Lecture # 13

- **Greedy Algorithms**
  - **Huffman Codes**

{An Application of **Binary Trees** and **Priority Queues**}

Purpose of Huffman Coding

- **Proposed by Dr. David A. Huffman in 1952**
  - ***"A Method for the Construction of Minimum Redundancy Codes"***
- **Applicable to many forms of data transmission**
  - **Our example: text files**

The Basic Algorithm

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency!
- Yet all characters are allocated the same amount of space
  - 1 char = 1 byte, be it **e** or **x**

The Basic Algorithm

- Any savings in tailoring codes to frequency of character?
- Code word lengths are no longer fixed like ASCII.
- Code word lengths vary and will be shorter for the more frequently used characters.

The (Real) Basic Algorithm

1. Scan text to be compressed and tally occurrence of all characters.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

Building a Tree

Scan the original text

- Consider the following short text:

  Eerie eyes seen near lake.

- Count up the occurrences of all characters in the text

Building a Tree

Scan the original text

Eerie eyes seen near lake.

- **What characters are present?**

E e r i space

y s n a r l k .

Building a Tree

Scan the original text

Eerie eyes seen near lake.

- What is the frequency of each character in the text?

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		

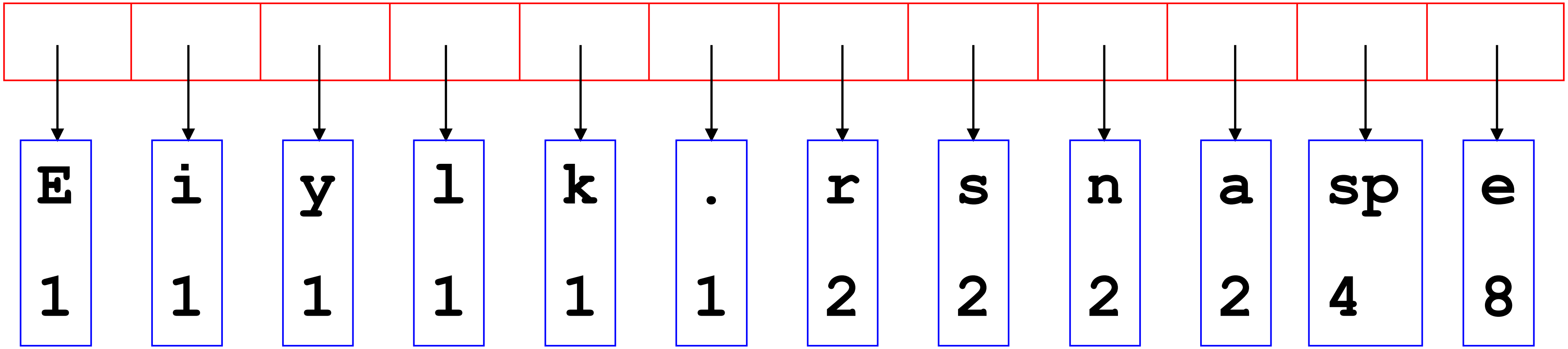
Building a Tree

Prioritize characters

- Create binary tree nodes with character and frequency of each character
- Place nodes in a priority queue
  - The lower the occurrence, the higher the priority in the queue

Building a Tree

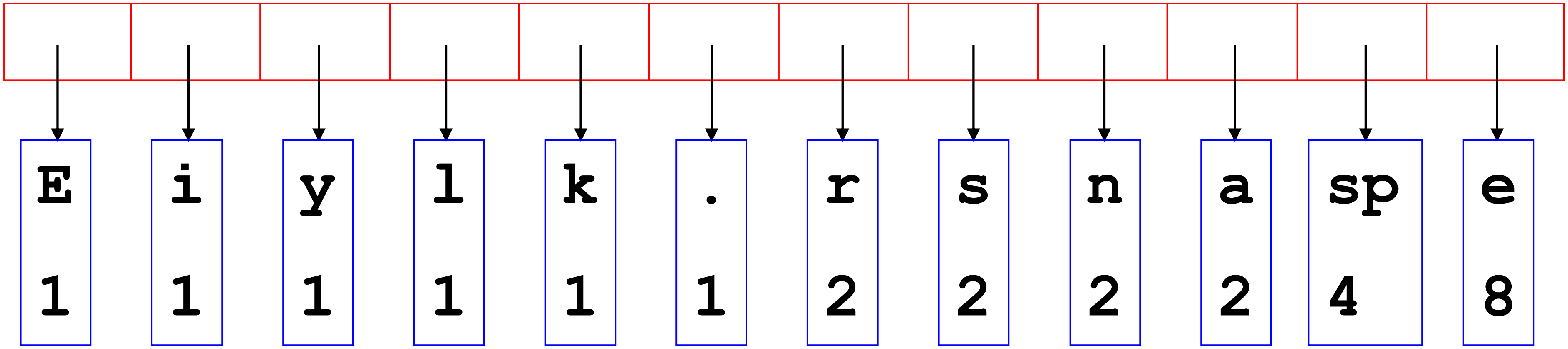
- The queue after inserting all nodes



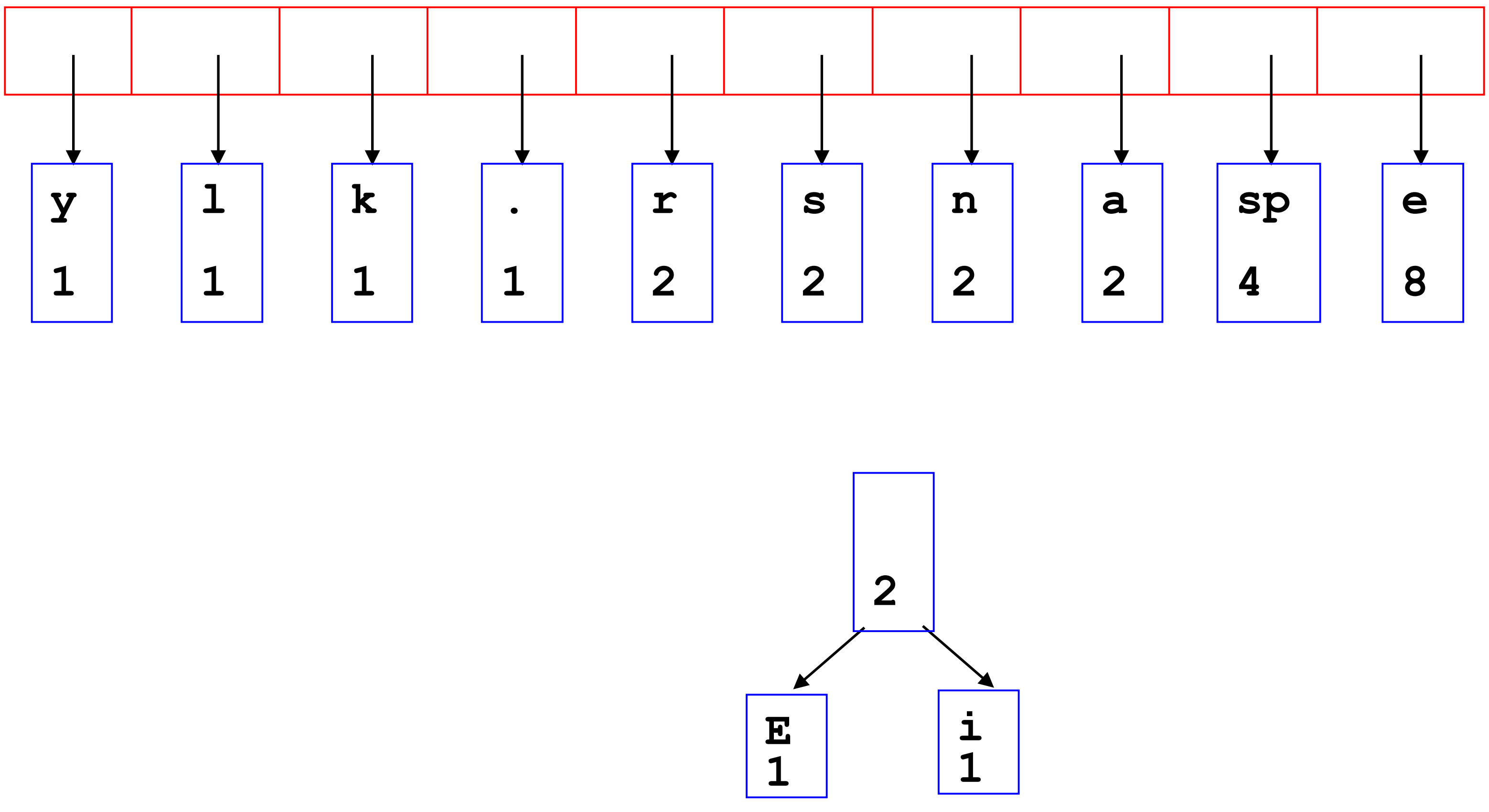
Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue

