Chapter 3: Greedy Algorithm Design Technique - II

Devesh C Jinwala , IIT Jammu, India

October 14, 2019

Design and Analysis of Algorithms
IIT Jammu, Jammu

Fixed length Character Coding Schemes

- Fixed length codes or variable length codes
- Consider first Fixed length codes like ASCII.
  - How many bits are required to encode C distinct characters ?
  - Consider an alphabet = <a, u, x, z> and an example string *aaxuaxz* ?
  - How many characters are required to encode using 8-bit code?
  - How many characters are required to encode using 2-bit code?
  - Now consider a string of 1000 characters from the same alphabet..
  - How many bits are required for this string in 8-bit ASCII code ?
  - How many bits are required for this string in 2-bit fixed length code ?

Variable length Character Coding Schemes

- Are we exploiting the frequency of occurence of the characters in this scheme ?

Obviously the variable length code desired for longer strings.

Variable length Character Coding Schemes

- Are we exploiting the frequency of occurence of the characters in this scheme ?
- Say a occurs 996 times, x occurs twice and u and z occur once each i.e. 1000 characters.

Obviously the variable length code desired for longer strings.

Variable length Character Coding Schemes

- Are we exploiting the frequency of occurence of the characters in this scheme ?
- Say a occurs 996 times, x occurs twice and u and z occur once each i.e. 1000 characters.
- What are the bits required for the string *aaxuaxz* ?

Obviously the variable length code desired for longer strings.

Variable length Character Coding Schemes

- Are we exploiting the frequency of occurence of the characters in this scheme ?
- Say a occurs 996 times, x occurs twice and u and z occur once each i.e. 1000 characters.
- What are the bits required for the string *aaxuaxz* ?
- Assume a=0, x=10, u=110, z=111 is the bit representation.

Obviously the variable length code desired for longer strings.

Variable length Character Coding Schemes

- Are we exploiting the frequency of occurence of the characters in this scheme ?
- Say a occurs 996 times, x occurs twice and u and z occur once each i.e. 1000 characters.
- What are the bits required for the string *aaxuaxz* ?
- Assume a=0, x=10, u=110, z=111 is the bit representation.
- What are the bits required for the 1000 character string with these frequencies ?

Obviously the variable length code desired for longer strings.

Prefix Code by Huffman

- But, how to decode a string of variable length ??
- Prefix code first defined by Huffman
- definition
  - e.g. 1 and 01 are prefix codes
  - e.g. 1, 01, 001, 000 are prefix codes
  - e.g. 0, 10, 110, 111 are prefix codes
- definition

Decoding Variable length code encoded strings

Steps for decoding a variable length encoded string

- Scan the bit sequence from left to right.
- As soon as you have seen enough bits to match the encoding of some letter, output this as the first letter of the text. This must be the correct first letter, since no shorter or longer prefix of the bit sequence could encode any other letter.
- Now delete the corresponding set of bits from the front of the message and iterate.

Decoding Variable length code encoded strings

Steps for decoding a variable length encoded string

- Scan the bit sequence from left to right.
- As soon as you have seen enough bits to match the encoding of some letter, output this as the first letter of the text. This must be the correct first letter, since no shorter or longer prefix of the bit sequence could encode any other letter.
- Now delete the corresponding set of bits from the front of the message and iterate.

- Consider the alphabet S = a, b, c, d, e and the encoding $\gamma(a) = 11$ $\gamma(b) = 01$ $\gamma(c) = 001$ $\gamma(d) = 10$ and $\gamma(e) = 000$. Is this code a prefix code ? IF so, show to what the encoding represented by 0010000011101 decodes to ?

Decoding Variable length code encoded strings

Steps for decoding a variable length encoded string

- Scan the bit sequence from left to right.
- As soon as you have seen enough bits to match the encoding of some letter, output this as the first letter of the text. This must be the correct first letter, since no shorter or longer prefix of the bit sequence could encode any other letter.
- Now delete the corresponding set of bits from the front of the message and iterate.

- Consider the alphabet S = a, b, c, d, e and the encoding $\gamma(a) = 11$ $\gamma(b) = 01$ $\gamma(c) = 001$ $\gamma(d) = 10$ and $\gamma(e) = 000$. Is this code a prefix code ? IF so, show to what the encoding represented by 0010000011101 decodes to ?
- Consider the alphabet S = a, b, c, d and the encoding $\gamma(a) = 0$ $\gamma(b) = 10$ $\gamma(c) = 110$ $\gamma(d) = 111$. How would the string 1101001101000 be decoded ?

