Greedy Algorithms
Huffman coding

Optimization Problems

- A problem that may have many feasible solutions.
- Each solution has a value
- In maximization problem, we wish to find a solution to maximize the value
- In the minimization problem, we wish to find a solution to minimize the value

The Diet Problem

	Carbs	Protein	Fat	Iron	Cost
1 slice bread	30	5	1.5	10	30¢
1 cup yogurt	10	9	2.5	0	80¢
2tsp Peanut Butter	6	8	18	6	20¢
US RDA Minimum	300	50	70	100	

Minimize $30\,x_1 + 80\,x_2 + 20\,x_3$

s.t.
$$30x_1 + 10\,x_2 + 6\,x_3 \geq 300$$
$$5x_1 + 9x_2 + 8x_3 \geq 50$$
$$1.5x_1 + 2.5\,x_2 + 18\,x_3 \geq 70$$
$$10x_1 + 6\,x_3 \geq 100$$
$$x_1, x_2, x_3 \geq 0$$

Data Compression

Huffman Coding

- A technique to compress data effectively
  - Usually between 20%-90% compression
- Lossless compression
  - No information is lost
  - When decompress, you get the original file

- Suppose we have 1000000000 (1G) character data file that we wish to include in an email.
- Suppose file only contains 26 letters {a,…,z}. $5(f_a + f_b + \Lambda\ f_z)$
- Suppose each letter α in {a,…,z} occurs with frequency f_α.
- Suppose we encode each letter by a binary code
- If we use a fixed length code, we need 5 bits for each character
- The resulting message length is

Huffman Codes

Main Idea: Frequency-Based Encoding

Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre-cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

"Fifteen men on the dead man's chest–Yo-ho-ho, and a bottle of rum!" in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

"This is a handy cove," says he at length; "and a pleasant sittyated grog-shop. Much company, mate?"

My father told him no, very little company, the more was the pity.

"Well, then," said he, "this is the berth for me. Here you, matey," he cried to the man who trundled the barrow; "bring up alongside and help up my chest. I'll stay here a bit," he continued. "I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at – there"; and he threw down three or four gold pieces on the threshold. "You can tell me when I've worked through that," says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

- Assume in this file only 6 characters appear
  - E, A, C, T, K, N
- The frequencies are:

Character	Frequency
E	10,000
A	4,000
C	300
T	200
K	100
N	100

- **Option I (No Compression)**
  - Each character = 1 Byte (8 bits)
  - Total file size = 14,700 * 8 = 117,600 bits

Character	Fixed Encoding
E	000
A	001
C	010
T	100
K	110
N	100

- **Option 2 (Fixed size compression)**
  - We have 6 characters, so we need 3 bits to encode them
  - Total file size = 14,700 * 3 = 44,100 bits

- **Option 3 (Huffman compression)**
  - Variable-length compression
  - Assign shorter codes to more frequent characters and longer codes to less frequent characters
  - Total file size:

(10,000 x 1) + (4,000 x 2) + (300 x 3) + (200 x 4) + (100 x 5) + (100 x 5) = 20,700 bits

Char.	HuffmanEncoding
E	0
A	10
C	110
T	1110
K	11110
N	11111

Which characters should we assign shorter codes; which characters will have longer codes?

Data Compression: A Smaller Example

- Suppose the file only has 6 letters {a,b,c,d,e,f} with frequencies

a	*b*	*c*	*d*	*e*	*f*	
.45	.13	.12	.16	.09	.05	
000	001	010	011	100	101	Fixed length
0	101	100	111	1101	1100	Variable length

- Fixed length 3G=3000000000 bits
- Variable length

$$(.45 \bullet 1 + .13 \bullet 3 + .12 \bullet 3 + .16 \bullet 3 + .09 \bullet 4 + .05 \bullet 4) = 2.24G$$

A variable-length coding for characters

- More frequent characters □ shorter codes
- Less frequent characters □ longer codes

It is not like ASCII coding where all characters have the same coding length (8 bits)