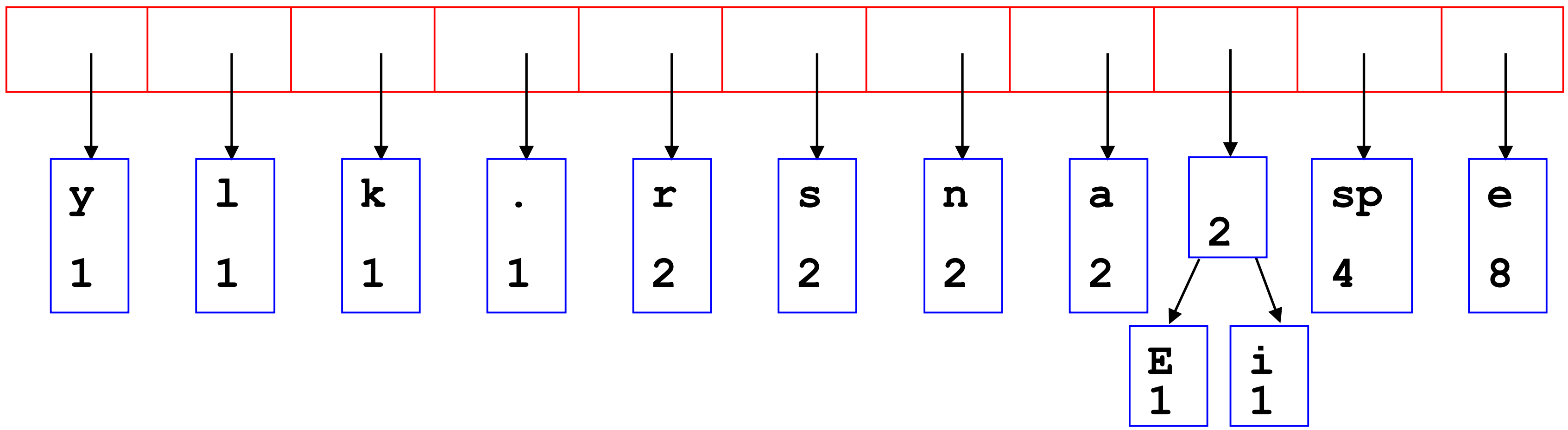
Building a Tree



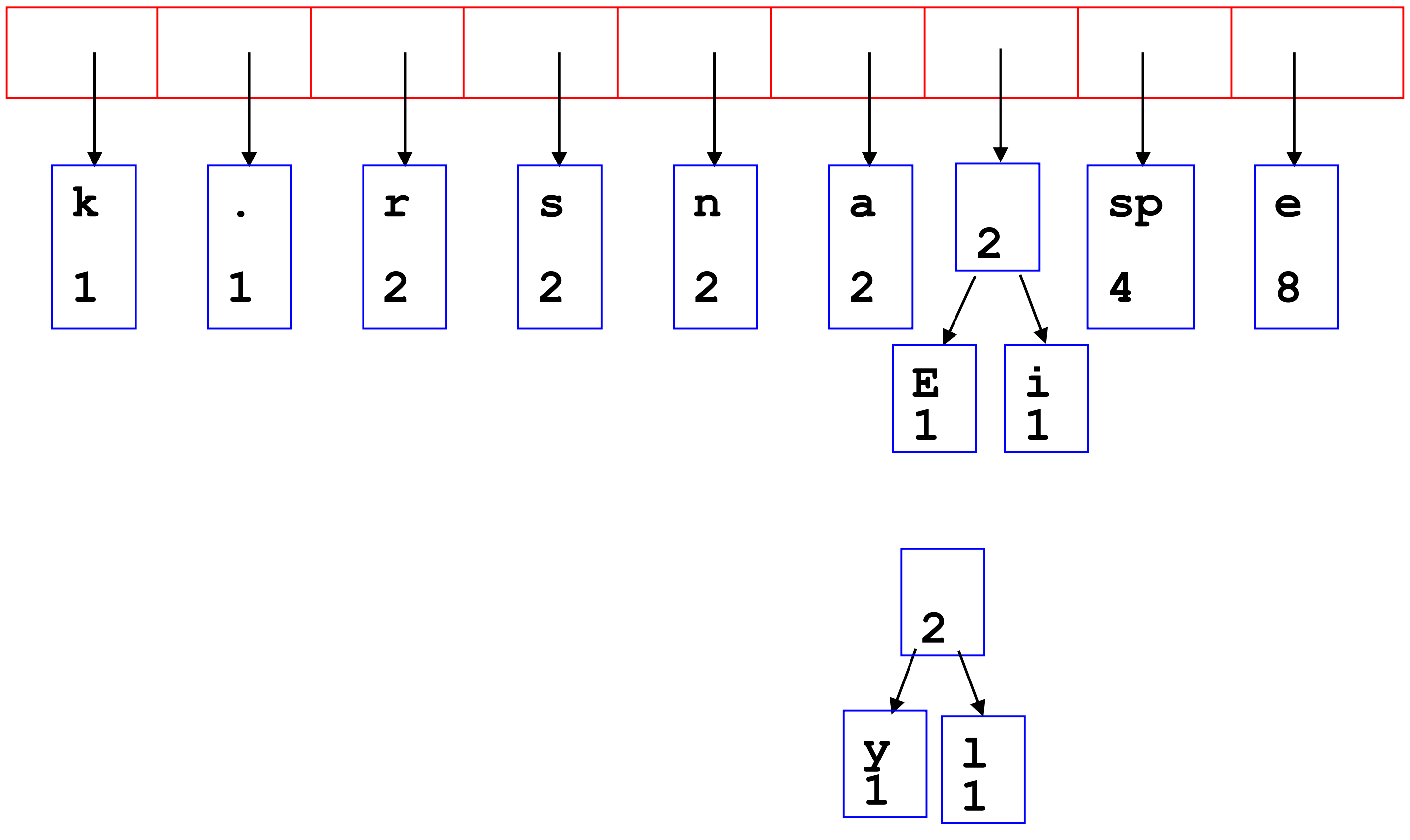
Building a Tree



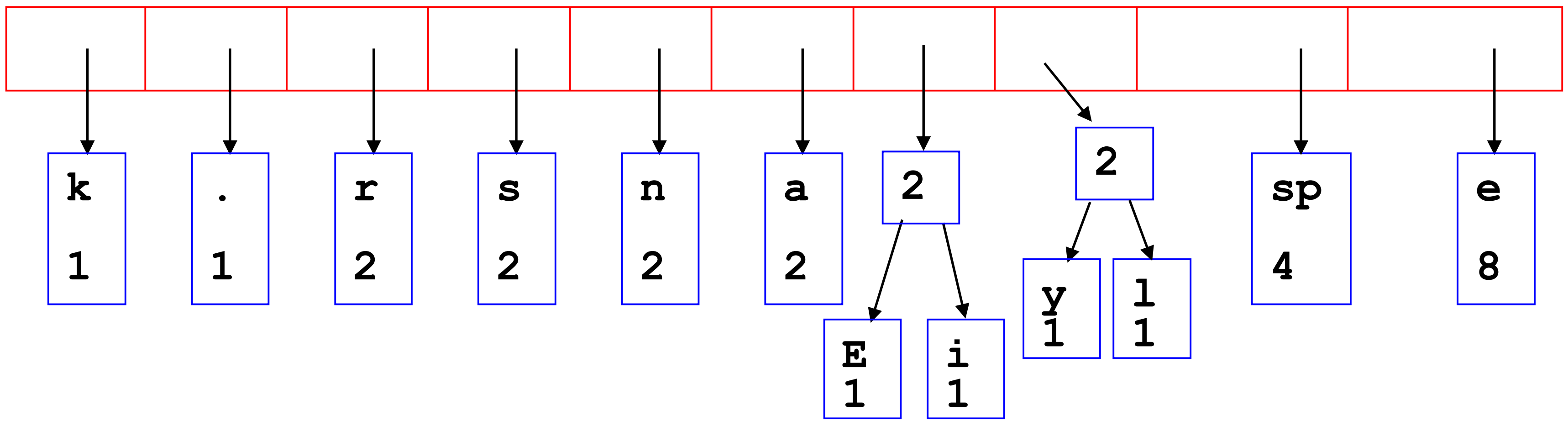
Building a Tree



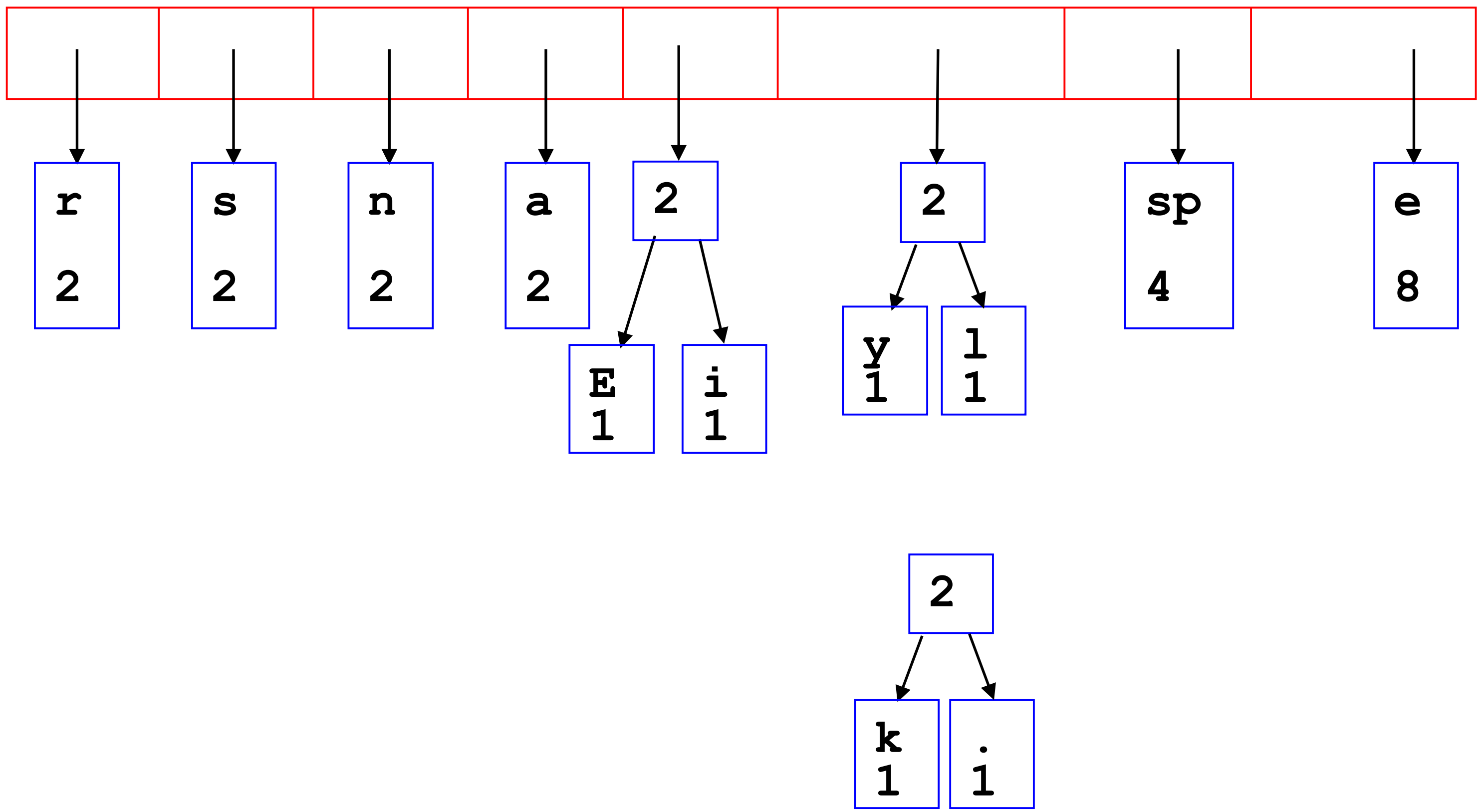
Building a Tree