Prefix codes and its TBL

Average Bits per Letter

The total bits of a prefix code encoded string c is the sum over all symbols of frequency of each character c, times the number of bits of encoding of c.

Given that C is the alphabet, c is the prefix code function i.e. c(x) is the encoding of any character x ϵC, f(x) is the frequency of occurrence of a character c ϵC, we have

Total Bit Length $= \sum_{x \epsilon C} f(x)|c(x)|$ for every $x \epsilon C$

Goal: To find a prefix code representation that has the lowest possible average bits per letter.
We anyway need suitable data structure to represent variable length code.

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.
- The binary tree representing this prefix code would be as:

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.
- The binary tree representing this prefix code would be as:
- Note that only the leaves of this tree have the label, the internal nodes do not have.

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.
- The binary tree representing this prefix code would be as:
- Note that only the leaves of this tree have the label, the internal nodes do not have.
- An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.
- The binary tree representing this prefix code would be as:
- Note that only the leaves of this tree have the label, the internal nodes do not have.
- An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.
- A prefix code can always be represented as a full binary tree/trie

Data Structure for prefix codes

- Let us represent a prefix code as a binary tree
- Consider the alphabet as : c(a) = 11 c(e) = 01 c(k) = 001 c(l) = 10 c(u) = 000.
- The binary tree representing this prefix code would be as:
- Note that only the leaves of this tree have the label, the internal nodes do not have.
- An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.
- A prefix code can always be represented as a full binary tree/trie
- A binary tree representation can give optimal code. But let us verify....

External Weighted Path length of a tree

- Let C be the alphabet, $|C|$ be the no of characters in the alphabet. How many leaves the corresponding binary tree would have ?

External Weighted Path length of a tree

- Let C be the alphabet, $|C|$ be the no of characters in the alphabet. How many leaves the corresponding binary tree would have ?
- Then, no of bits required to encode a file is given by, for the tree T as,
  TBL(T) $= \sum_{x \epsilon C} f(x) d_{\tau}(x)$ for every $x \epsilon C$

External Weighted Path length of a tree

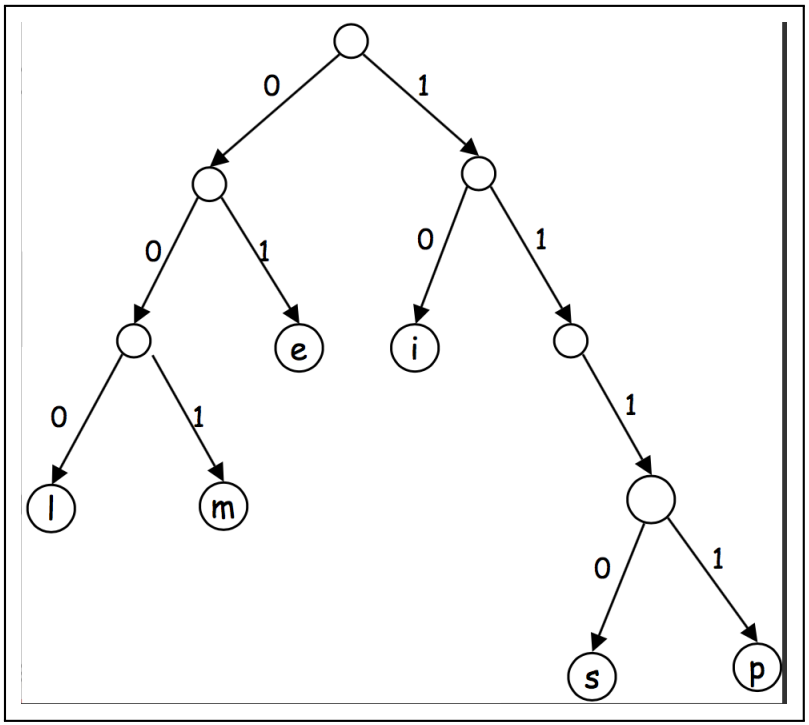
- Let C be the alphabet, $|C|$ be the no of characters in the alphabet. How many leaves the corresponding binary tree would have ?
- Then, no of bits required to encode a file is given by, for the tree T as,
  TBL(T) $= \sum_{x \epsilon C} f(x) d_{\tau}(x)$ for every $x \epsilon C$
- def: Weighted External path length of a binary tree

