CSCI 5451 Fall 2015

Week 5 Notes

Professor Ellen Gethner

September 14, 2015

Greedy Algorithms: Second Big Example

- **Big Greedy Example 2: Data Compression by way of Huffman Encoding**

Greedy Algorithms: Second Big Example

- **Big Greedy Example 2: Data Compression by way of Huffman Encoding**

- **Problem.** Given a text file, find a compressed file, as small as possible, such that the original file can be reconstructed fully and efficiently.

Greedy Algorithms: Second Big Example

- **Big Greedy Example 2: Data Compression by way of Huffman Encoding**

- **Problem.** Given a text file, find a compressed file, as small as possible, such that the original file can be reconstructed fully and efficiently.

- **Some Applications.**
  1. Storage savings

  2. Communications (when the cost of sending information is greater than the cost of reconstruction)

Data Compression, thinking about designing an algorithm...

- For purposes of illustration, assume that our alphabet is a sequence of letters from $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$.

Data Compression, thinking about designing an algorithm...

- For purposes of illustration, assume that our alphabet is a sequence of letters from $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$.

- Each character in an encoding will be a unique string of bits.

Data Compression, thinking about designing an algorithm...

- For purposes of illustration, assume that our alphabet is a sequence of letters from $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$.

- Each character in an encoding will be a unique string of bits.

- If the length of each character is the same, say $k \in \mathbb{N}$, then the length of the encoded file is nk where n is the number of characters in the file.

Data Compression, thinking about designing an algorithm...

- For purposes of illustration, assume that our alphabet is a sequence of letters from $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$.

- Each character in an encoding will be a unique string of bits.

- If the length of each character is the same, say $k \in \mathbb{N}$, then the length of the encoded file is nk where n is the number of characters in the file.

- For example, the length of each character in standard ASCII is seven bits.

Data Compression, thinking about designing an algorithm...

- For purposes of illustration, assume that our alphabet is a sequence of letters from $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$.

- Each character in an encoding will be a unique string of bits.

- If the length of each character is the same, say $k \in \mathbb{N}$, then the length of the encoded file is nk where n is the number of characters in the file.

- For example, the length of each character in standard ASCII is seven bits.

- **Better Idea.** Choose smaller bit representations for characters that occur more frequently (like "e" in the English language).

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.
- Suppose in our new encoding scheme we decide to shorten A to 1001.

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.
- Suppose in our new encoding scheme we decide to shorten A to 1001.
- Then we could no longer encode M by 1001101 because the new version of A would be a prefix of M, thus causing an ambiguity.

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.
- Suppose in our new encoding scheme we decide to shorten A to 1001.
- Then we could no longer encode M by 1001101 because the new version of A would be a prefix of M, thus causing an ambiguity.
- We could use delimiters, but that would defeat our purpose of compression.

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.
- Suppose in our new encoding scheme we decide to shorten A to 1001.
- Then we could no longer encode M by 1001101 because the new version of A would be a prefix of M, thus causing an ambiguity.
- We could use delimiters, but that would defeat our purpose of compression.
- Thus, in an encoding, we should as that all of the prefixes of the encoding of one character are not the same as a complete encoding of any other character.

Data Compression: Prefix Constraint

- In ASCII, the letter A is encoded by 1000001.
- Suppose in our new encoding scheme we decide to shorten A to 1001.
- Then we could no longer encode M by 1001101 because the new version of A would be a prefix of M, thus causing an ambiguity.
- We could use delimiters, but that would defeat our purpose of compression.
- Thus, in an encoding, we should as that all of the prefixes of the encoding of one character are not the same as a complete encoding of any other character.
- The above constraint is called the Prefix Constraint.

Problem Restatement

- **Problem**. Given text file,

Problem Restatement

- **Problem**. Given text file,
- find an encoding of the alphabet of characters

Problem Restatement

- **Problem**. Given text file,
- find an encoding of the alphabet of characters
- that satisfies the prefix constraint and

Problem Restatement

- **Problem**. Given text file,
- find an encoding of the alphabet of characters
- that satisfies the prefix constraint and
- that minimizes the total number of bits needed to encode the text.

Data Compression Design, continued

- **Set-up.** Let the text file be called F and

Data Compression Design, continued

- **Set-up.** Let the text file be called F and

- denote the characters in the text file by $C_1, C_2, \ldots, C_n$ and suppose character C_i occurs f_i times in the text file.

Data Compression Design, continued

- **Set-up.** Let the text file be called F and
- denote the characters in the text file by $C_1, C_2, \ldots, C_n$ and suppose character C_i occurs f_i times in the text file.
- The quantity f_i is called the *frequency* of character C_i.