Two main questions

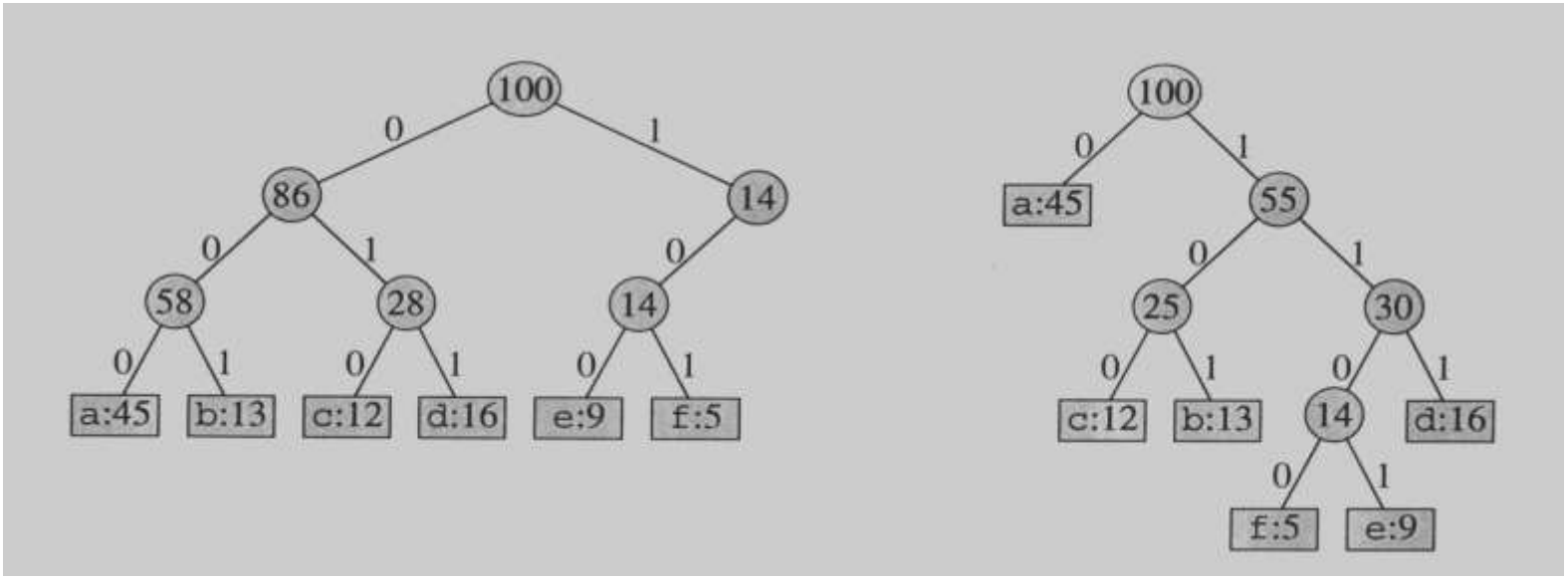
- How to assign codes *(Encoding process)?*
- How to decode (from the compressed file, generate the original file) *(Decoding process)?*

How to Decode?

- At first it is not obvious how decoding will happen, but this is possible if we use prefix codes

Prefix Codes

- No encoding of a character can be the prefix of the longer encoding of another character, for example, we could not encode *t* as 01 and *x* as 01101 since 01 is a prefix of 01101
- By using a binary tree representation we will generate prefix codes provided all letters are leaves



Some Properties

- Prefix codes allow easy decoding
  - Given a: 0, b: 101, c: 100, d: 111, e: 1101, f: 1100
  - Decode 001011101 going left to right, 0|01011101, a|0|1011101, a|a|101|1101, a|a|b|1101, a|a|b|e
- An optimal code must be a full binary tree (a tree where every internal node has two children)
- For *C* leaves there are *C-1* internal nodes
- The number of bits to encode a file is

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

where $f(c)$ is the freq of c, $d_T(c)$ is the tree depth of c, which corresponds to the code length of c

Optimal Prefix Coding Problem

- Input: Given a set of n letters $(c_1,\ldots, c_n)$ with frequencies $(f_1,\ldots, f_n)$.