External Weighted Path length of a tree

- Let C be the alphabet, $|C|$ be the no of characters in the alphabet. How many leaves the corresponding binary tree would have ?
- Then, no of bits required to encode a file is given by, for the tree T as,
  TBL(T) $= \sum_{x \epsilon C} f(x) d_{\tau}(x)$ for every $x \epsilon C$
- def: Weighted External path length of a binary tree
- TBL(T) = Cost of the tree T

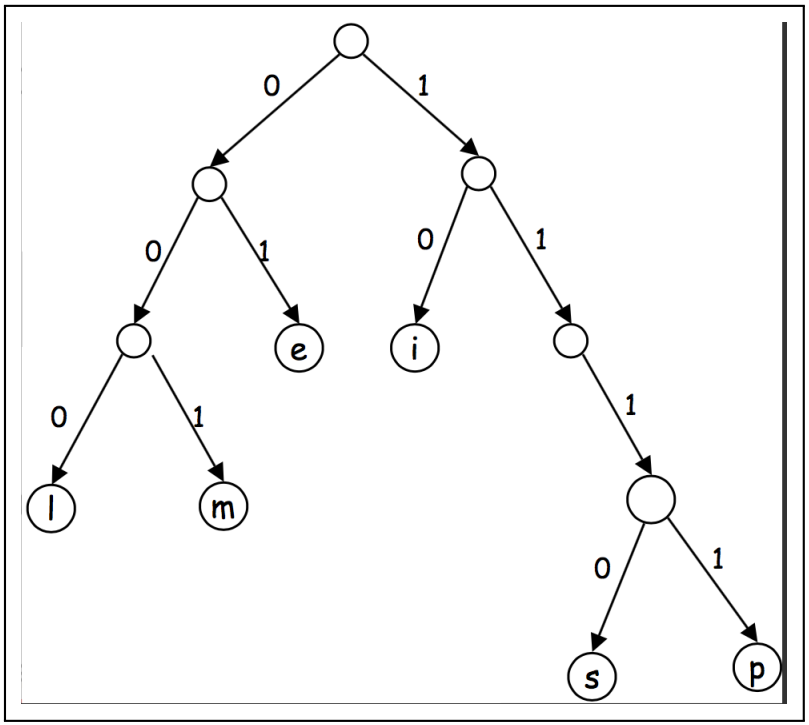
Prefix codes and Full binary trees

- Consider now another tree as shown for the code viz.



Prefix codes and Full binary trees

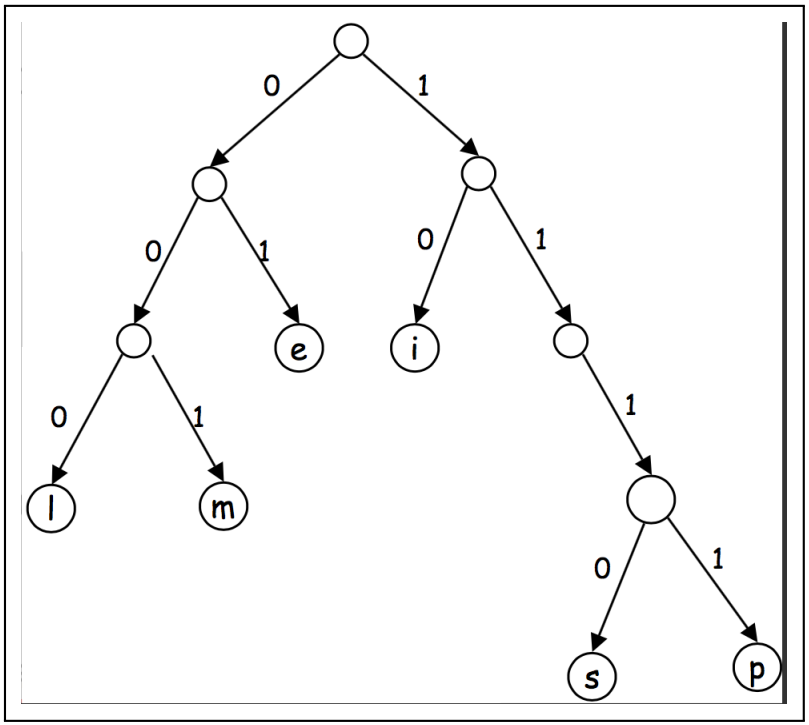
- Consider now another tree as shown for the code viz.