Data Compression Design, continued

- **Set-up.** Let the text file be called F and
- denote the characters in the text file by $C_1, C_2, \ldots, C_n$ and suppose character C_i occurs f_i times in the text file.
- The quantity f_i is called the *frequency* of character C_i.
- **Notation.** The encoding will be called E,

Data Compression Design, continued

- **Set-up.** Let the text file be called F and
- denote the characters in the text file by $C_1, C_2, \dots, C_n$ and suppose character C_i occurs f_i times in the text file.
- The quantity f_i is called the *frequency* of character C_i.
- **Notation.** The encoding will be called E,
- C_i will be represented by big string S_i of length s_i.

Data Compression Design, continued

- **Set-up.** Let the text file be called F and
- denote the characters in the text file by $C_1, C_2, \ldots, C_n$ and suppose character C_i occurs f_i times in the text file.
- The quantity f_i is called the *frequency* of character C_i.
- **Notation.** The encoding will be called E,
- C_i will be represented by big string S_i of length s_i.
- Then the length of F encoded by E is

$$L(E,F) = \sum_{i=1}^{n} s_i f_i.$$

Another Restatement of the Data Compression Problem

- Recall: $L(E, F) = \sum_{i=1}^{n} s_i f_i$.

Another Restatement of the Data Compression Problem

- Recall: $L(E, F) = \sum_{i=1}^{n} s_i f_i$.

- **Goal.** Find an encoding E that satisfies the prefix constraint, and

Another Restatement of the Data Compression Problem

- Recall: $L(E, F) = \sum_{i=1}^{n} s_i f_i$.
- **Goal.** Find an encoding E that satisfies the prefix constraint, and
- that minimizes $L(E, F)$.

More Design Thoughts

- How should the encoding behave?

More Design Thoughts

- How should the encoding behave?
- Scan through the bit sequence left to right until we arrive at a character.

More Design Thoughts

- How should the encoding behave?
- Scan through the bit sequence left to right until we arrive at a character.
- Record the character and continued scanning until another character is found.

More Design Thoughts

- How should the encoding behave?
- Scan through the bit sequence left to right until we arrive at a character.
- Record the character and continued scanning until another character is found.
- Record and repeat.

More Design Thoughts: Binary Tree

- Consider a binary tree in which each node has either two children or no children, and moreover

More Design Thoughts: Binary Tree

- Consider a binary tree in which each node has either two children or no children, and moreover

- siblings are labeled with 0 or 1 and

More Design Thoughts: Binary Tree

- Consider a binary tree in which each node has either two children or no children, and moreover
- siblings are labeled with 0 or 1 and
- no two siblings have the same label.

More Design Thoughts: Binary Tree

- Consider a binary tree in which each node has either two children or no children, and moreover
- siblings are labeled with 0 or 1 and
- no two siblings have the same label.
- If we build such a tree and somehow use the leaves of the tree in the encoding, then the encoding will satisfy the prefix constraint.

More Design Thoughts: Binary Tree

- Consider a binary tree in which each node has either two children or no children, and moreover
- siblings are labeled with 0 or 1 and
- no two siblings have the same label.
- If we build such a tree and somehow use the leaves of the tree in the encoding, then the encoding will satisfy the prefix constraint.
- That is, when the encoded file is scanned and we reach a leaf, we determine the encoding of the character.

More Design Thoughts: Binary Tree

- **Another Problem Restatement.** Construct the binary tree that minimizes $L(E, F)$.

More Design Thoughts: Binary Tree

- **Another Problem Restatement.** Construct the binary tree that minimizes $L(E, F)$.

- **Goal.** Reduce the problem with n characters to one of $n - 1$ characters so that we can use inductive reasoning.

More Design Thoughts: Binary Tree

- **Another Problem Restatement.** Construct the binary tree that minimizes $L(E, F)$.
- **Goal.** Reduce the problem with n characters to one of $n - 1$ characters so that we can use inductive reasoning.
- We **won't** eliminate a character from the alphabet!

More Design Thoughts: Binary Tree

- **Another Problem Restatement.** Construct the binary tree that minimizes $L(E, F)$.
- **Goal.** Reduce the problem with n characters to one of $n - 1$ characters so that we can use inductive reasoning.
- We **won't** eliminate a character from the alphabet!
- Instead, we'll create a new artificial character made up of two low frequency characters.

Create a Binary Tree Inductively

- Let C_i and C_j be the two characters of lowest frequency, where

Create a Binary Tree Inductively

- Let C_i and C_j be the two characters of lowest frequency, where

- we arbitrarily break ties whenever necessary.

Create a Binary Tree Inductively

- Let C_i and C_j be the two characters of lowest frequency, where
- we arbitrarily break ties whenever necessary.
- We will see shortly that there exists a binary tree that minimizes $L(E, F)$ in which C_i and C_j correspond to leaves of maximum distance from the root.

Create a Binary Tree Inductively

- Let C_i and C_j be the two characters of lowest frequency, where
- we arbitrarily break ties whenever necessary.
- We will see shortly that there exists a binary tree that minimizes $L(E, F)$ in which C_i and C_j correspond to leaves of maximum distance from the root.
- In fact, since each node has either 0 or 2 children, we can always assume that C_i and C_j are siblings.