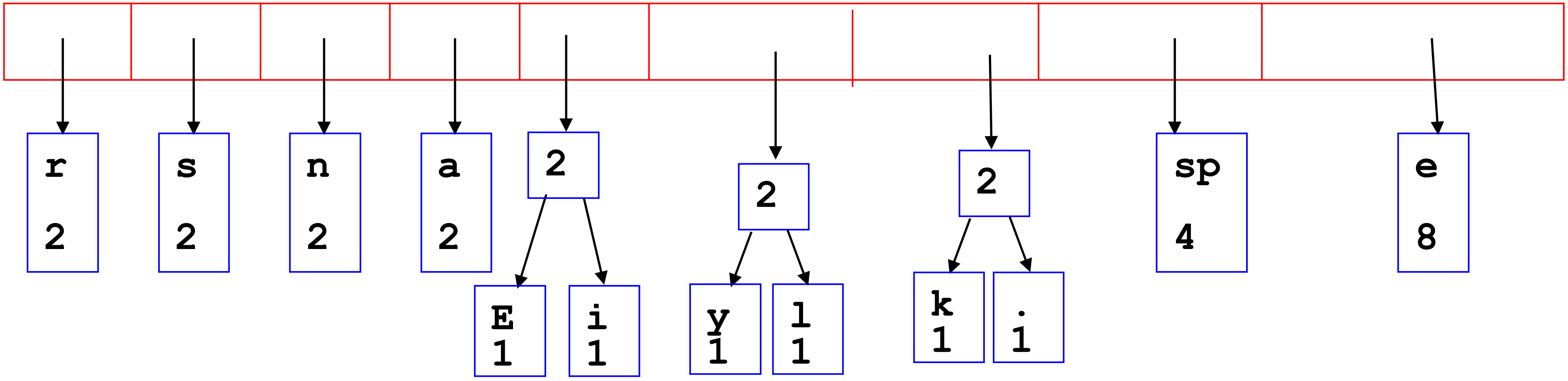
Building a Tree



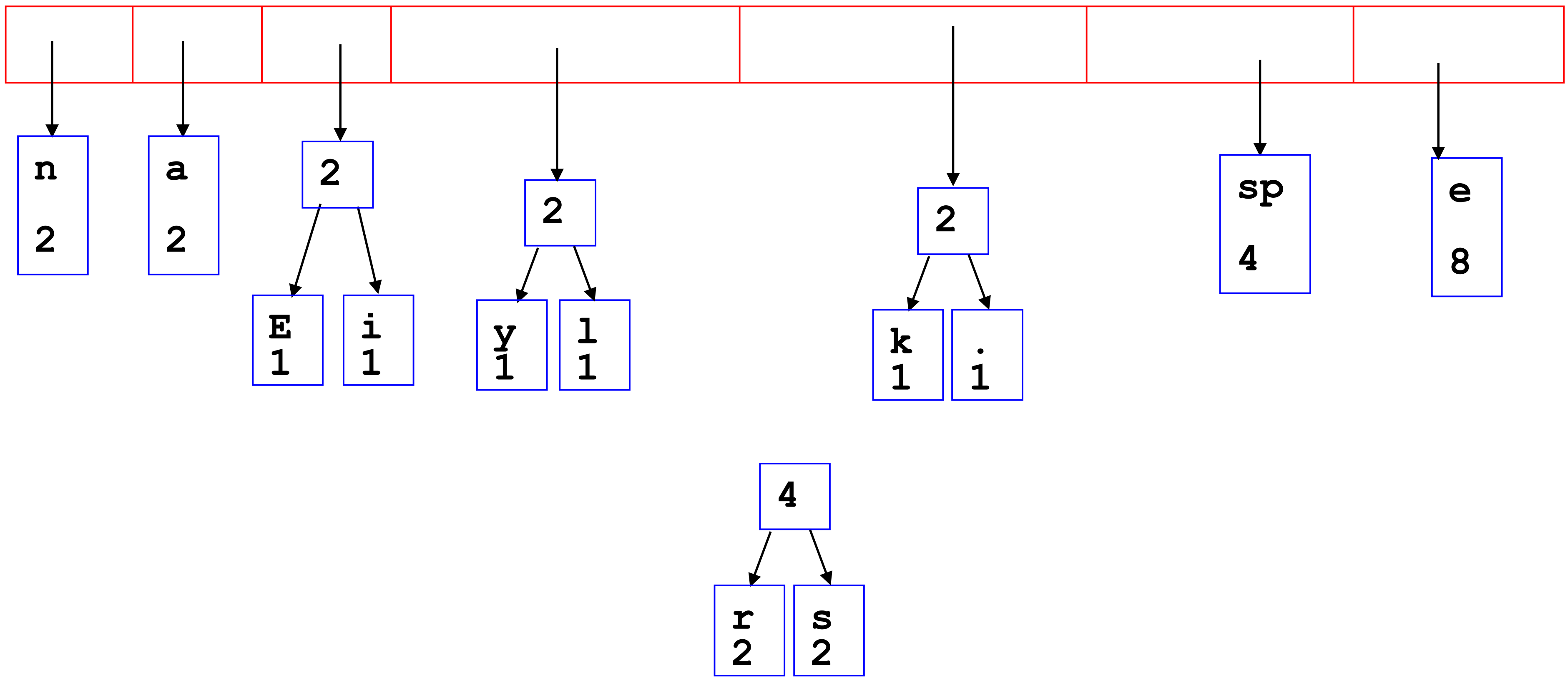
Building a Tree



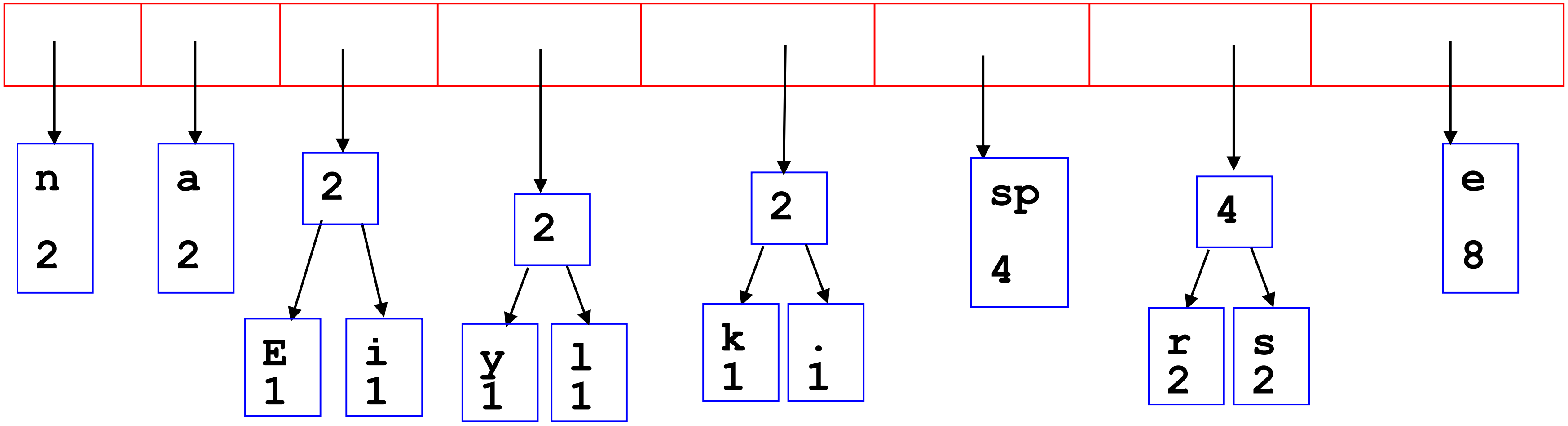
Building a Tree



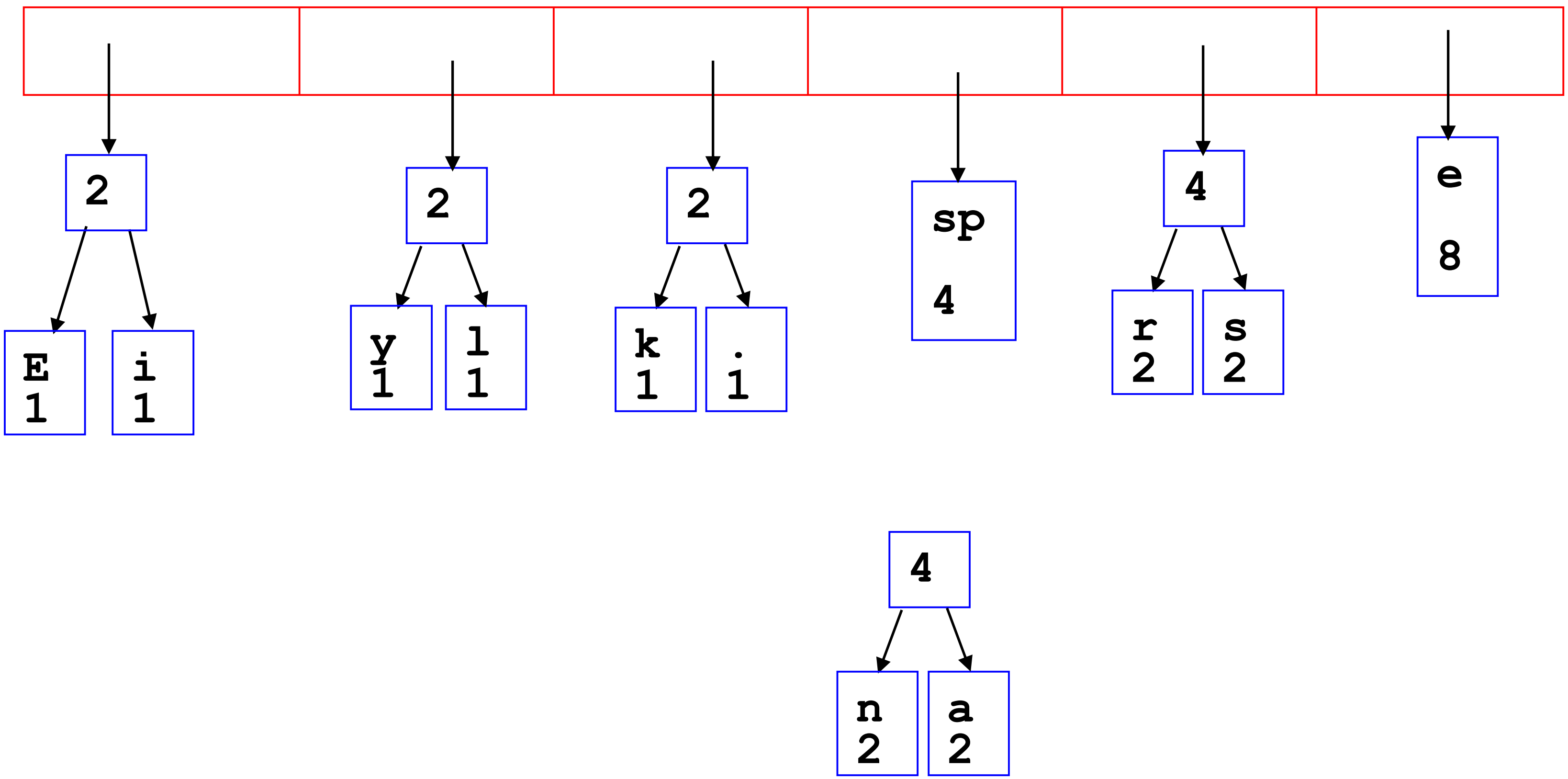
Building a Tree