- Construct a full binary tree T to define a prefix code that minimizes the average code length

$$\text{Average}(T) = \sum_{i=1}^{n} f_i \bullet \text{length}_T(c_i)$$

Greedy Algorithms

- Many optimization problems can be solved using a greedy approach
  - The basic principle is that local optimal decisions may may be used to build an optimal solution
  - But the greedy approach may not always lead to an optimal solution overall for all problems
- A *greedy algorithm* always makes the choice that looks best at the moment

  Examples:
    - Driving in Los Angeles, NY, or Boston for that matter
    - Playing cards
    - Invest on stocks
    - Choose a university
  - Hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works
- Greedy algorithms tend to be easier to code

David Huffman's idea

- A Term paper at MIT

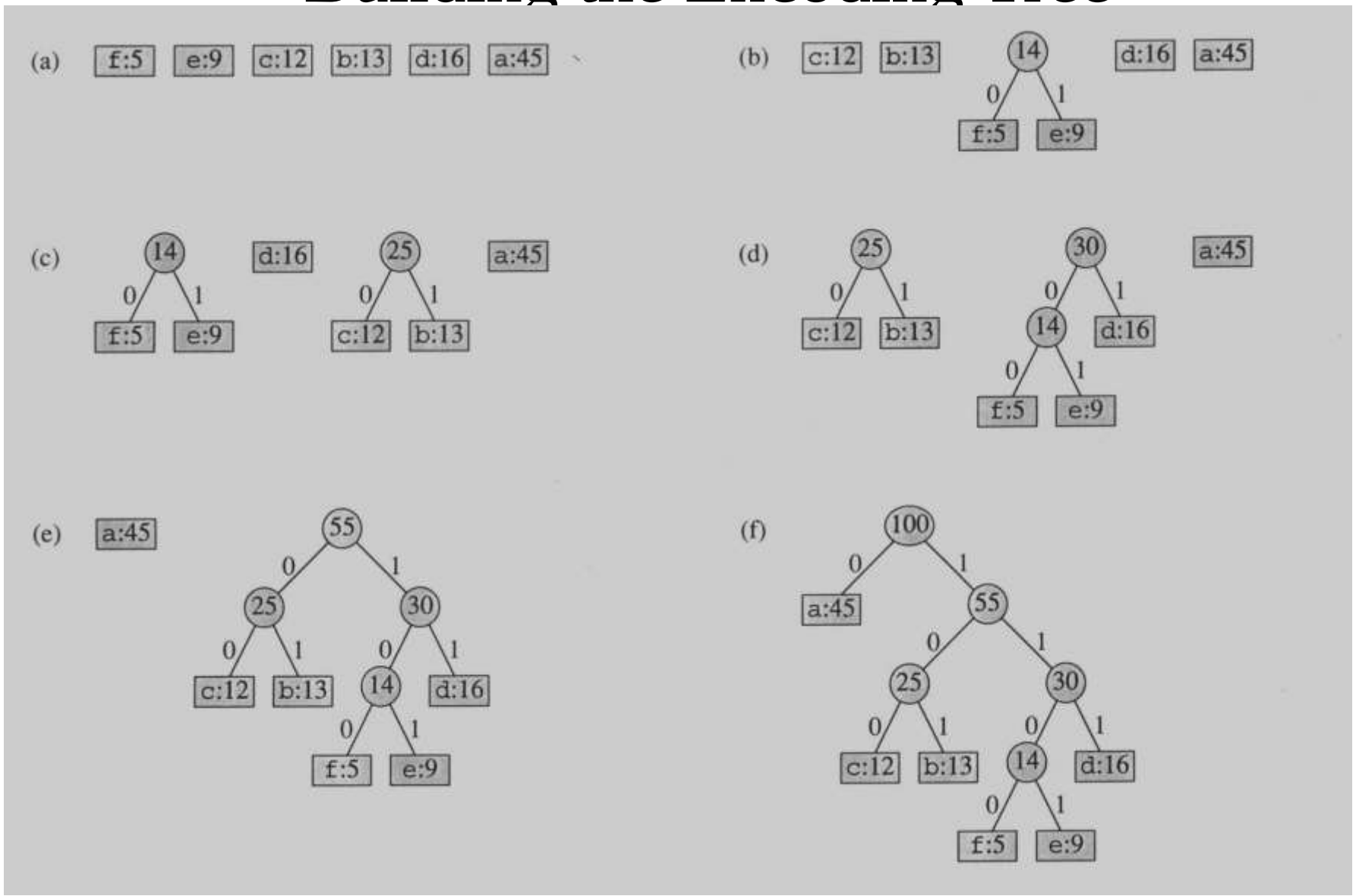
- Build the tree (code) bottom-up in a greedy fashion