Create a Binary Tree Inductively

- Let C_i and C_j be the two characters of lowest frequency, where
- we arbitrarily break ties whenever necessary.
- We will see shortly that there exists a binary tree that minimizes $L(E, F)$ in which C_i and C_j correspond to leaves of maximum distance from the root.
- In fact, since each node has either 0 or 2 children, we can always assume that C_i and C_j are siblings.
- The new character is called C_{ij} and will have frequency $f_i + f_j$, where recall that f_x is the frequency of C_x.

Create a Binary Tree Inductively: C_{ij} has frequency $f_i + f_j$

- The new problem has and alphabet of $n-1$ characters ($n-2$ old and one new) and can be solved by an induction hypothesis.

Create a Binary Tree Inductively: C_{ij} has frequency $f_i + f_j$

- The new problem has and alphabet of $n - 1$ characters ($n - 2$ old and one new) and can be solved by an induction hypothesis.

- Obtain a solution to the original problem by substituting an internal node in the reduced problem

Create a Binary Tree Inductively: C_{ij} has frequency $f_i + f_j$

- The new problem has and alphabet of $n-1$ characters ($n-2$ old and one new) and can be solved by an induction hypothesis.

- Obtain a solution to the original problem by substituting an internal node in the reduced problem

- with two leaves corresponding to C_i and C_j in place of the leaf corresponding to C_{ij}.

Create a Binary Tree Inductively: C_{ij} has frequency $f_i + f_j$

- The new problem has and alphabet of $n - 1$ characters ($n - 2$ old and one new) and can be solved by an induction hypothesis.

- Obtain a solution to the original problem by substituting an internal node in the reduced problem

- with two leaves corresponding to C_i and C_j in place of the leaf corresponding to C_{ij}.

- The process that we'll make precise is called Huffman Encoding and the tree so constructed is called a Huffman Tree.

Create a Binary Tree Inductively: C_{ij} has frequency $f_i + f_j$

- The new problem has and alphabet of $n-1$ characters ($n-2$ old and one new) and can be solved by an induction hypothesis.
- Obtain a solution to the original problem by substituting an internal node in the reduced problem
- with two leaves corresponding to C_i and C_j in place of the leaf corresponding to C_{ij}.
- The process that we'll make precise is called Huffman Encoding and the tree so constructed is called a Huffman Tree.
- We'll do an example before writing down the algorithm.

Example of a Huffman Encoding

- The textfile is $F = helloyellowcows$

Example of a Huffman Encoding

- The textfile is $F = helloyellowcows$
- The alphabet of characters is $\{h, e, l, o, y, w, c, s\}$, and

Example of a Huffman Encoding

- The textfile is $F = helloyellowcows$
- The alphabet of characters is $\{h, e, l, o, y, w, c, s\}$, and
- the array of corresponding frequencies is $\{1, 2, 4, 3, 1, 2, 1, 1\}$.

Example of a Huffman Encoding

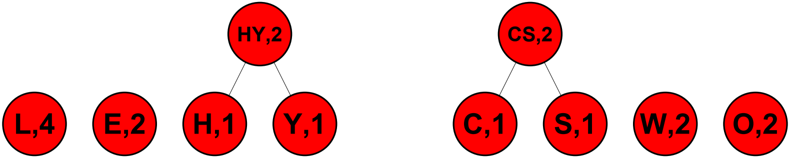
- The textfile is $F = helloyellowcows$
- The alphabet of characters is $\{h, e, l, o, y, w, c, s\}$, and
- the array of corresponding frequencies is $\{1, 2, 4, 3, 1, 2, 1, 1\}$.
- That is, h occurs once in F, e occurs twice in F, and so on.

Example of a Huffman Encoding

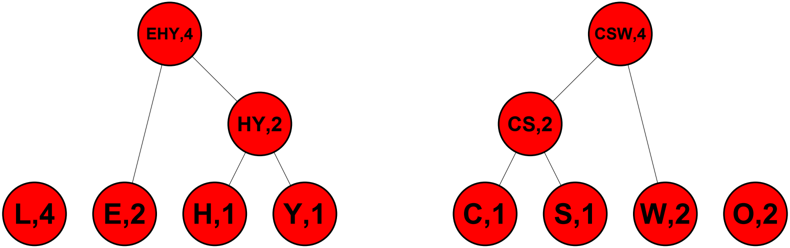
- The textfile is $F = helloyellowcows$
- The alphabet of characters is $\{h, e, l, o, y, w, c, s\}$, and
- the array of corresponding frequencies is $\{1, 2, 4, 3, 1, 2, 1, 1\}$.
- That is, h occurs once in F, e occurs twice in F, and so on.
- Here comes the Huffman Tree...

Example...