- IS the code used in this a prefix code ?

Prefix codes and Full binary trees

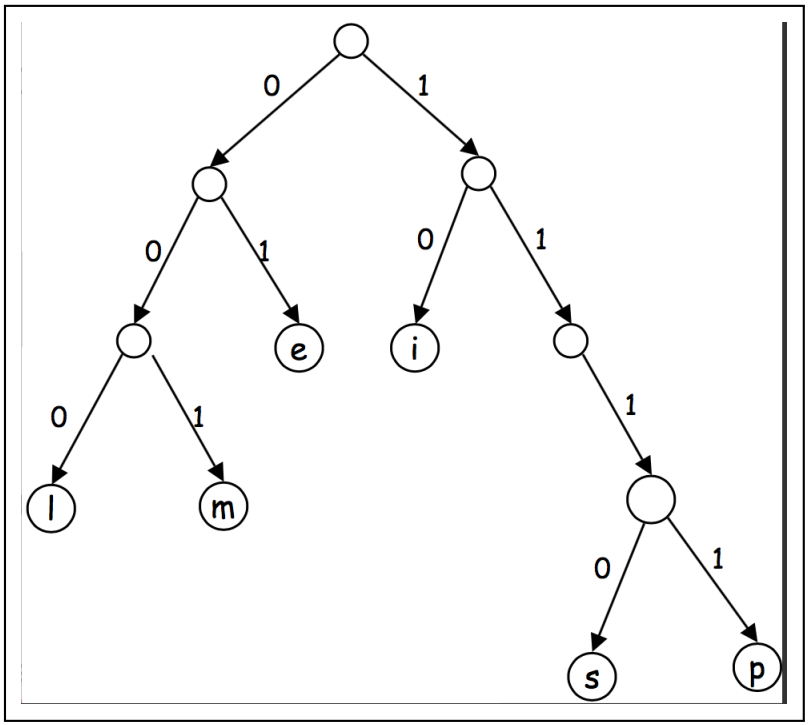
- Consider now another tree as shown for the code viz.



- IS the code used in this a prefix code ?
- Assuming each character occurs only once in a string, what is the TBL(T) = Cost of the tree T ?

Prefix codes and Full binary trees

- Consider now another tree as shown for the code viz.



- IS the code used in this a prefix code ?
- Assuming each character occurs only once in a string, what is the TBL(T) = Cost of the tree T ?
- *Can this prefix code be made more efficient*? How ?

Prefix codes and Full binary trees...

- The issue is *how can this prefix code be made more efficient*?

Prefix codes and Full binary trees...

- The issue is *how can this prefix code be made more efficient*?
- Assuming each character occurs only once in a string, now what is the TBL(T) = Cost of the tree T ?

Prefix codes and Full binary trees...

- The issue is *how can this prefix code be made more efficient*?
- Assuming each character occurs only once in a string, now what is the TBL(T) = Cost of the tree T ?
- def: Extended binary tree

Prefix codes and Full binary trees...

- The issue is *how can this prefix code be made more efficient*?
- Assuming each character occurs only once in a string, now what is the TBL(T) = Cost of the tree T ?
- def: Extended binary tree
- def: Full binary tree or a trie - definition

Prefix codes and Full binary trees...

- The issue is *how can this prefix code be made more efficient*?
- Assuming each character occurs only once in a string, now what is the TBL(T) = Cost of the tree T ?
- def: Extended binary tree
- def: Full binary tree or a trie - definition
- Show that a prefix code can always be represented as a full binary tree.

Huffman Tree construction illustrations

- Construct the full binary tree for the variable length prefix codes:

Huffman Tree construction illustrations

- Construct the full binary tree for the variable length prefix codes:
  - a = 00, x = 01, u = 10, z = 11 and the frequency being 996,2,1,1 respectively

Huffman Tree construction illustrations

- Construct the full binary tree for the variable length prefix codes:
  - a = 00, x = 01, u = 10, z = 11 and the frequency being 996,2,1,1 respectively
  - a = 0, x = 10, u = 110, z = 111 with the same frequency as above

Constructing the tree: Huffman Encoding

- Defining Huffman tree
- Construct the Huffman tree for the following

a	b	c	d	e	f
6	2	3	3	4	9

The Huffman Tree Construction pseudocode

```
The Algorithm Huffman-Tree(C)
1.n = |C|
2.for i = 1 to n-1
3.        do z = ALLOCATE_NODE()
4.        x = left(z) = EXTRACT_MIN(Q)
5.        y = right(z) = EXTRACT_MIN(Q)
6.        f[z] = f[x] + f[y]
7.        INSERT(Q,z)
8.return EXTRACT_MIN(Q)
```

Why (n-1) in line 2?