- Origami aficionado

Huffman Algorithm

- **Step 1: Get Frequencies**
  - Scan the file to be compressed and count the occurrence of each character
  - Sort the characters based on their frequency
- **Step 2: Build Tree & Assign Codes**
  - Build a Huffman-code tree (binary tree)
  - Traverse the tree to assign codes
- **Step 3: Encode (Compress)**
  - Scan the file again and replace each character by its code
- **Step 4: Decode (Decompress)**
  - Huffman tree is the key to decompress the file

Building the Encoding Tree



All Characters are leaf nodes

Number at the root=Total no.of Characters

High Frequency-Near the root

Low Frequency-Far away from

The Algorithm

```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)
6          right[z] ← y ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)          ▷ Return the root of the tree.
```

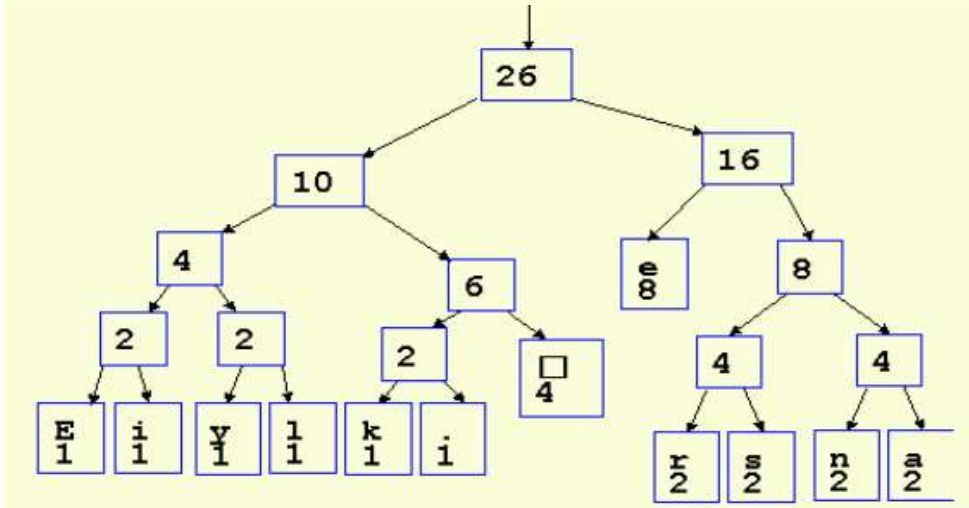
- An appropriate data structure is a binary min-heap
- Rebuilding the heap is *lg n* and *n-1* extractions are made, so the complexity is O(*n lg n*)