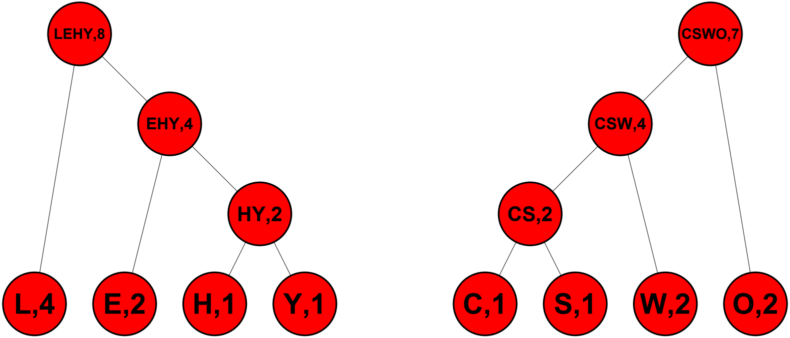
Example...



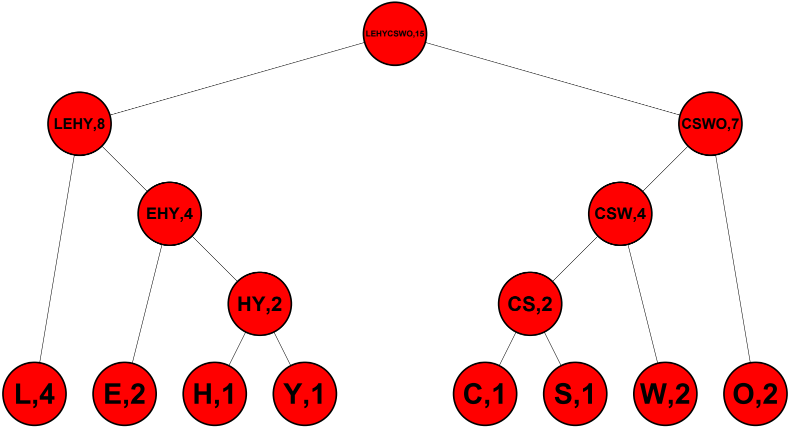
Example...



Example...



Example...

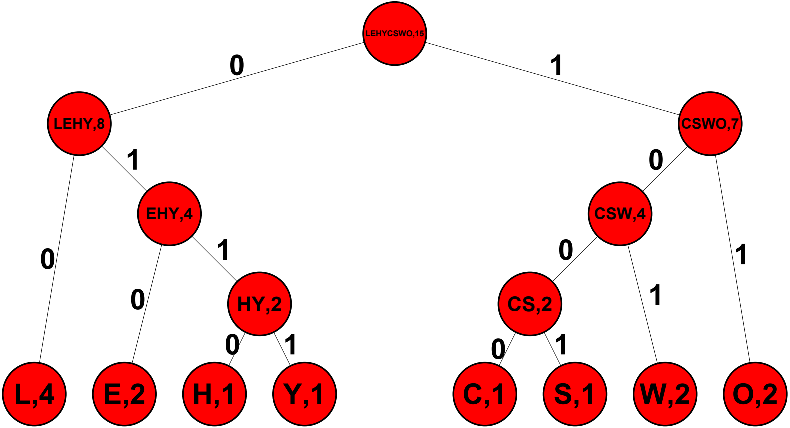


LEHYCSWO,15
LEHY,8
CSWO,7
EHY,4
CSW,4
HY,2
CS,2
L,4
E,2
H,1
Y,1
C,1
S,1
W,2
O,2

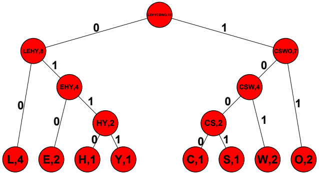
Example, continued

- Each internal node has two children; assign the left child a 0 and the right child a 1 (on edges to avoid clutter).

Example, continued

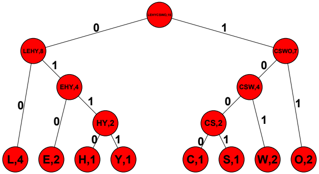


Example, almost finished



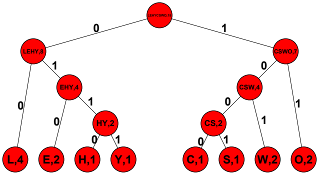
- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.

Example, almost finished



- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.
- $c_1 = \ell = 00$ cost $= 4 \times 2 = 8$ (frequency $\times$ length)

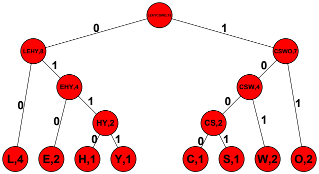
Example, almost finished



- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.

- $c_1 = \ell = 00$ cost $= 4 \times 2 = 8$ (frequency $\times$ length)
- $c_2 = e = 010$ cost $= 2 \times 3 = 6$

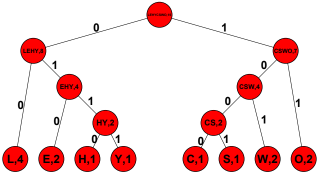
Example, almost finished



- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.