Why the last line ?

The Huffman Tree Construction pseudocode

```
The Algorithm HEAP–EXTRACT–MIN(A)
1. if heap_size[A] < 1
2.    then return error heap underflow
3. min = [A]
4. A[1] = A[heap−size[A]]
5. heap−size[A] = heap−size[A]      1
6. HEAPIFY(A, 1)
7. return min
```

Time Complexity

- HEAP-EXTRACT-MIN & INSERT take O(lg n) time on Q with n objects
- Therefore, $T(n) = \sum_{i=1}^{n} lg\ i = O(lg(n!)) = O(n\ lg\ n)$

Optimal Merge Patterns

Problem definition

To merge n sorted files in such a way that the fewest number of comparisons are required while merging them.

Say Given files x_1, x_2, x_3, x_4, x_5 with $|x1|$=20, $|x2|$=30, $|x3|$=10, $|x4|$=5, $|x5|$=30. Show haw can they be optimally merged.

What are the various record moves required merging these files in different patterns ?

Guessing Game

- Consider the game of guessing a chosen object from n possibilities (say, an integer between 1 and n) by asking questions answerable by yes or no.
- Different strategies for playing this game can be modeled by decision trees

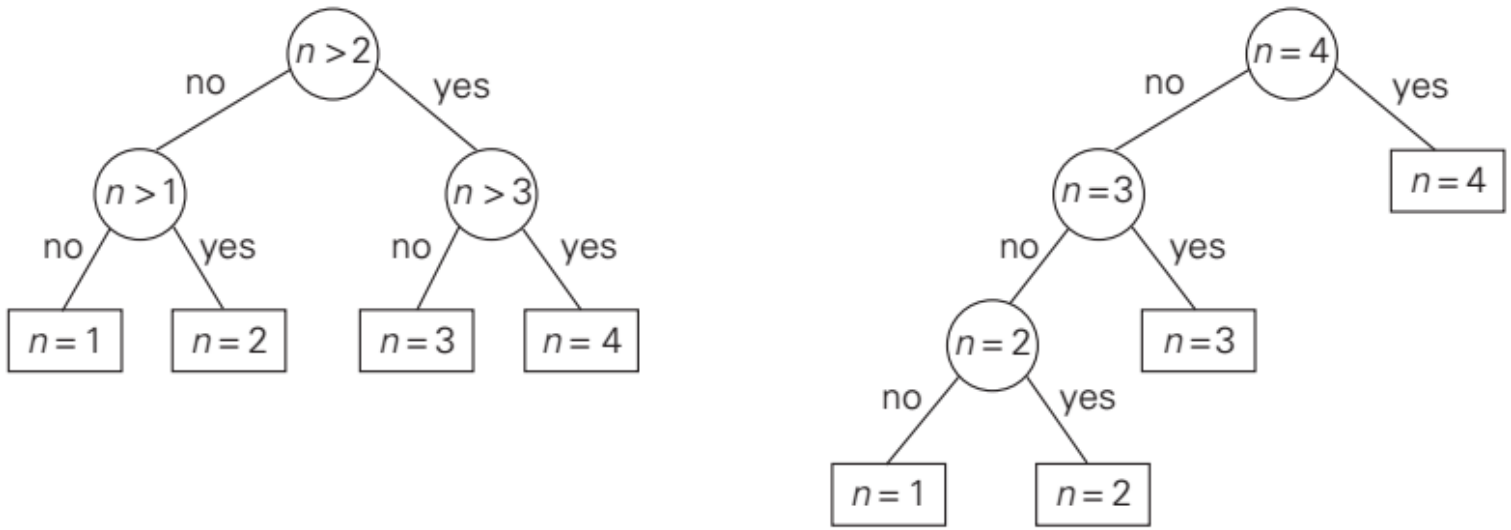


FIGURE 9.13 Two decision trees for guessing an integer between 1 and 4.