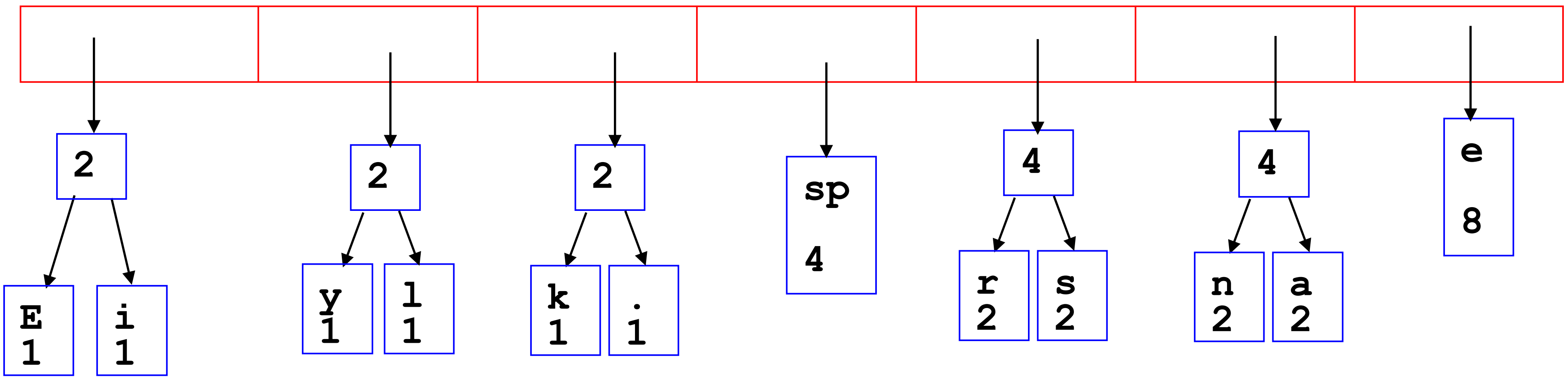
Building a Tree



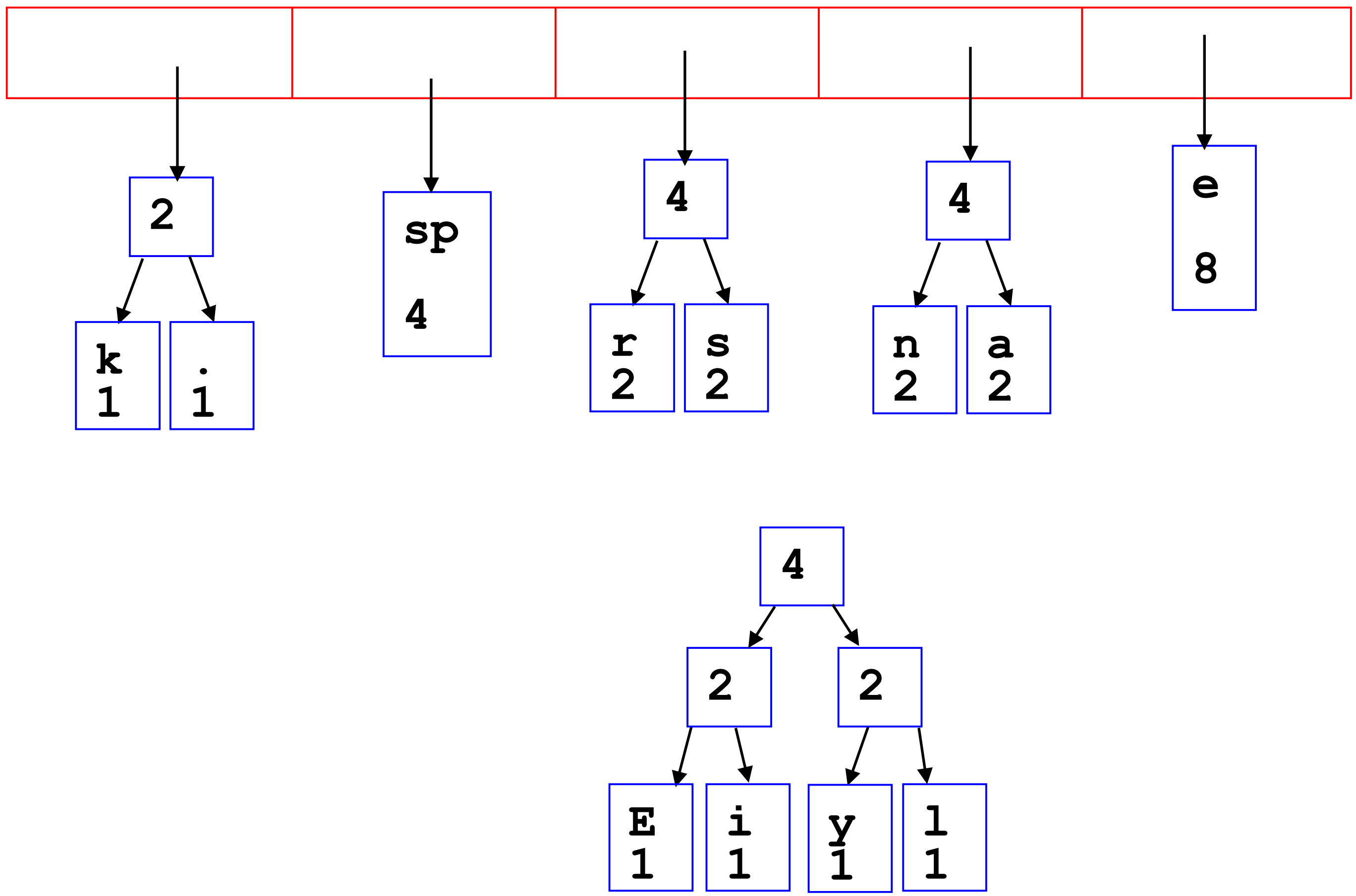
Building a Tree



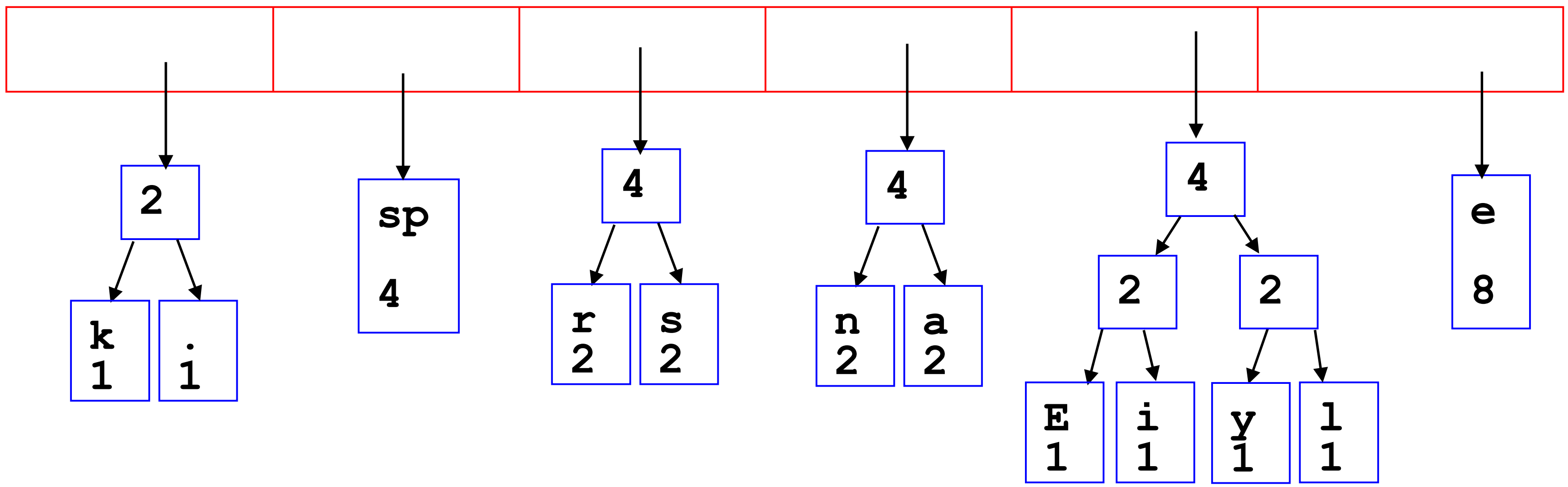
Building a Tree



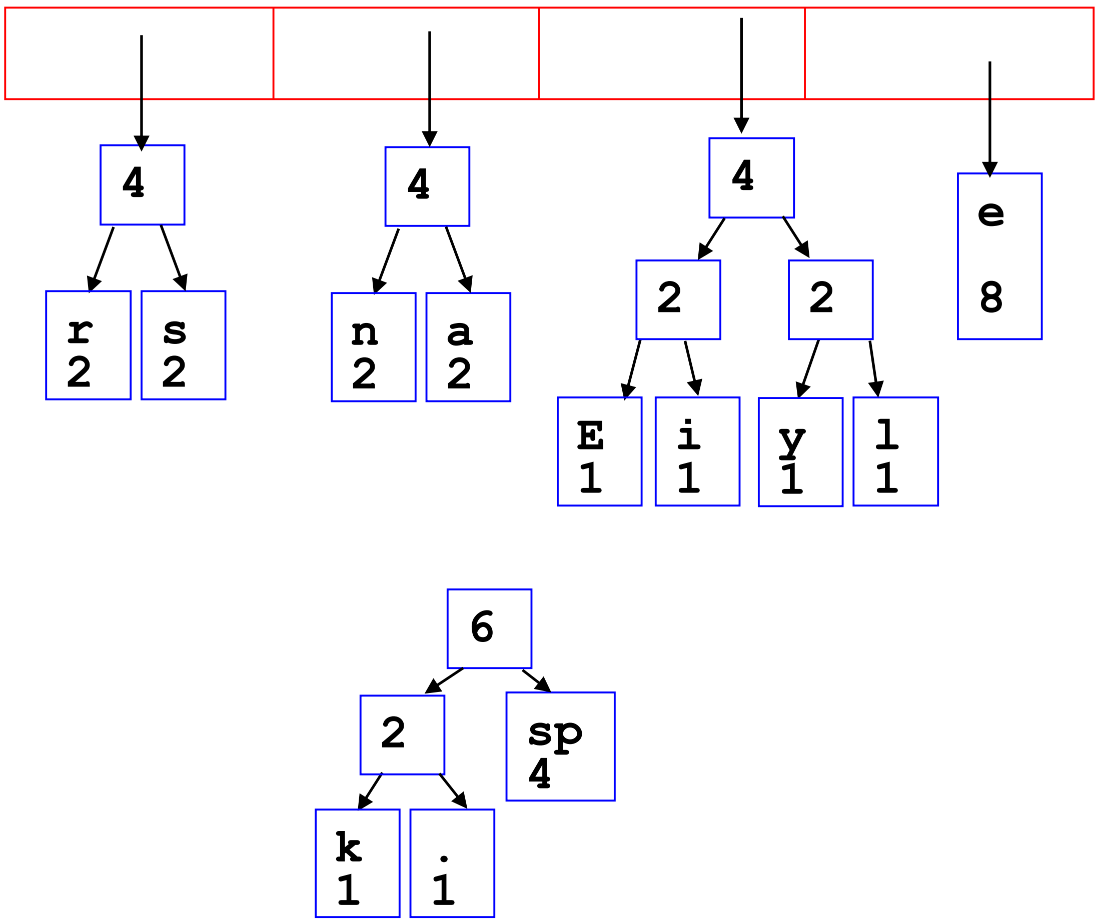
Building a Tree