- $c_1 = \ell = 00$ cost $= 4 \times 2 = 8$ (frequency $\times$ length)
- $c_2 = e = 010$ cost $= 2 \times 3 = 6$
- $c_3 = h = 00$ cost $= 1 \times 4 = 4$

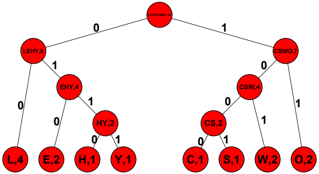
Example, almost finished



- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.

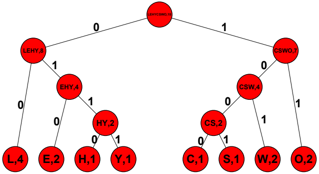
- $c_1 = \ell = 00$ cost $= 4 \times 2 = 8$ (frequency $\times$ length)
- $c_2 = e = 010$ cost $= 2 \times 3 = 6$
- $c_3 = h = 00$ cost $= 1 \times 4 = 4$
- $c_4 = y = 0111$ cost $= 1 \times 4 = 4$

Example, almost finished



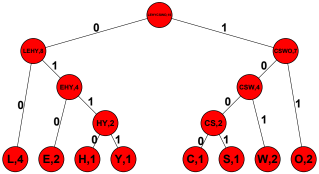
- Determine the encoding of each character α by taking the unique path from the root to the leaf corresponding to character α.
- $c_1 = \ell = 00$ cost $= 4 \times 2 = 8$ (frequency $\times$ length)
- $c_2 = e = 010$ cost $= 2 \times 3 = 6$
- $c_3 = h = 00$ cost $= 1 \times 4 = 4$
- $c_4 = y = 0111$ cost $= 1 \times 4 = 4$
- $c_5 = c = 1000$ cost $= 1 \times 4 = 4$

Example, finished



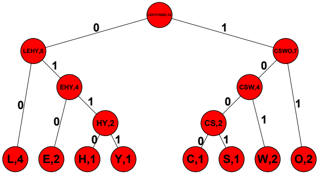
- $c_6 = s = 1001 \quad \text{cost} = 1 \times 4 = 4$

Example, finished



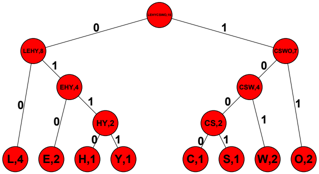
- $c_6 = s = 1001 \quad \text{cost} = 1 \times 4 = 4$
- $c_7 = w = 101 \quad \text{cost} = 2 \times 3 = 6$

Example, finished



- $c_6 = s = 1001 \quad \text{cost} = 1 \times 4 = 4$
- $c_7 = w = 101 \quad \text{cost} = 2 \times 3 = 6$
- $c_8 = o = 11 \quad \text{cost} = 3 \times 2 = 6$

Example, finished



- $c_6 = s = 1001 \quad \text{cost} = 1 \times 4 = 4$
- $c_7 = w = 101 \quad \text{cost} = 2 \times 3 = 6$
- $c_8 = o = 11 \quad \text{cost} = 3 \times 2 = 6$

- Thus, tallying the results, we see that $L(E, \mathit{helloyellowcows}) = 42$.

Implementation of Huffman Encoding

- **To Do.** Insertions, deletions, and build a binary tree.

Implementation of Huffman Encoding

- **To Do.** Insertions, deletions, and build a binary tree.
- Using a heap is an efficient method of accomplishing the former two items and

Implementation of Huffman Encoding

- **To Do.** Insertions, deletions, and build a binary tree.
- Using a heap is an efficient method of accomplishing the former two items and
- runs in time $O(lg(n)$ in the worst case.

Algorithm HuffmanEncoding(S, f)

- **Input** S (a string of characters) and f (the array of frequencies of S)
- **Output** T (the Huffman Tree for S)
- **begin**
- **insert** characters into **heap** H according to their
- frequencies (lowest to highest)
- **while** H is not empty **do**
- **if** H contains only one character X **then**
- make X the **root** of tree T
- **else** pick two characters X and Y with lowest
- frequencies f_x and f_y and **delete** them from H;
- **replace** X and Y with a new character Z whose
- frequency is $f_X + f_Y$;
- **insert** Z to H.
- make X and Y **children** of Z in T.
- **end**

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.
- **Proof.** It suffices to show

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.
- **Proof.** It suffices to show
- **Greedy Choice Property:** (1) Any optimal code tree for file F with alphabet $C = \{C_1, C_2, \ldots, C_n\}$ has a and b at maximum distance from the root of T, where

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.
- **Proof.** It suffices to show
- **Greedy Choice Property:** (1) Any optimal code tree for file F with alphabet $C = \{C_1, C_2, \ldots, C_n\}$ has a and b at maximum distance from the root of T, where
- a and b are two characters of lowest frequency in F, and

