T}\left(c\right)$$

$$\begin{matrix}B\left(T\right)&=B\left(T′\right)–cost\left(z\right)+cost\left(x\right)+cost\left(y\right)\\&=B\left(T′\right)−f\left[z\right].d\_{T}\left(z\right)+f\left[x\right].d\_{T}\left(x\right)+f\left[y\right].d\_{T}\left(y\right)\\&=B\left(T′\right)−f\left[z\right].d\_{T}\left(z\right)+\left(f\left[x\right]+f\left[y\right]\right)\left(d\_{T}\left(z\right)+1\right)\\&=B\left(T′\right)−f\left[z\right].d\_{T}\left(z\right)+f\left[z\right]\left(d\_{T}\left(z\right)+1\right)\\&=B\left(T′\right)−f\left[z\right]\end{matrix}$$

$$\begin{matrix}d\_{T}\left(x\right)=d\_{T}\left(z\right)+1\\d\_{T}\left(y\right)=d\_{T}\left(z\right)+1\end{matrix}$$

$$\begin{matrix}B\left(T\right)=B\left(T′\right)+f\left[x\right]+f\left[y\right]\end{matrix}$$



bg right:30% h:500px

## Optimal Substructure Property - Proof

* We want to prove that $T′$ is optimal for
	+ $C′=C–\{x,y\}∪\{z\}$
* Assume by contradiction that that there exists another solution for $C′$ with smaller cost than $T′$. Call this solution $R′$:
* $B\left(R′\right)<B\left(T′\right)$
* Let us construct another prefix tree $R$ by adding $x\&y$ as children of $z$ in $R′$

$$\begin{matrix}B\left(T\right)=B\left(T′\right)+f\left[x\right]+f\left[y\right]\end{matrix}$$



bg right:30% h:500px

## Optimal Substructure Property - Proof

* Let us construct another prefix tree $R$ by adding $x\&y$ as children of $z$ in $R′$.
* We have:
	+ $B\left(R\right)=B\left(R′\right)+f\left[x\right]+f\left[y\right]$
* In the beginning, we assumed that:
	+ $B\left(Rʹ\right)<B\left(T′\right)$
* So, we have:
	+ $B\left(R\right)<B\left(T′\right)+f\left[x\right]+f\left[y\right]=B\left(T\right)$

**Contradiction! Proof complete**



bg right:30% h:500px

## Greedy Algorithm for Huffman Coding - Summary

* For the greedy algorithm, we have proven that:
	+ **The greedy choice property** holds.
	+ **The optimal substructure property** holds.
* So, the greedy algorithm is optimal.

## References

* [Introduction to Algorithms, Third Edition | The MIT Press](https://mitpress.mit.edu/books/introduction-algorithms-third-edition)
* [Bilkent CS473 Course Notes (new)](http://nabil.abubaker.bilkent.edu.tr/473/)
* [Bilkent CS473 Course Notes (old)](http://cs.bilkent.edu.tr/~ugur/teaching/cs473/)

$−End−Of−Week−9−Course−Module−$