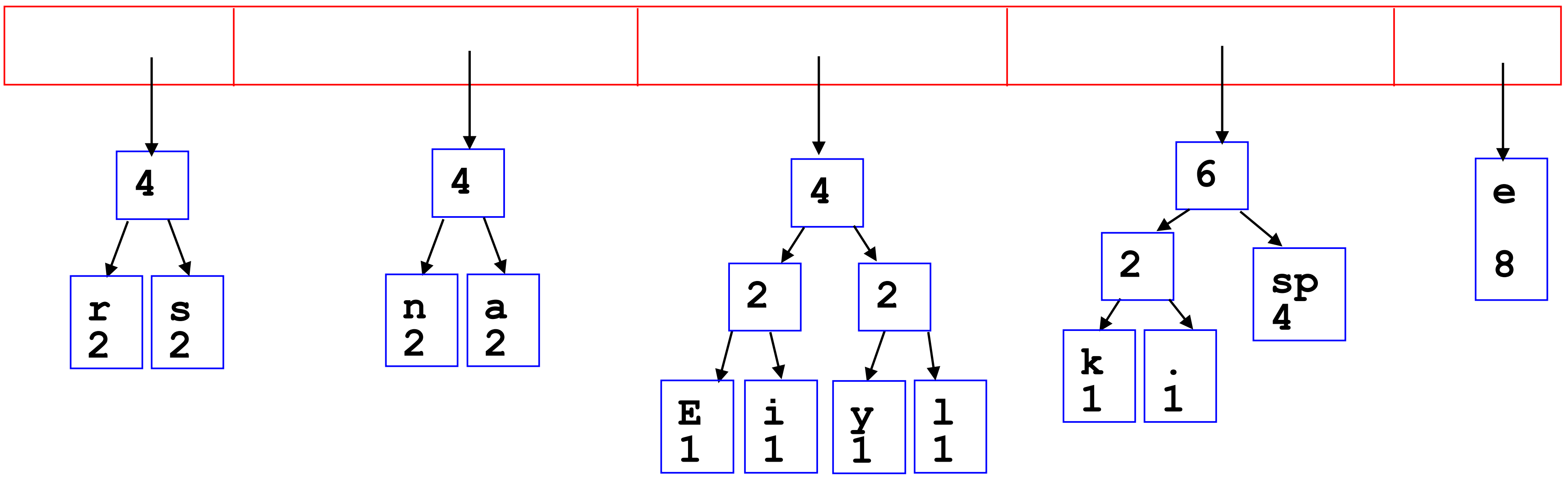
Building a Tree



Building a Tree

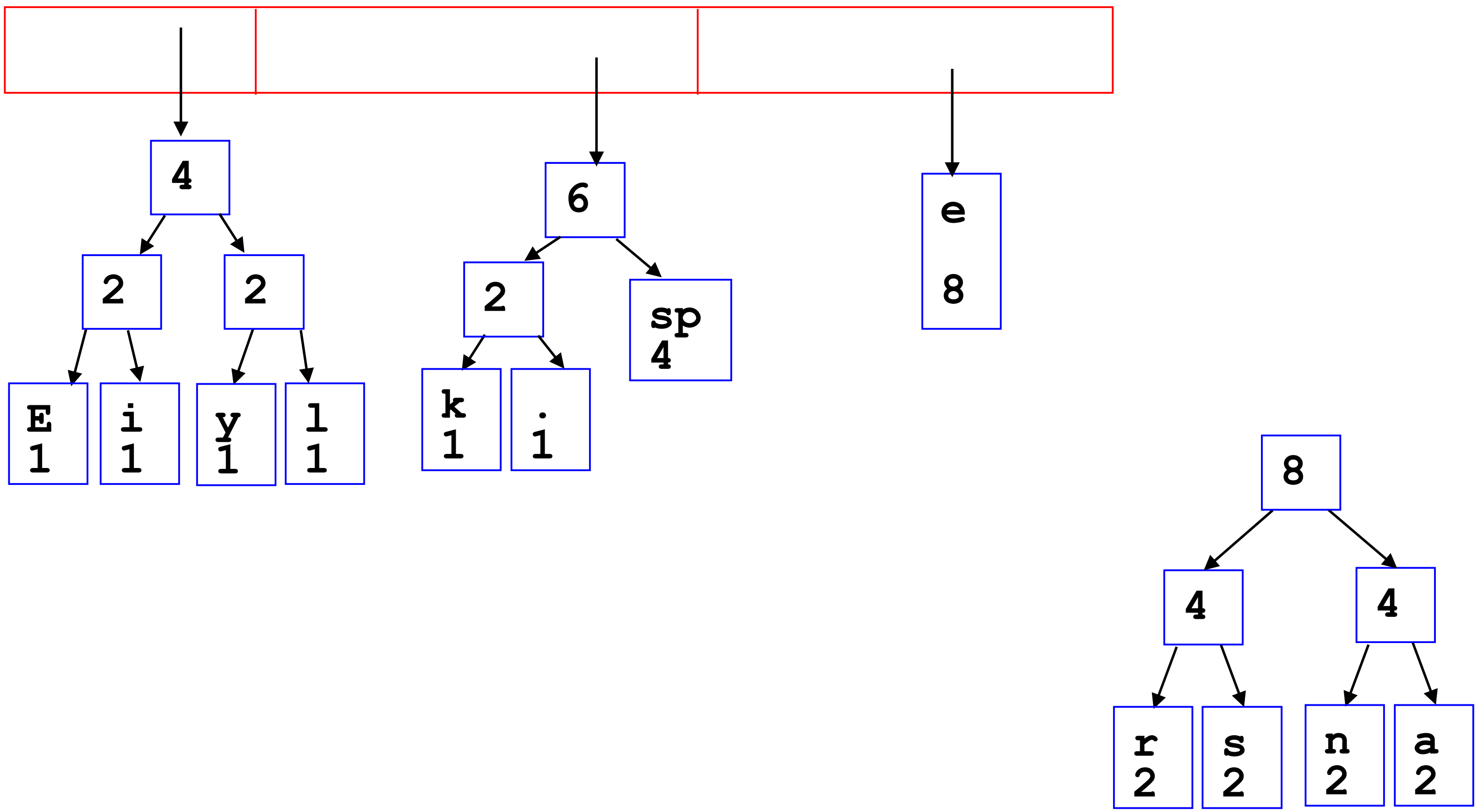


Building a Tree

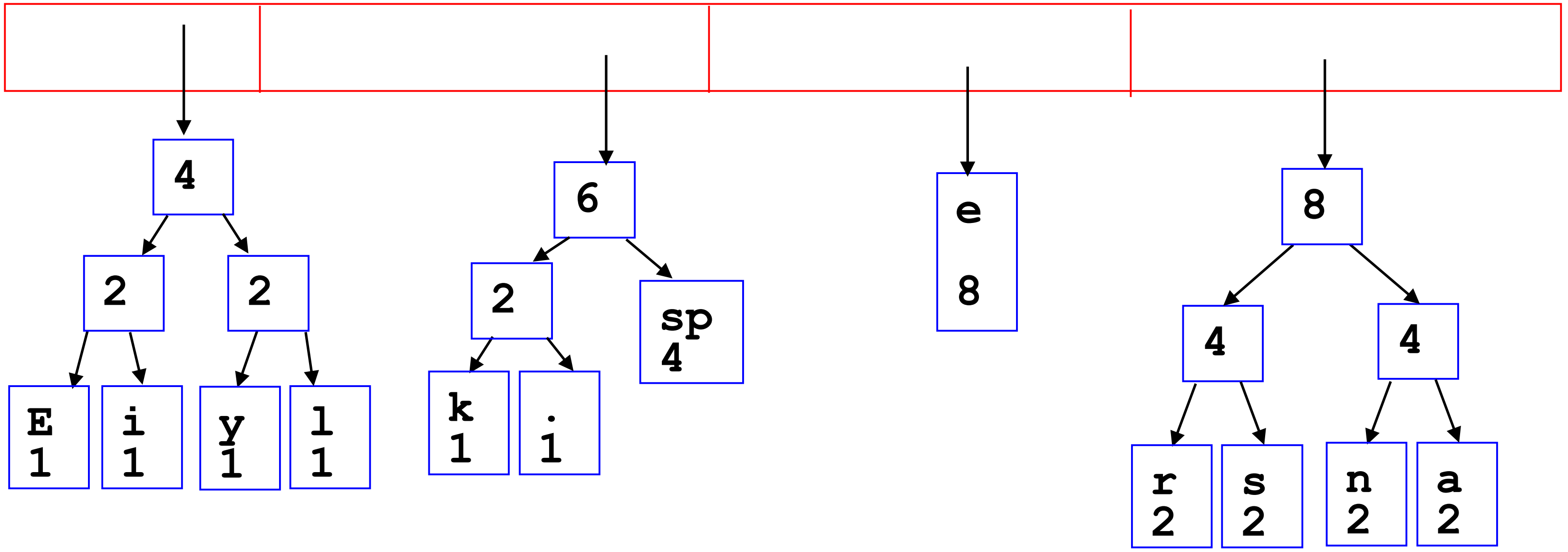


What is happening to the characters with a low number of occurrences?