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.
- **Proof.** It suffices to show
- **Greedy Choice Property:** (1) Any optimal code tree for file F with alphabet $C = \{C_1, C_2, \ldots, C_n\}$ has a and b at maximum distance from the root of T, where
- a and b are two characters of lowest frequency in F, and
- **Optimal Substructure Property:** (2) Any optimal code tree for F is optimal is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$

Proof that the Huffman Tree Minimizes $L(E, F)$

- **Theorem.** The Huffman Tree produces an optimal code. That is, the encoding E derived by Algorithm HuffmanEncoding will minimize $L(E, F)$.
- **Proof.** It suffices to show
- **Greedy Choice Property:** (1) Any optimal code tree for file F with alphabet $C = \{C_1, C_2, \ldots, C_n\}$ has a and b at maximum distance from the root of T, where
- a and b are two characters of lowest frequency in F, and
- **Optimal Substructure Property:** (2) Any optimal code tree for F is optimal is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$
- with the frequency of new character $\underline{ab}$ given by $f_a + f_b$.

Proof of Huffman Encoding correctness, continued

- **Verification of (1), Greedy Choice Property.**

Proof of Huffman Encoding correctness, continued

- **Verification of (1), Greedy Choice Property.**

- Let T be an optimal code tree for C and suppose c and d are siblings farthest from the root of T.

Proof of Huffman Encoding correctness, continued

- **Verification of (1), Greedy Choice Property.**
- Let T be an optimal code tree for C and suppose c and d are siblings farthest from the root of T.
- Swap a and b (lowest frequency characters) with c and d respectively to form new tree T'.

Proof of Huffman Encoding correctness, continued

- **Verification of (1), Greedy Choice Property.**
- Let T be an optimal code tree for C and suppose c and d are siblings farthest from the root of T.
- Swap a and b (lowest frequency characters) with c and d respectively to form new tree T'.
- Let $\ell_T(x)$ be the length of encoded character x in tree T, and

Proof of Huffman Encoding correctness, continued

- **Verification of (1), Greedy Choice Property.**
- Let T be an optimal code tree for C and suppose c and d are siblings farthest from the root of T.
- Swap a and b (lowest frequency characters) with c and d respectively to form new tree T'.
- Let $\ell_T(x)$ be the length of encoded character x in tree T, and
- Let $\ell_{T'}(x)$ be the length of encoded character x in tree T'.

Proof of Huffman Encoding correctness, continued

- Then $COST(T) - COST(T')$

Proof of Huffman Encoding correctness, continued

- Then $COST(T) - COST(T')$
- $= \sum_{x \in C} f_x \ell_T(x) - \sum_{x \in C} f_x \ell_{T'}(x)$

Proof of Huffman Encoding correctness, continued

- Then $COST(T) - COST(T')$
- $= \sum_{x \in C} f_x \ell_T(x) - \sum_{x \in C} f_x \ell_{T'}(x)$
- $= f_a(\ell_T(a) - \ell_{T'}(a)) + f_b(\ell_T(b) - \ell_{T'}(b))$
- $+ f_c(\ell_T(c) - \ell_{T'}(c)) + f_d(\ell_T(d) - \ell_{T'}(d)) = (*)$.

Proof of Huffman Encoding correctness, continued

- Then $COST(T) - COST(T')$
- $= \sum_{x \in C} f_x \ell_T(x) - \sum_{x \in C} f_x \ell_{T'}(x)$
- $= f_a(\ell_T(a) - \ell_{T'}(a)) + f_b(\ell_T(b) - \ell_{T'}(b))$
- $+ f_c(\ell_T(c) - \ell_{T'}(c)) + f_d(\ell_T(d) - \ell_{T'}(d)) = (*)$.
- But $\ell_{T'}(a) = \ell_T(c)$ and $\ell_{T'}(b) = \ell_T(d)$ and

Proof of Huffman Encoding correctness, continued

- Then $COST(T) - COST(T')$
- $= \sum_{x \in C} f_x \ell_T(x) - \sum_{x \in C} f_x \ell_{T'}(x)$
- $= f_a(\ell_T(a) - \ell_{T'}(a)) + f_b(\ell_T(b) - \ell_{T'}(b))$
- $+ f_c(\ell_T(c) - \ell_{T'}(c)) + f_d(\ell_T(d) - \ell_{T'}(d)) = (*)$.
- But $\ell_{T'}(a) = \ell_T(c)$ and $\ell_{T'}(b) = \ell_T(d)$ and
- $\ell_{T'}(c) = \ell_T(a)$ and $\ell_{T'}(d) = \ell_T(b)$.

