Huffman Coding:
An Application of Binary Trees and Priority Queues

Encoding and Compression of Data

- Fax Machines
- ASCII
- Variations on ASCII
  - min number of bits needed
  - cost of savings
  - patterns
  - modifications

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 (ASCII)
  - 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.

Suppose that we have a 100,000 character data file that we wish to store . The file contains only 6 characters, appearing with the following frequencies:

	a	b	c	d	e	f
Freq	45000	13000	12000	16000	9000	5000
A fixed lenght	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

A *binary code* encodes each character as a binary string or *codeword*.

We would like to find a binary code that encodes the file using as few bits as possible, ie., *compresses it* as much as possible.

In a *fixed-length code* each codeword has the same length. In a *variable-length code* codewords may have different lengths. Here are examples of fixed and variable legth codes for our problem (note that a fixed-length code must have at least 3 bits per codeword).

The fixed length-code requires 100,000*3=300,000 bits to store the file.

The variable-length code

	a	b	c	d	e	f
Freq	45000	13000	12000	16000	9000	5000
A fixed lenght	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

The variable-length code uses only (45*1+13*3+12*3+16*3+9*4+5*4)*1000=224,000bits

saving a lot of space (300,000 vs 224,000)

Code

- Code is a set of codewords
- C1={**000,001,010,011,100,101**}
- C2={**0,101,100,111,1101,1100**}

	a	b	c	d	e	f
A fixed lenght	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

Example:

Then bad is encoded into

Fixed Lenght: 001000011

Variable Length: 1010111

Decoding

- Given an encoded message, *decoding* is the process of turning it back into the original message. A message is *uniquely decodable*
- Given codewords

	a	b	c	d	e	f
A fixed lenght	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

- 0110000111 fixed length
- 001010101 variable length

Decoding

- Given codewords
- 011 000 011

  d a d
- 001010101 variable length
- 0 0 101 0 101

  a a b a b

	a	b	c	d	e	f
A fixed lenght	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

Decoding Example

	a	b	c	d
Variable length codewords	1	110	10	111

- Relative to above codewords 1101111 is not uniquely decoded since it could have encoded either bad or acad

110 1 111

b a d

1 10 1 111

a c a d

Prefix-Codes

Fixed-length codes are always uniquely decodable(why)

Prefix Code: A code is called a prefix (free) code if no codeword is a prefix of another one.

Example

	a	b	c	d	e	f
Variable length	0	101	100	111	1101	1100

İs a prefix code.

Prefix Code

Important Fact: Every message encoded by a prefix free code is uniquely decodable. Since no code- word is a prefix of any other we can always find the first codeword in a message, peel it off, and continue decoding.

	a	b	c	d	e	f
Variable length	0	101	100	111	1101	1100

Example

0101100111000 1100 = abcdaaf

We are therefore interested in finding good (best compression) prefix-free codes.