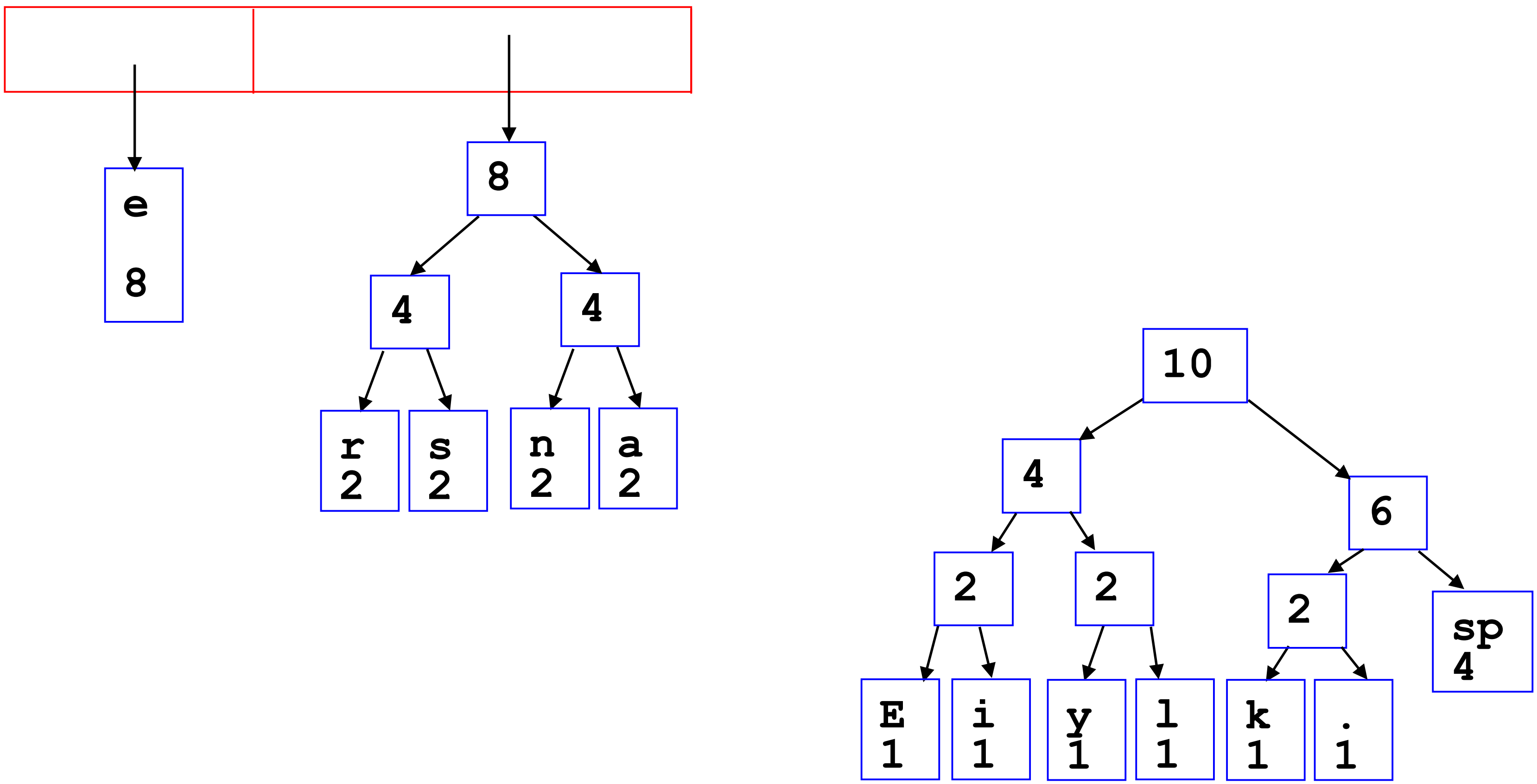
Building a Tree



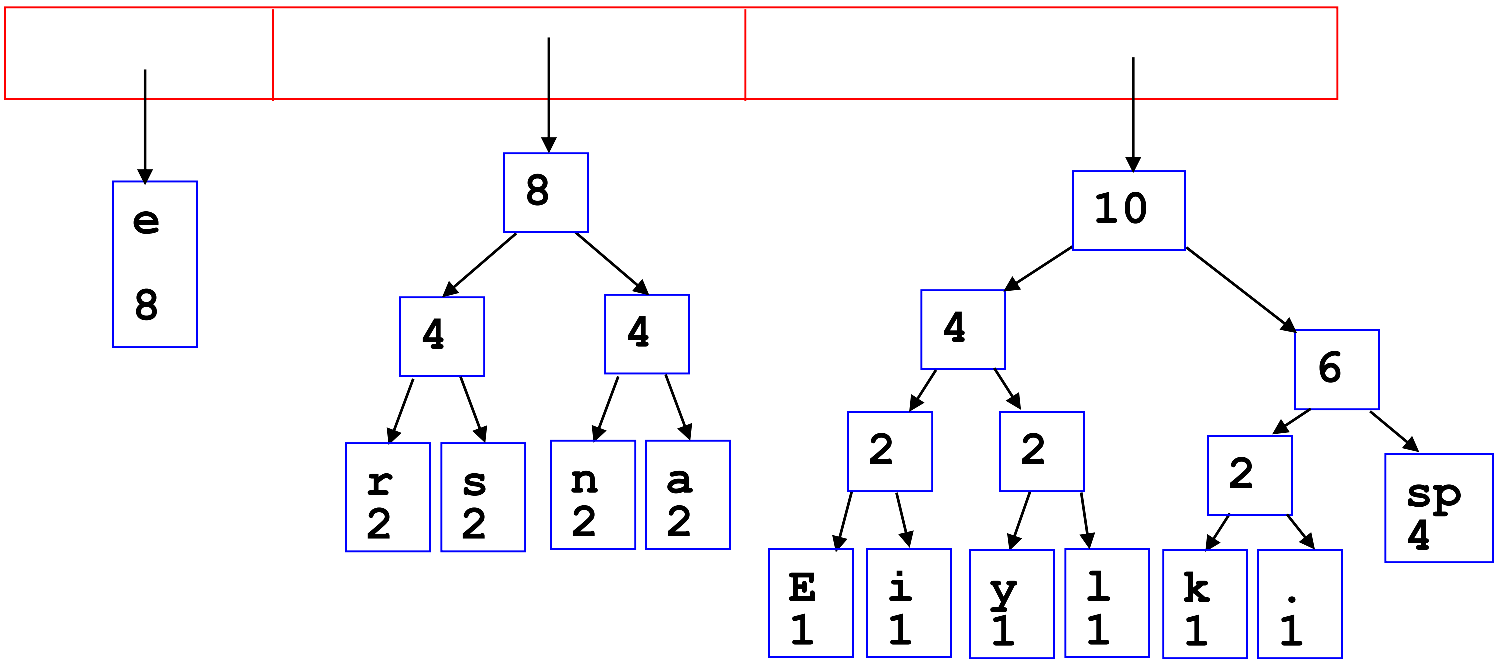
Building a Tree



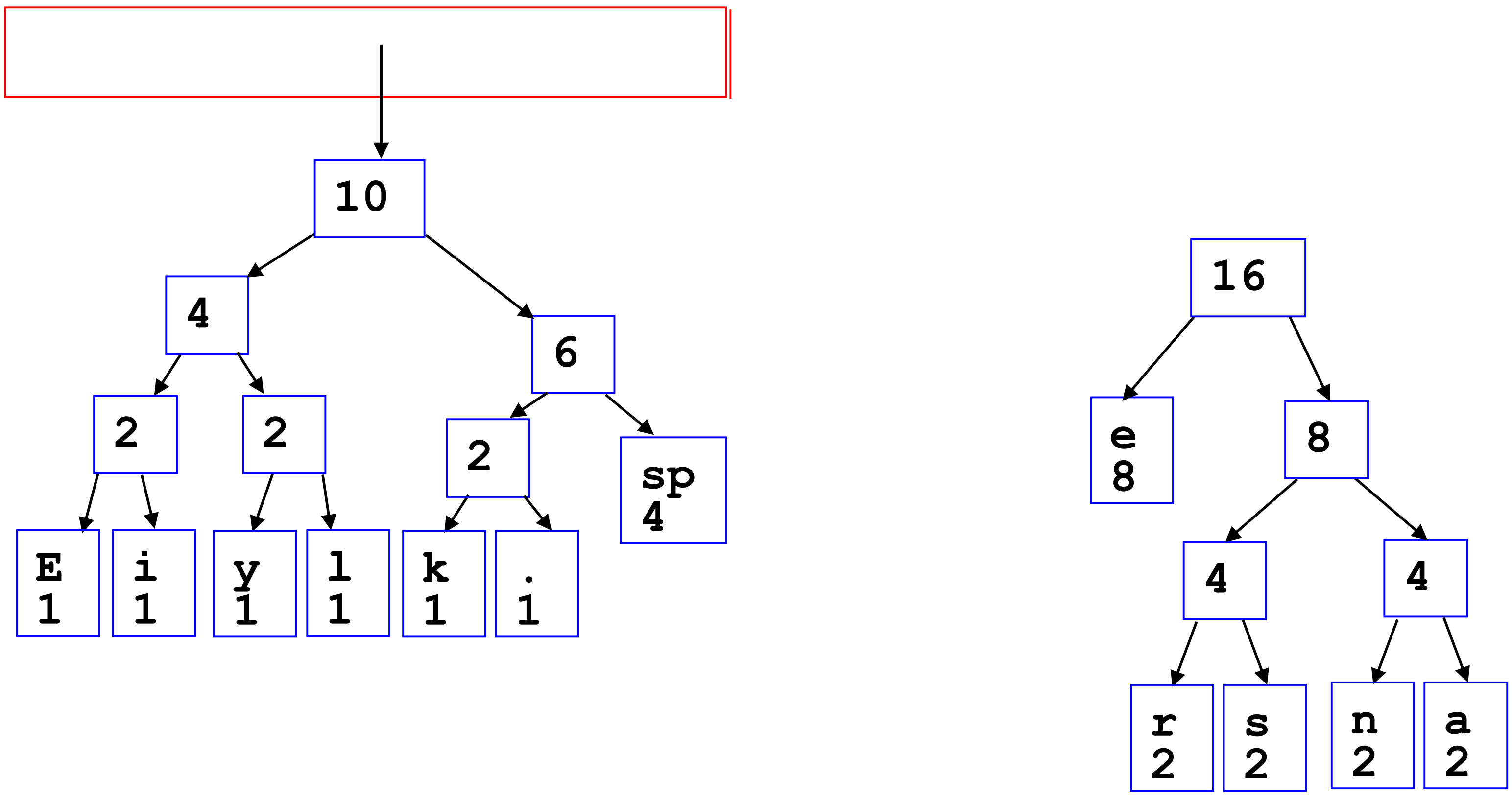
Building a Tree



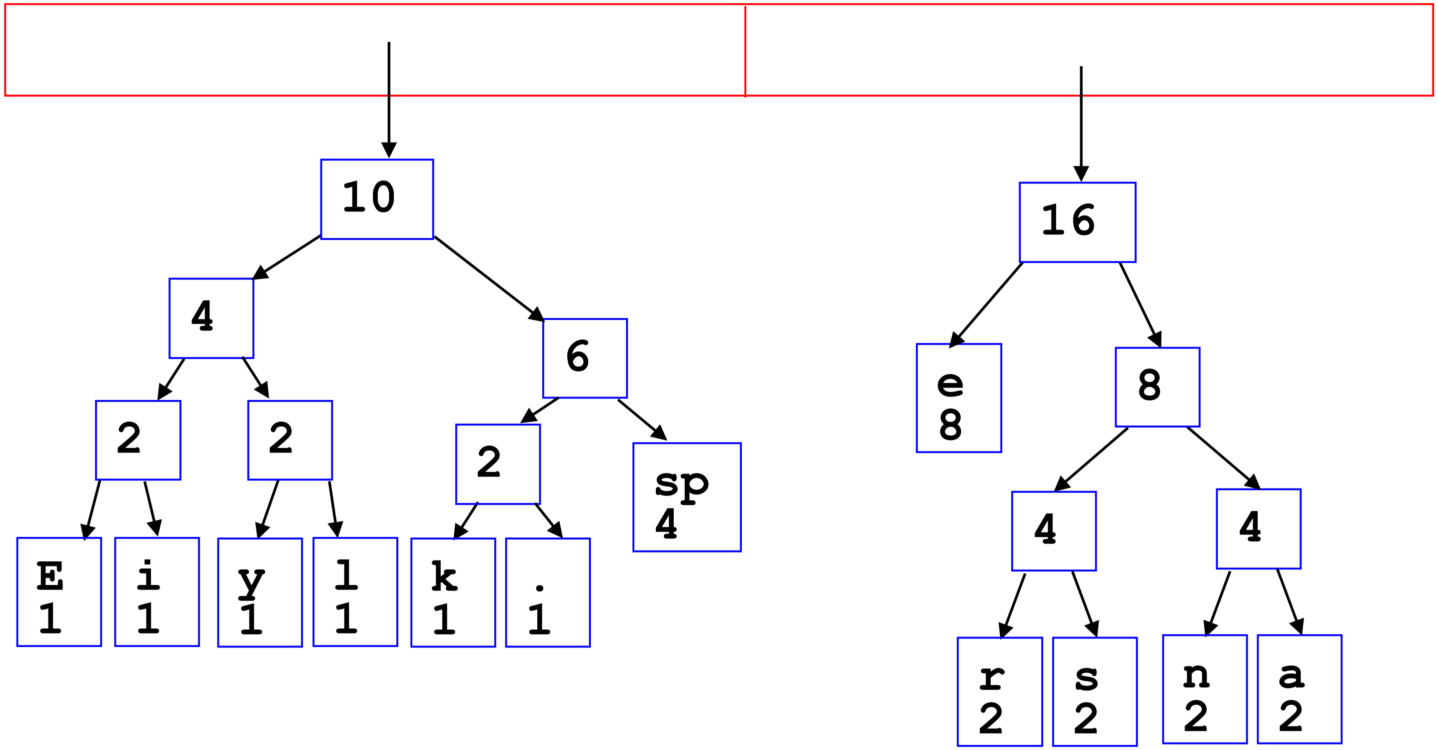
Building a Tree



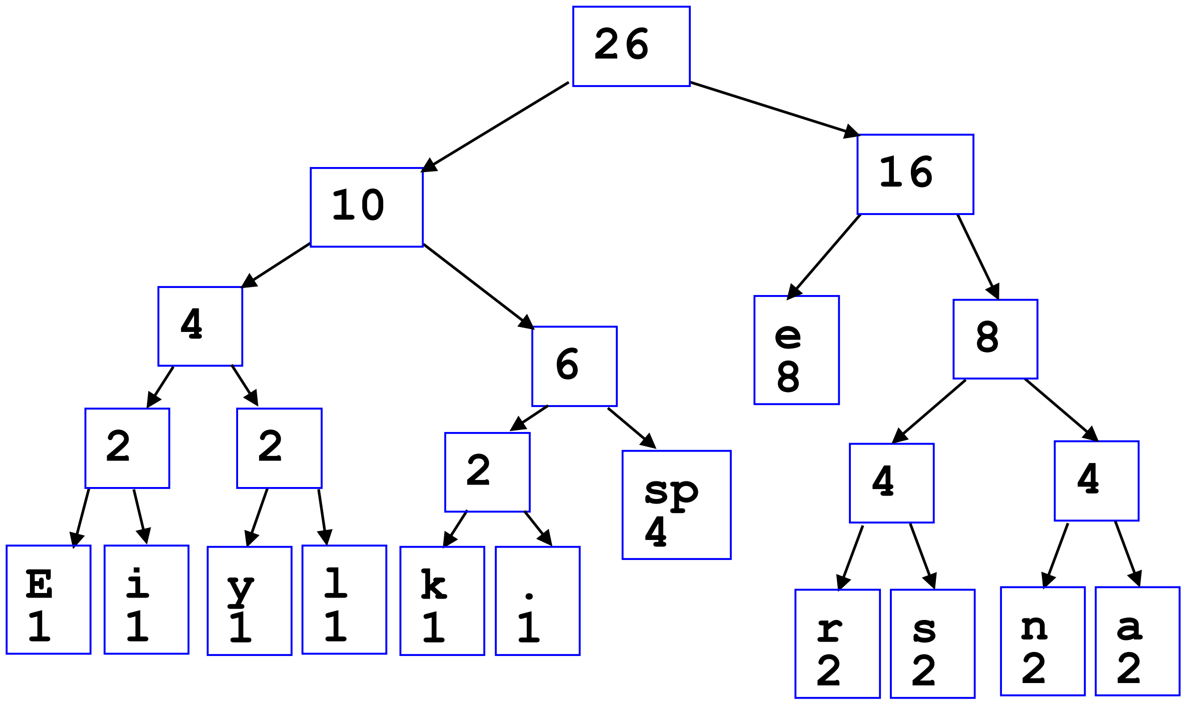
Building a Tree



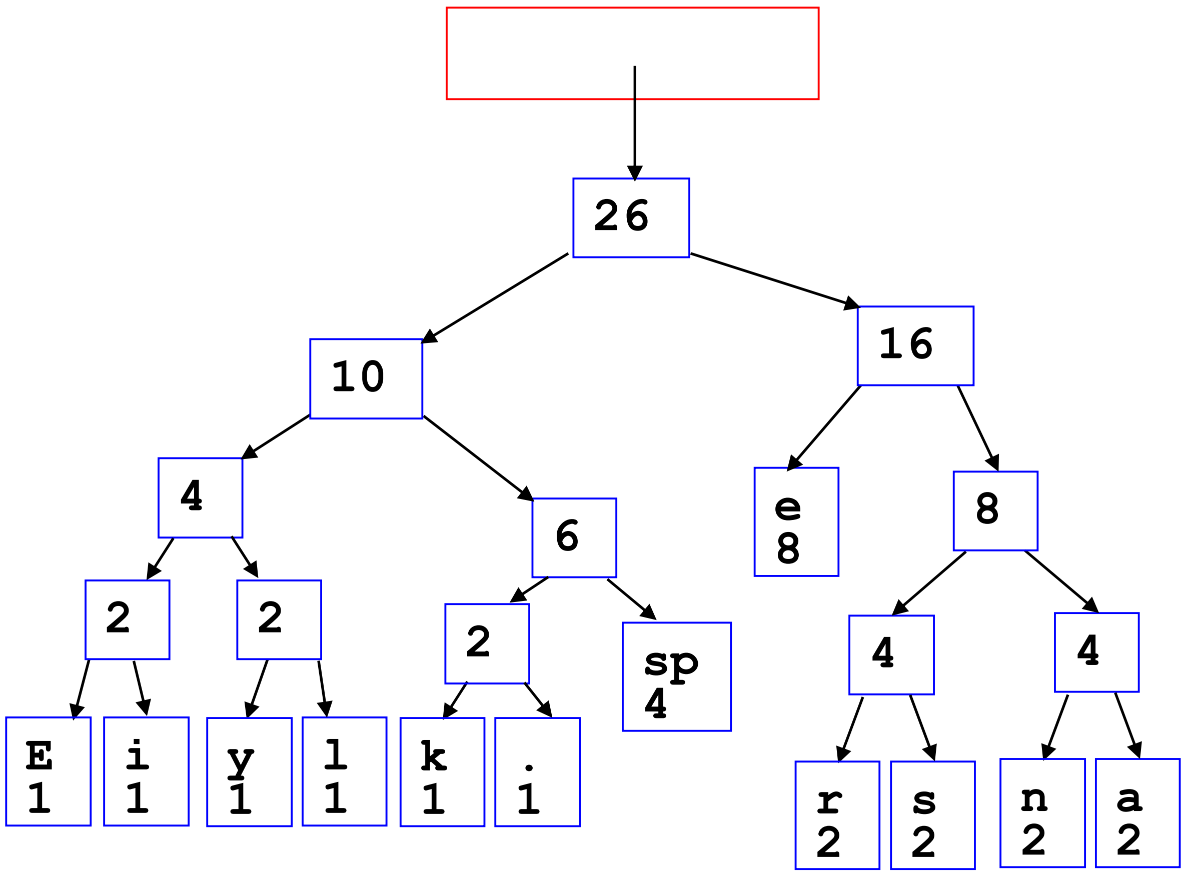