Proof of Huffman Encoding correctness, continued

- Thus (*) on the previous slide can be rewritten as

Proof of Huffman Encoding correctness, continued

- Thus (*) on the previous slide can be rewritten as
- $f_a(\ell_T(a) - \ell_T(c)) + f_b(\ell_T(b) - \ell_T(d)) + f_c(\ell_T(c) - \ell_T(a)) + f_d(\ell_T(d) - \ell_T(b))$

Proof of Huffman Encoding correctness, continued

- Thus (*) on the previous slide can be rewritten as
- $f_a(\ell_T(a) - \ell_T(c)) + f_b(\ell_T(b) - \ell_T(d)) + f_c(\ell_T(c) - \ell_T(a)) + f_d(\ell_T(d) - \ell_T(b))$
- $= (f_c - f_a)(\ell_T(c) - \ell_T(a)) + (f_d - f_b)(\ell_T(d) - \ell_T(b))$

Proof of Huffman Encoding correctness, continued

- Thus (*) on the previous slide can be rewritten as
- $f_a(\ell_T(a) - \ell_T(c)) + f_b(\ell_T(b) - \ell_T(d)) + f_c(\ell_T(c) - \ell_T(a)) + f_d(\ell_T(d) - \ell_T(b))$
- $= (f_c - f_a)(\ell_T(c) - \ell_T(a)) + (f_d - f_b)(\ell_T(d) - \ell_T(b))$
- ≥ 0 because
  - $f_d \geq f_b$ and $f_c \geq f_a$, and
  - $\ell_T(d) \geq \ell_T(b)$ and $\ell_T(c) \geq \ell_T(a)$.

Proof of Huffman Encoding correctness, continued

- Thus (*) on the previous slide can be rewritten as
- $f_a(\ell_T(a) - \ell_T(c)) + f_b(\ell_T(b) - \ell_T(d)) + f_c(\ell_T(c) - \ell_T(a)) + f_d(\ell_T(d) - \ell_T(b))$
- $= (f_c - f_a)(\ell_T(c) - \ell_T(a)) + (f_d - f_b)(\ell_T(d) - \ell_T(b))$
- ≥ 0 because
  - $f_d \geq f_b$ and $f_c \geq f_a$, and
  - $\ell_T(d) \geq \ell_T(b)$ and $\ell_T(c) \geq \ell_T(a)$.
- This completes the verification of (1) (the Greedy Choice Property).

Proof of Huffman Encoding correctness, continued

- **Subconclusion.** We may assume WLOG that any optimal code tree has low frequency characters at maximum distance from the root of T.

Proof of Huffman Encoding correctness, continued

- **Subconclusion.** We may assume WLOG that any optimal code tree has low frequency characters at maximum distance from the root of T.
- That is, low frequency characters are leaves of T.

Proof of Huffman Encoding correctness, continued

- **Verification of (2), The Optimal Substructure Property.**

Proof of Huffman Encoding correctness, continued

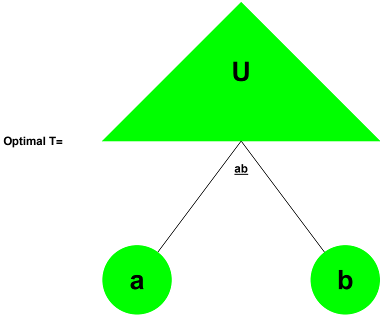
- **Verification of (2), The Optimal Substructure Property.**

- Let T be any optimal code tree for file F with alphabet C.

Proof of Huffman Encoding correctness, continued

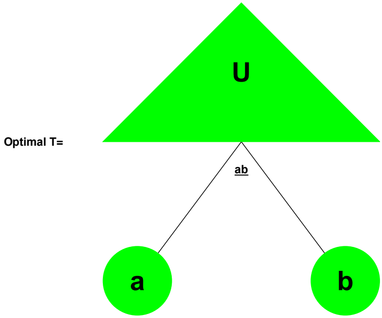
- **Verification of (2), The Optimal Substructure Property.**

- Let T be any optimal code tree for file F with alphabet C.

- 

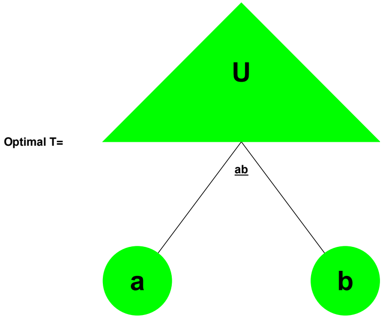
Proof of Huffman Encoding correctness, continued

- **Verification of (2), The Optimal Substructure Property.**
- Let T be any optimal code tree for file F with alphabet C.



- BWOC suppose tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is not optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$.

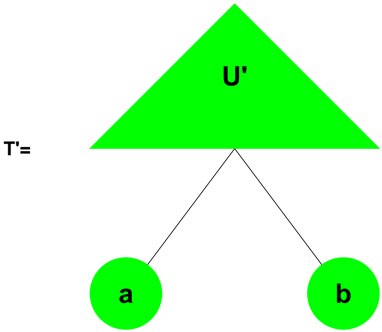
Proof of Huffman Encoding correctness, continued

- **Verification of (2), The Optimal Substructure Property.**
- Let T be any optimal code tree for file F with alphabet C.
- 

- BWOC suppose tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is not optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$.