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

Prioritize characters

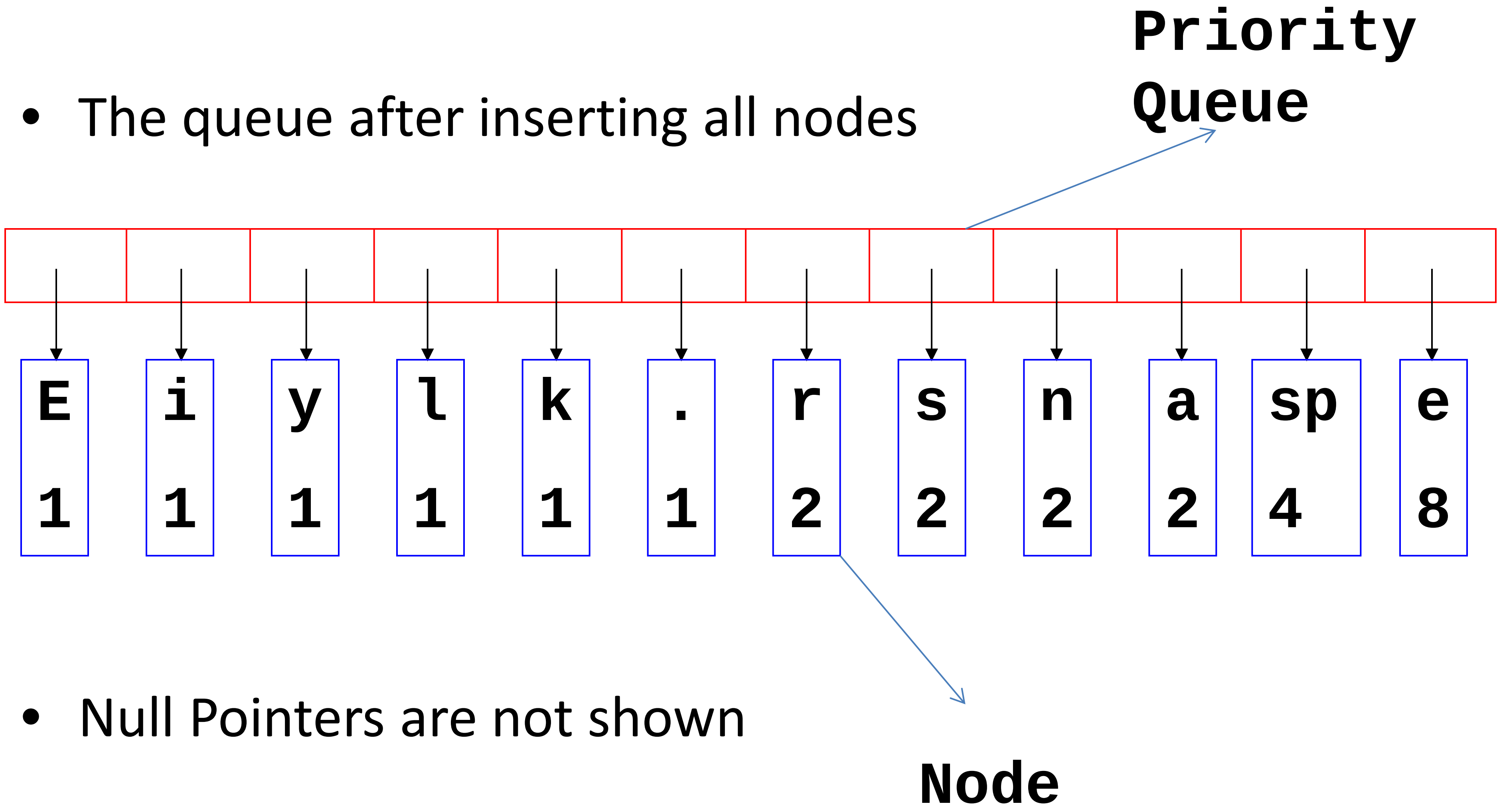
Uses binary tree nodes

```
struct _HuffNode
{
   char myChar;
   int myFrequency;
   struct HuffNode *left, *right;
}

priorityQueue myQueue;
```