Prefix codes and Full binary trees

- Exercise 16-3.2: Theorem1: Prove that a a binary tree that is not full cannot correspond to an optimal prefix code. That is prove that a full binary tree representation always gives optimal code

Prefix codes and Full binary trees

- Exercise 16-3.2: Theorem1: Prove that a a binary tree that is not full cannot correspond to an optimal prefix code. That is prove that a full binary tree representation always gives optimal code
- Exercise 16-3.4: Lemma1: Prove that we can also express the total cost of a tree for a code as the sum, over all internal nodes, of the combined frequencies of the two children of the node.

Correctness Proof: Huffman coding

Lemma 1

Let C be an alphabet in which each character c ϵ C has frequency $f(c)$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

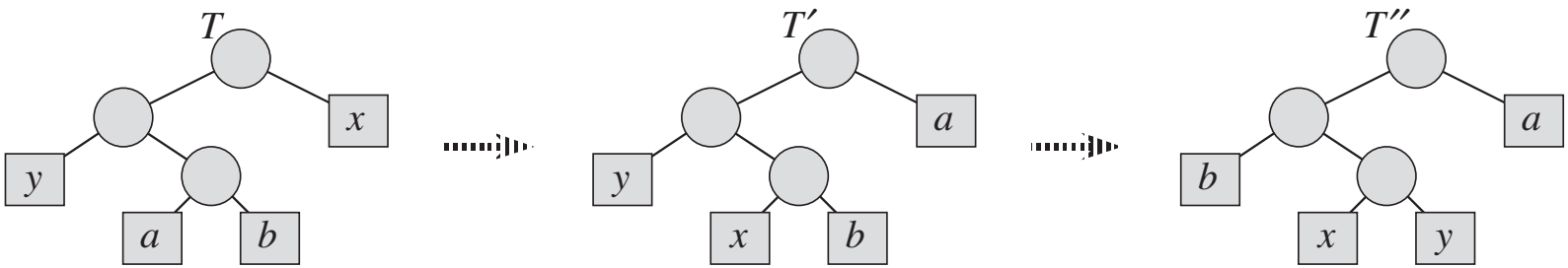


Figure 16.6 An illustration of the key step in the proof of Lemma 16.2. In the optimal tree T, leaves a and b are two siblings of maximum depth. Leaves x and y are the two characters with the lowest frequencies; they appear in arbitrary positions in T. Assuming that $x \neq b$, swapping leaves a and x produces tree T', and then swapping leaves b and y produces tree T''. Since each swap does not increase the cost, the resulting tree T'' is also an optimal tree.

Correctness Proof: Huffman coding...

Lemma 1

Let C be an alphabet in which each character c ϵ C has frequency $f(c)$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

- 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 TBL(T) i.e. 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 - hence the greedy choice property proved with Lemma1.

Correctness Proof: Huffman coding...

The Huffman's algorithm is correct

To prove that the greedy algorithm HUFFMAN is correct, we show that the problem of determining an optimal prefix code exhibits the greedy-choice and optimal-substructure properties.

- Step 1: Show that this problem satisfies the greedy choice property, that is, if a greedy choice is made by Huffman's algorithm, an optimal solution remains possible.
- Step 2: Show that this problem has an optimal substructure property, that is, an optimal solution to Huffman's algorithm contains optimal solution to subproblems.
- Step 3: Conclude correctness of Huffman's algorithm using step 1 and step 2.

Correctness Proof: Huffman coding ...

Lemma 2

Let C be a given alphabet with frequency c.freq defined for each character c ϵ C. Let x and y be two characters in C with minimum frequency.

Let C' be the alphabet C with the characters x and y removed and a new character added, so that C' = C - {x,y} U z. Define f for C' as for C, except that f(z) = f(x) + f(y).

Let T' be any tree representing an optimal prefix code for the alphabet C'.

Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

Other Compression Techniques

- IT is necessary to include the coding table into the encoded text to make its decoding possible.
- This drawback can be overcome by using dynamic Huffman encoding in which the coding tree is updated each time a new symbol is read from the source text.
- Modern schemes likeLempel-Ziv algorithms assign codewords not to individual symbols but to strings of symbols, allowing them to achieve better and more robust compressions in many applications.

Blank

Blank

Blank