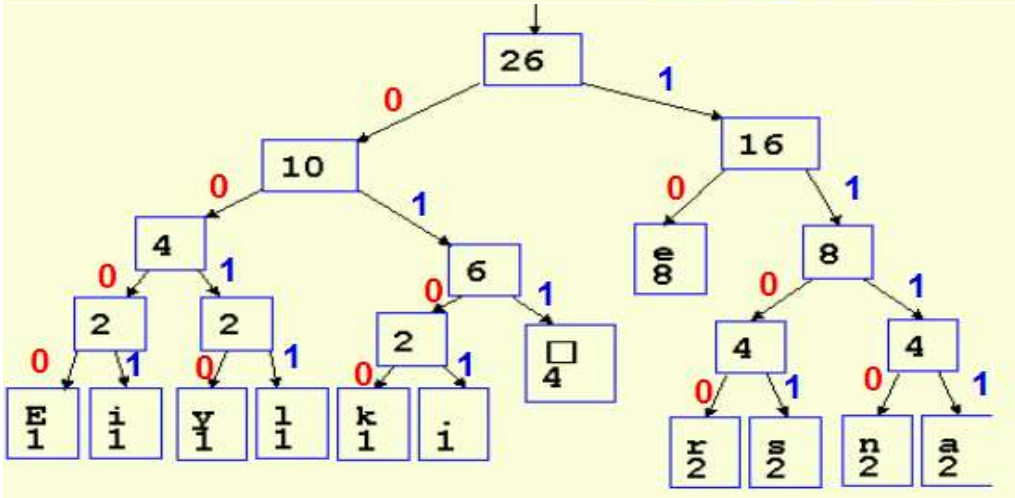
Get Frequencies

Build the Tree

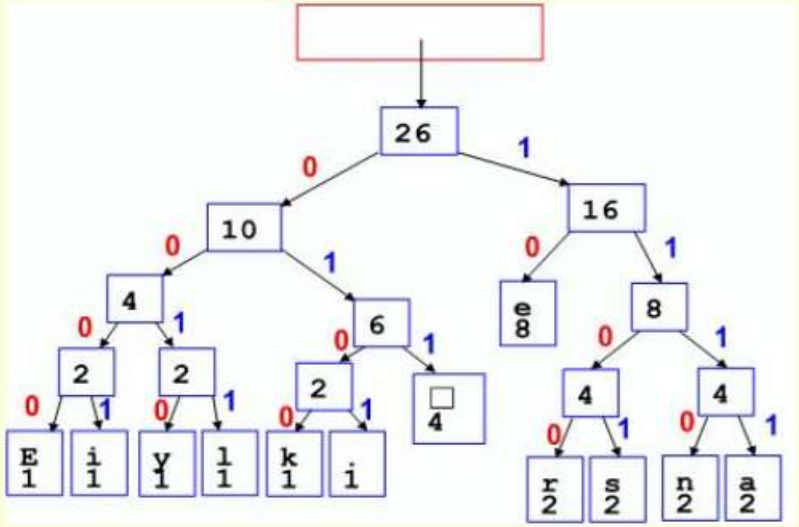
Assign Codes

Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2		
i	1	a	2		
space	4	l	1		





- Traverse the Tree



Coding Table

Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
space□	011
e	10
r	1100
s	1101
n	1110
a	1111

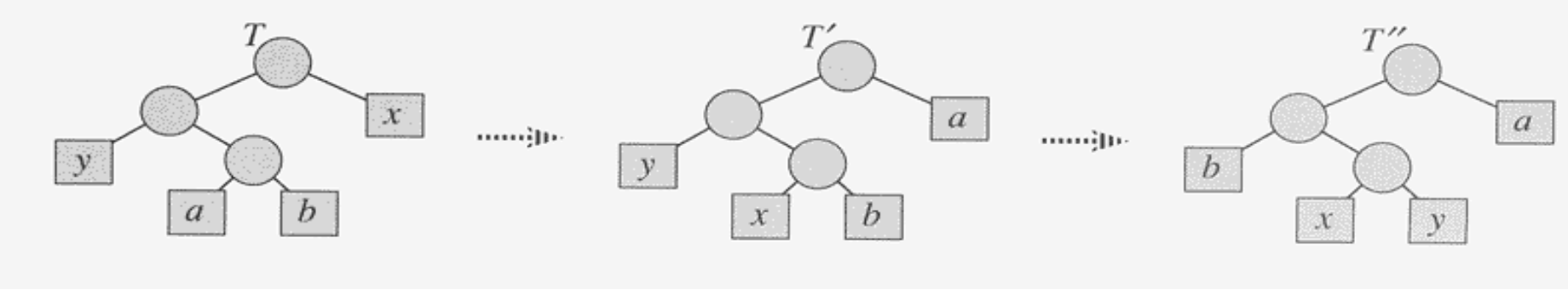
- **Traverse the tree**
  - **Any left edge □ add label 0**
  - **As right edge □ add label 1**
- **The code for each character is its root-to-leaf label sequence**

- Encode :EyE ---0000 0010 0000-
- Decode:0000→E:0010->y:0000->E

Correctness of Huffman's Algorithm

Lemma 16.2

Let C be an alphabet in which each character $c \in C$ 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.



Since each swap does not increase the cost, the resulting tree T" is also an optimal tree