Building a Tree

- The queue after inserting all nodes

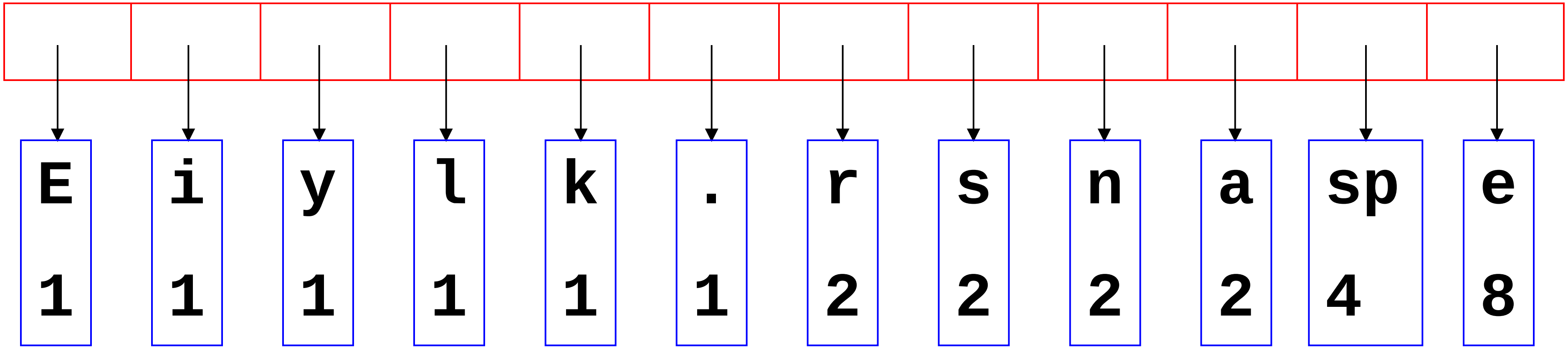


- Null Pointers are not shown