Building a Tree



Building a Tree



Building a Tree



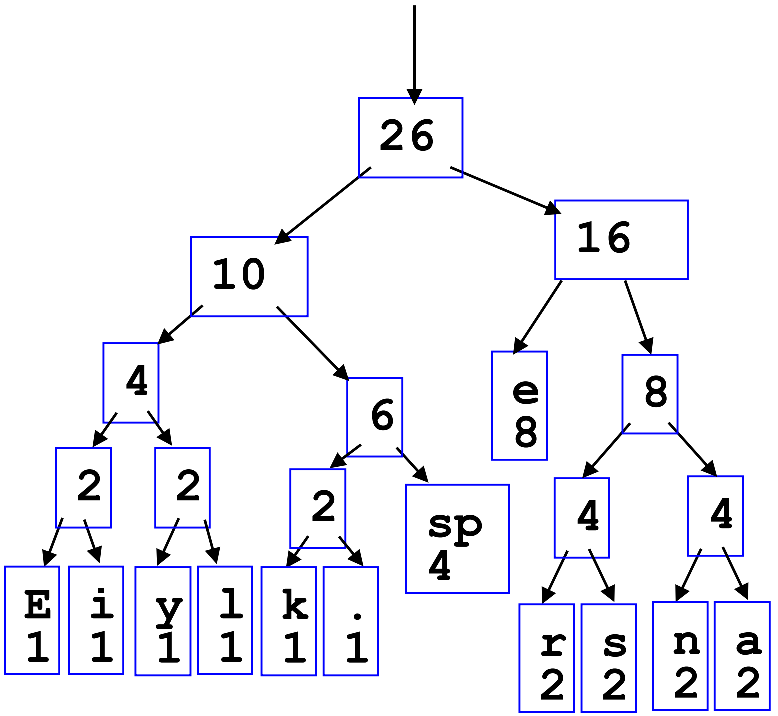
- After enqueueing this node there is only one node left in priority queue.

Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

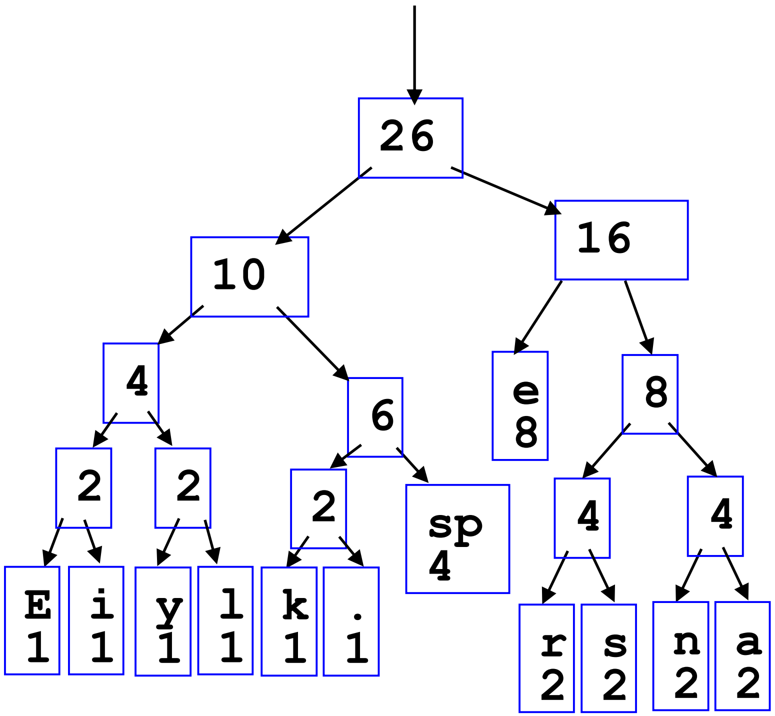
Frequency of root node should equal number of characters in text.