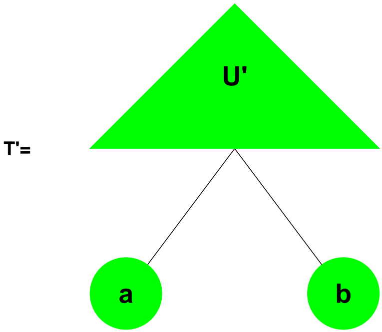
Proof of Huffman Encoding correctness, continued

- Consider T', an optimal code tree for file F with alphabet C.



Proof of Huffman Encoding correctness, continued

- Consider T', an optimal code tree for file F with alphabet C.

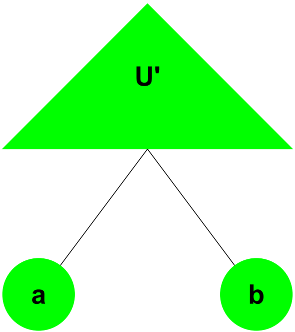


- By earlier work we know a and b are leaves of T', and

Proof of Huffman Encoding correctness, continued

- Consider T', an optimal code tree for file F with alphabet C.

T'=



- By earlier work we know a and b are leaves of T', and
- call the remainder of the tree (minus a and b) U'.

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b) \ell_T(a)$

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b) \ell_T(a)$
- (because a and b are siblings, in which case $\ell_T(a) = \ell_T(b)$)

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b) \ell_T(a)$
- (because a and b are siblings, in which case $\ell_T(a) = \ell_T(b)$)
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b)(\ell_U(\underline{ab}) + 1)$

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b)\ell_T(a)$
- (because a and b are siblings, in which case $\ell_T(a) = \ell_T(b)$)
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b)(\ell_U(\underline{ab}) + 1)$
- (because a is one level below $\underline{ab}$ in U)

Proof of Huffman Encoding correctness, continued

- Then $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) \quad + f_a \ell_T(a) + f_b \ell_T(b)$
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b) \ell_T(a)$
- (because a and b are siblings, in which case $\ell_T(a) = \ell_T(b)$)
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + (f_a + f_b)(\ell_U(\underline{ab}) + 1)$
- (because a is one level below $\underline{ab}$ in U)
- $= \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$

Proof of Huffman Encoding correctness, continued

- So far we have
  $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$

Proof of Huffman Encoding correctness, continued

- So far we have
  $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$

- $= COST(U) + f_{\underline{ab}}$.

Proof of Huffman Encoding correctness, continued

- So far we have $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$

- $= COST(U) + f_{\underline{ab}}$.

- Similarly, $COST(T') = COST(U') + f_{\underline{ab}}$.

Proof of Huffman Encoding correctness, continued

- So far we have $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$
- $= COST(U) + f_{\underline{ab}}$.
- Similarly, $COST(T') = COST(U') + f_{\underline{ab}}$.
- We have assumed that U is not optimal and thus $COST(U') < COST(U)$ in which case

Proof of Huffman Encoding correctness, continued

- So far we have
  $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$

- $= COST(U) + f_{\underline{ab}}$.

- Similarly, $COST(T') = COST(U') + f_{\underline{ab}}$.

- We have assumed that U is not optimal and thus $COST(U') < COST(U)$ in which case

- $COST(U') + f_{\underline{ab}} < COST(U) + f_{\underline{ab}}$

Proof of Huffman Encoding correctness, continued

- So far we have
  $COST(T) = \sum_{x \in C \setminus \{a,b\}} f_x \ell_T(x) + f_{\underline{ab}} \ell_U(\underline{ab}) + f_{\underline{ab}}$
- $= COST(U) + f_{\underline{ab}}$.
- Similarly, $COST(T') = COST(U') + f_{\underline{ab}}$.
- We have assumed that U is not optimal and thus $COST(U') < COST(U)$ in which case
- $COST(U') + f_{\underline{ab}} < COST(U) + f_{\underline{ab}}$
- $\Rightarrow COST(T') < COST(T)$, a contradiction (since T is optimal).

Conclusion

- The code tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$,

Conclusion

- The code tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$,
- and we have verified property (2), the optimal substructure property.

Conclusion

- The code tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$,
- and we have verified property (2), the optimal substructure property.
- Hence the HuffmanEncoding Algorithm is optimal.

Conclusion

- The code tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$,
- and we have verified property (2), the optimal substructure property.
- Hence the HuffmanEncoding Algorithm is optimal.
- **QED**

Conclusion

- The code tree $U = T \setminus \{a, b\} \cup \{\underline{ab}\}$ is optimal for $C \setminus \{a, b\} \cup \{\underline{ab}\}$,
- and we have verified property (2), the optimal substructure property.
- Hence the HuffmanEncoding Algorithm is optimal.
- **QED**
- **Exercise:** What is the run time of Algorithm HuffmanEncoding?

Next

Graph Theory and Graph Algorithms