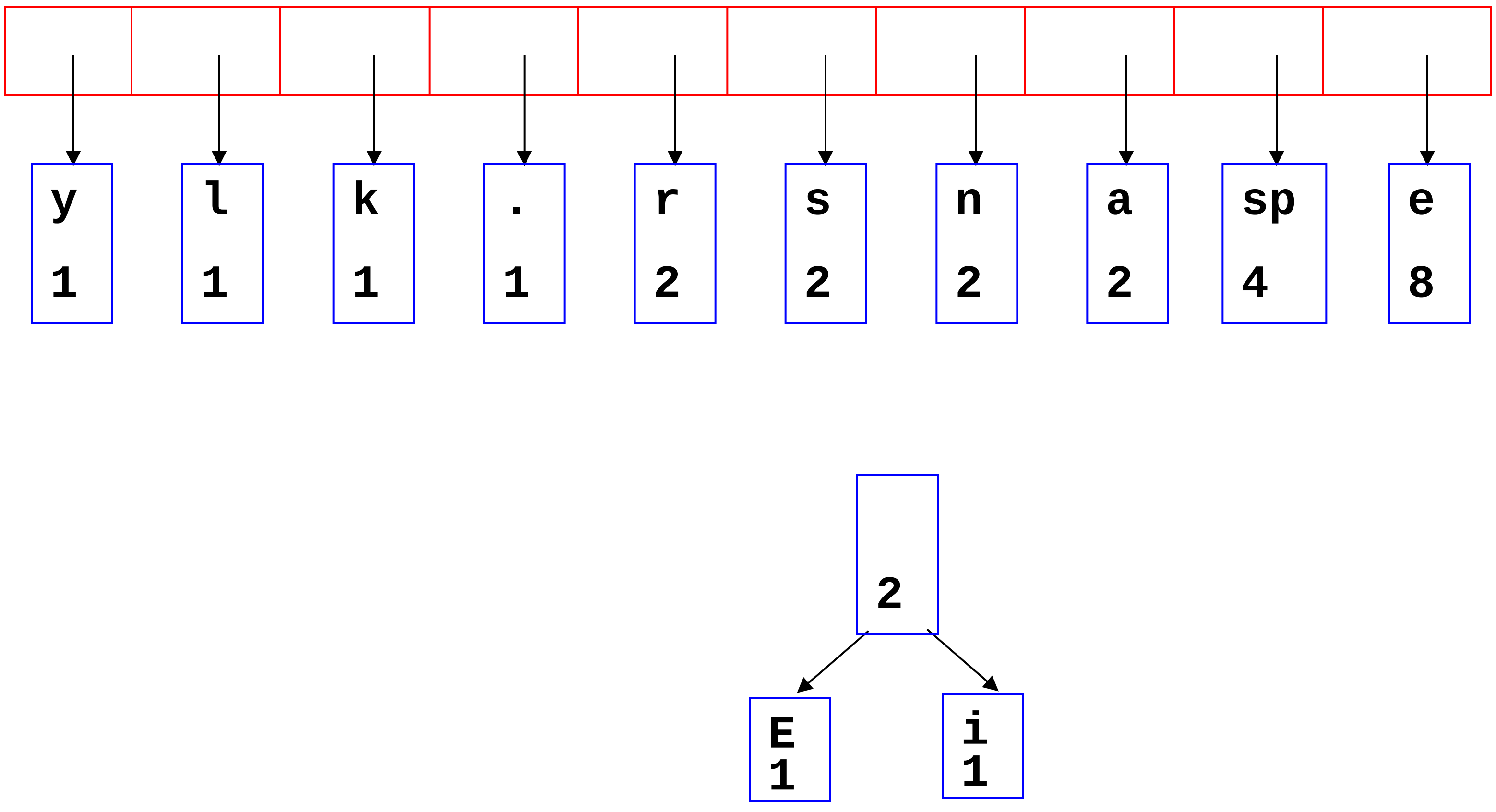
Building a Tree

- 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

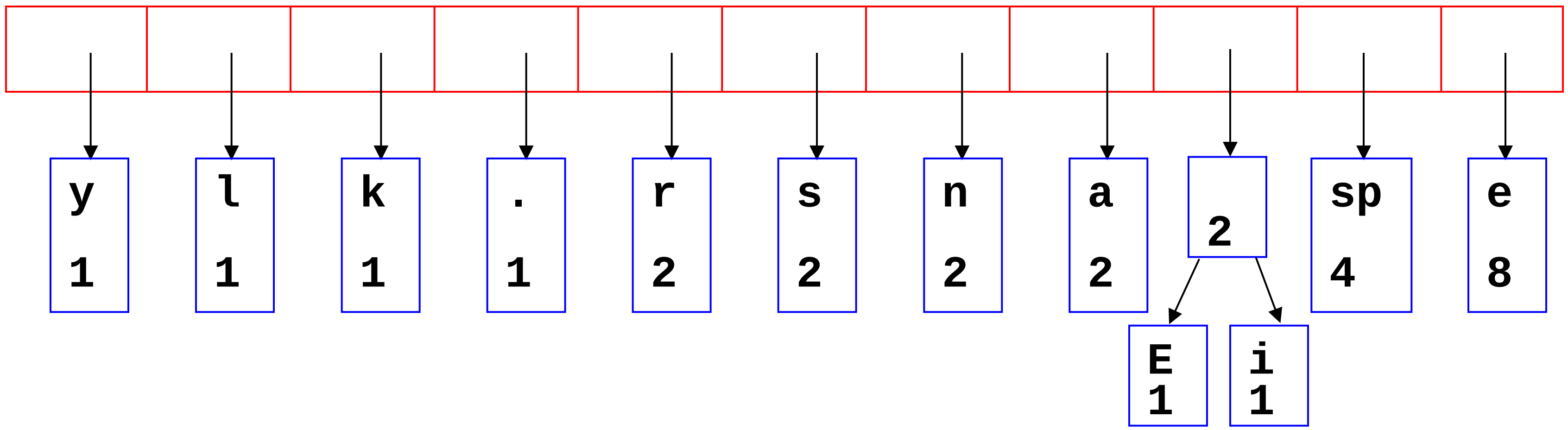