Eerie eyes seen near lake. 26 characters

Encoding the File
Traverse Tree for Codes

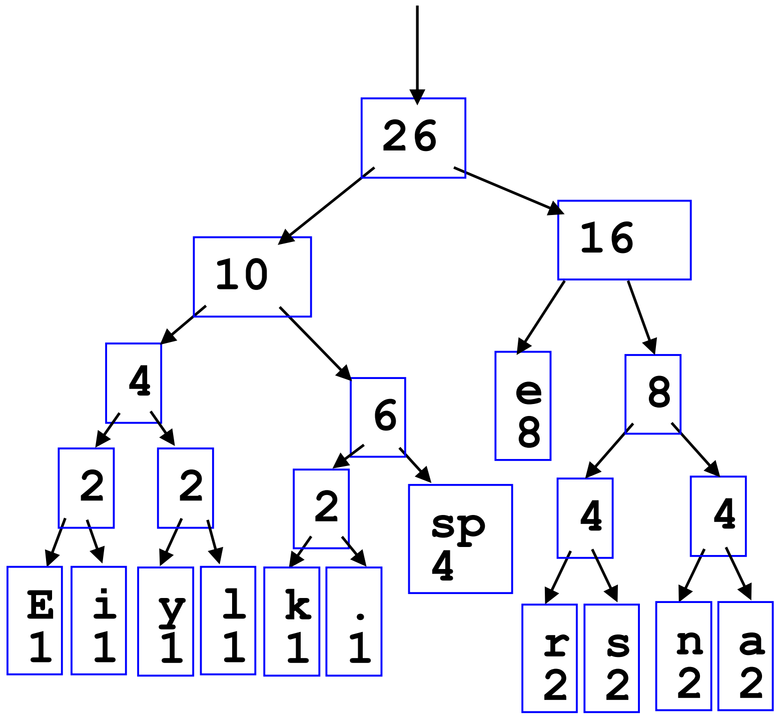
- Perform a traversal of the tree to obtain new code words
- Going left is a 0 going right is a 1
- code word is only completed when a leaf node is reached



Encoding the File

Traverse Tree for Codes

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



Encoding the File

- Rescan text and encode file using new code words

Eerie eyes seen near lake.

```
0000101100000110011
1000101011011010011
1110101111110001100
1111110100100101
```

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

Encoding the File
Results

- Have we made things any better?
- 73 bits to encode the text

```
0000101100000110011
1000101011011010011
1110101111110001100
111111010010010 1
```

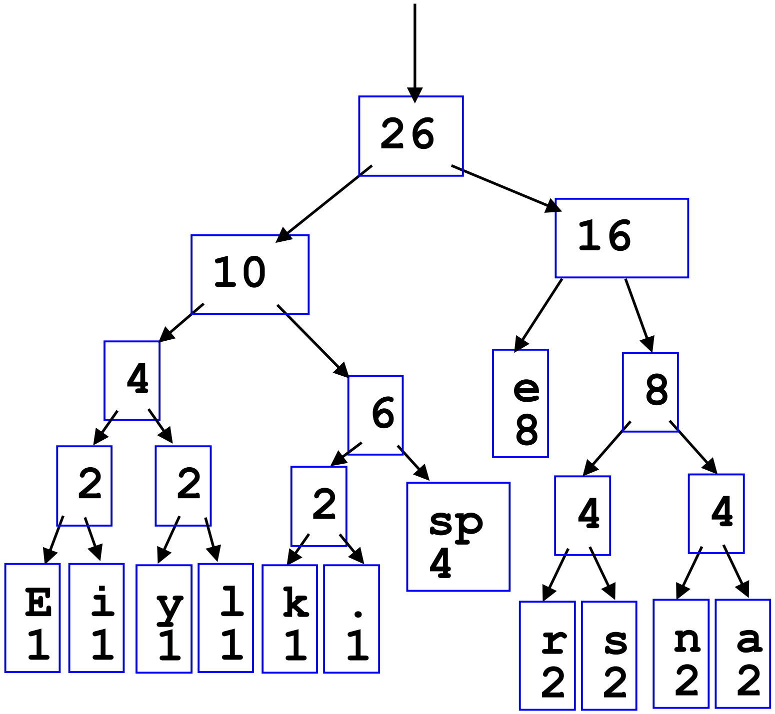
- ASCII would take 8 * 26 = 208 bits
- If modified code used 4 bits per character are needed. Total bits 4 * 26 = 104. Savings not as great.

Decoding the File

- How does receiver know what the codes are?
- Tree constructed for each text file.
  - Considers frequency for each file
  - Big hit on compression, especially for smaller files
- Data transmission is bit based versus byte based

Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 ⇒ go left
- 1 ⇒ go right

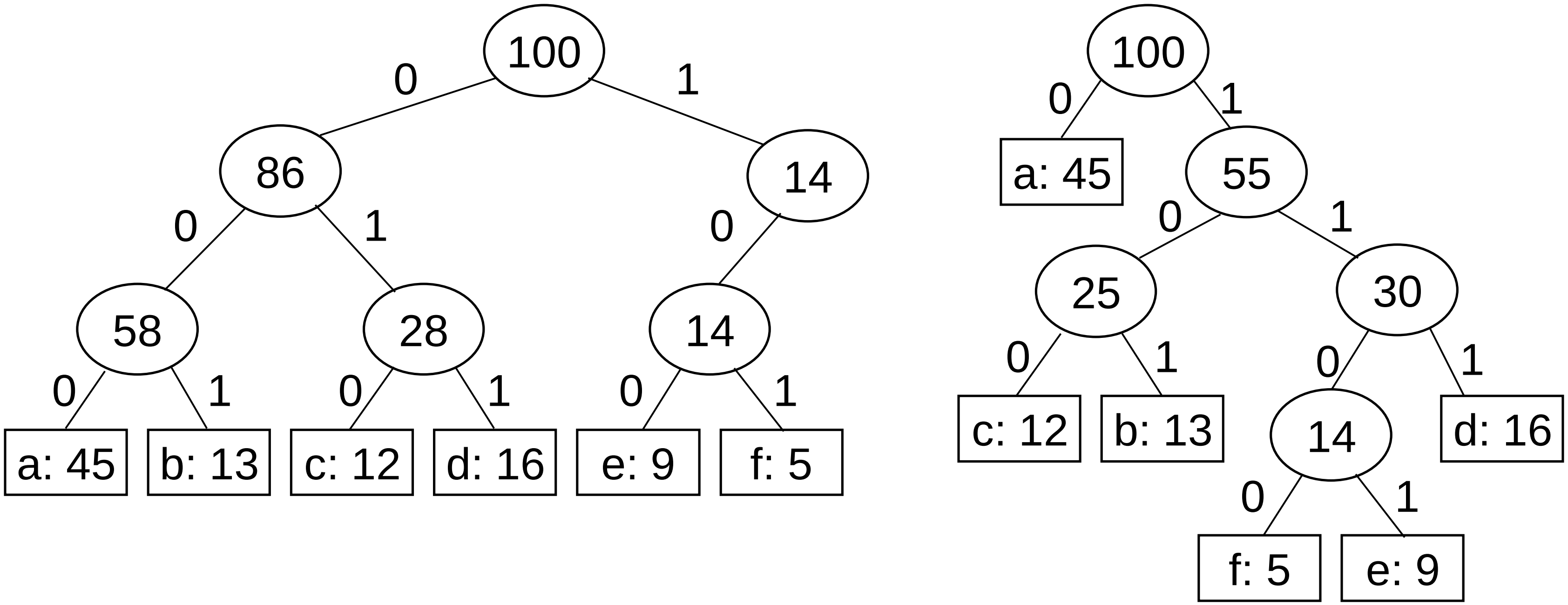



```
1010001101111101111
01111110000110101
```

Decoding with Binary Character Codes

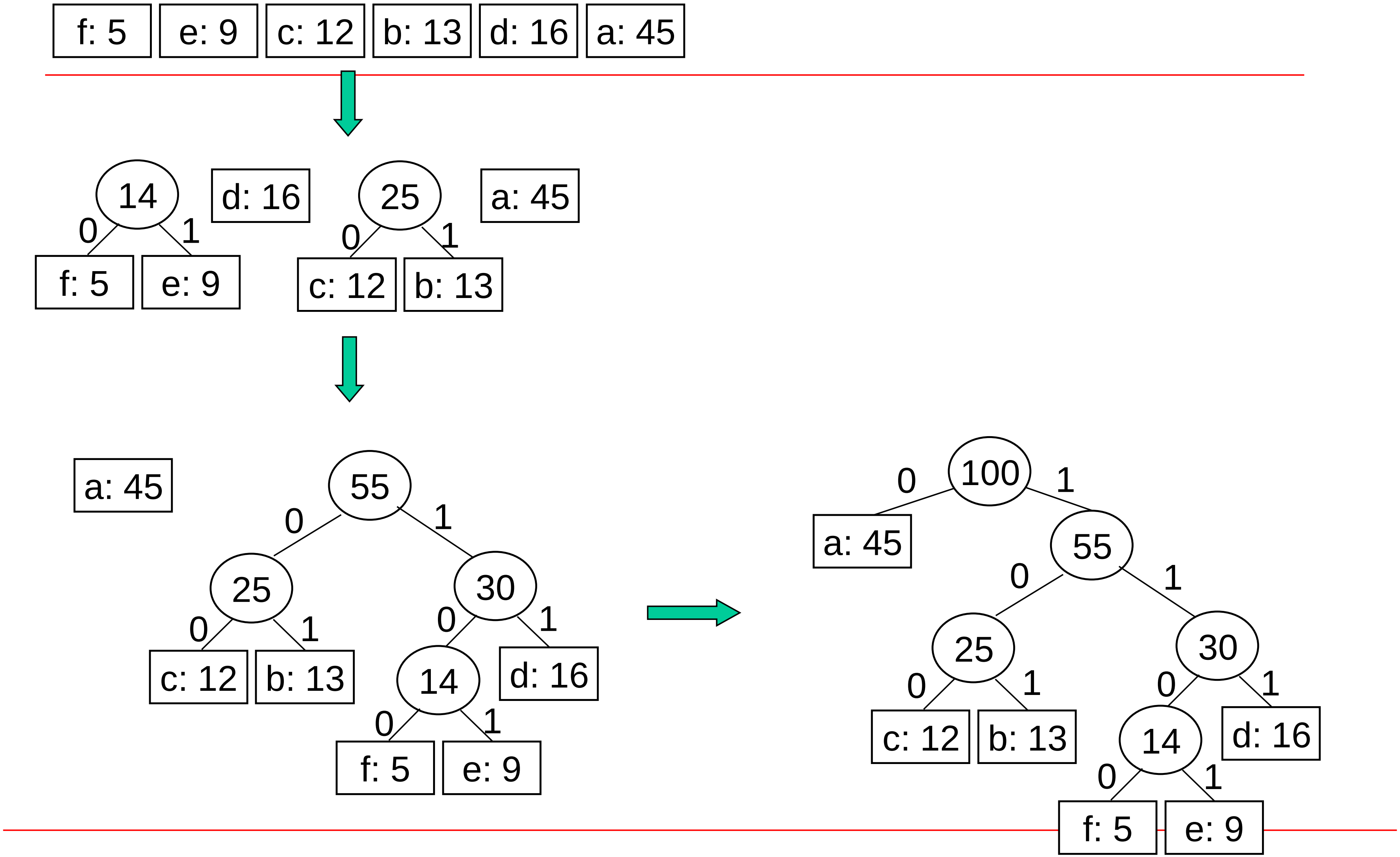
- **Prefix codes simplify decoding**
  - **No codeword is a prefix of another**
- *E.g.*:
  - **a = 0, b = 101, c = 100, d = 111, e = 1101, f = 1100**
  - **001011101 =** $\longrightarrow$ $0 \cdot 0 \cdot 101 \cdot 1101 = \text{aabe}$

Optimal Codes



- **An optimal code is always represented by a** full binary tree
  - **Every non-leaf has two children**
  - **Fixed-length code is not optimal, variable-length is**

Example



Analysis

- **Build-Heap can be performed in** $O(n)$.
- **Since each Heap operation takes** $O(\lg n)$ **time.**
- **So as a whole the total running time of huffman codes on a set of n unique characters is** $O(n.\lg n)$