Building a Tree



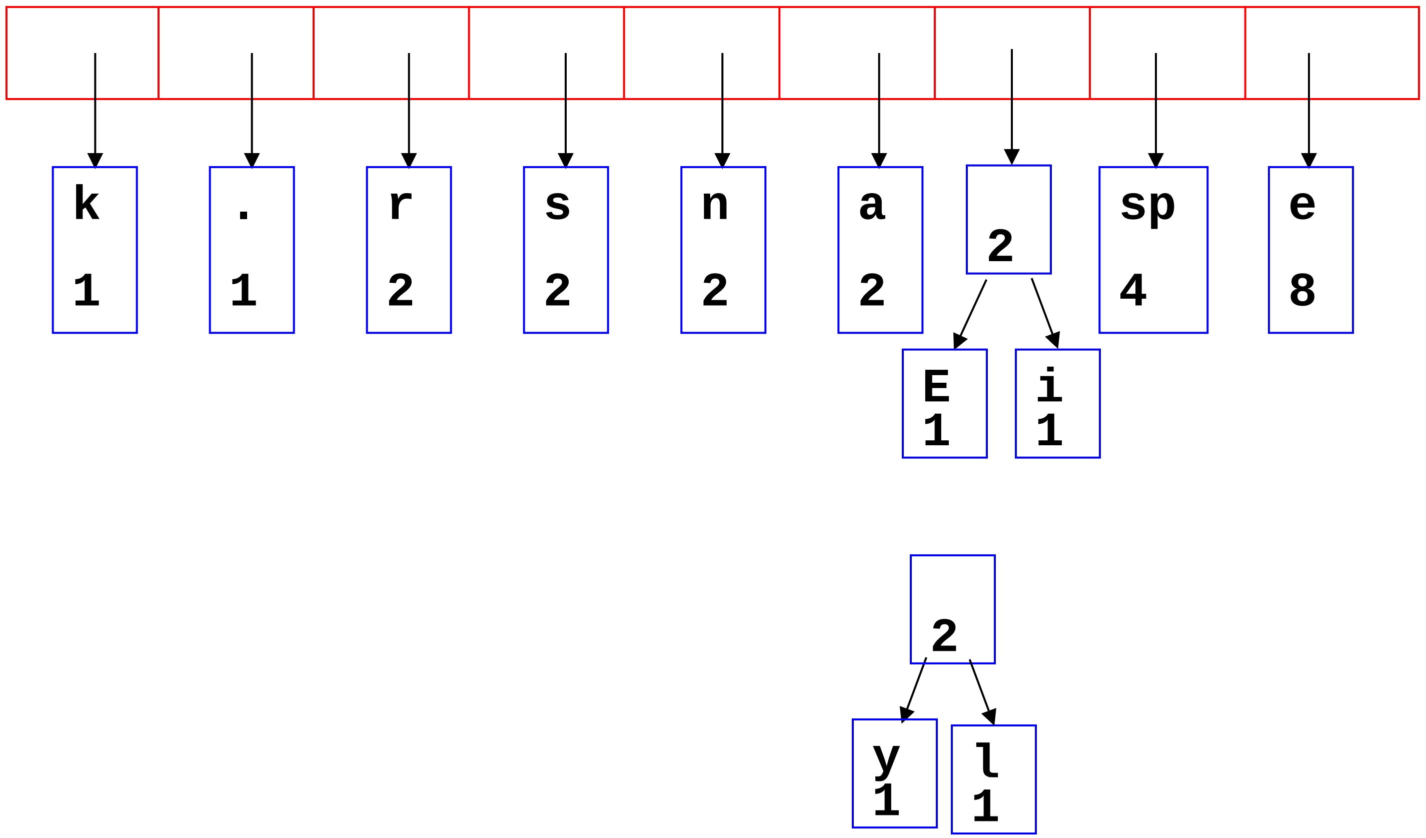
Building a Tree



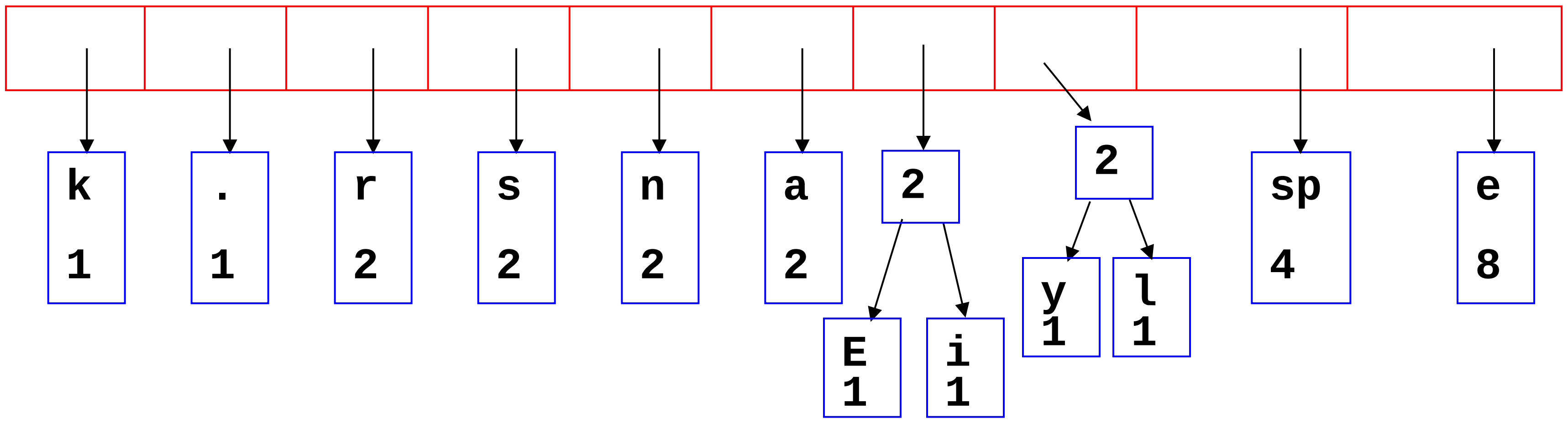
Building a Tree



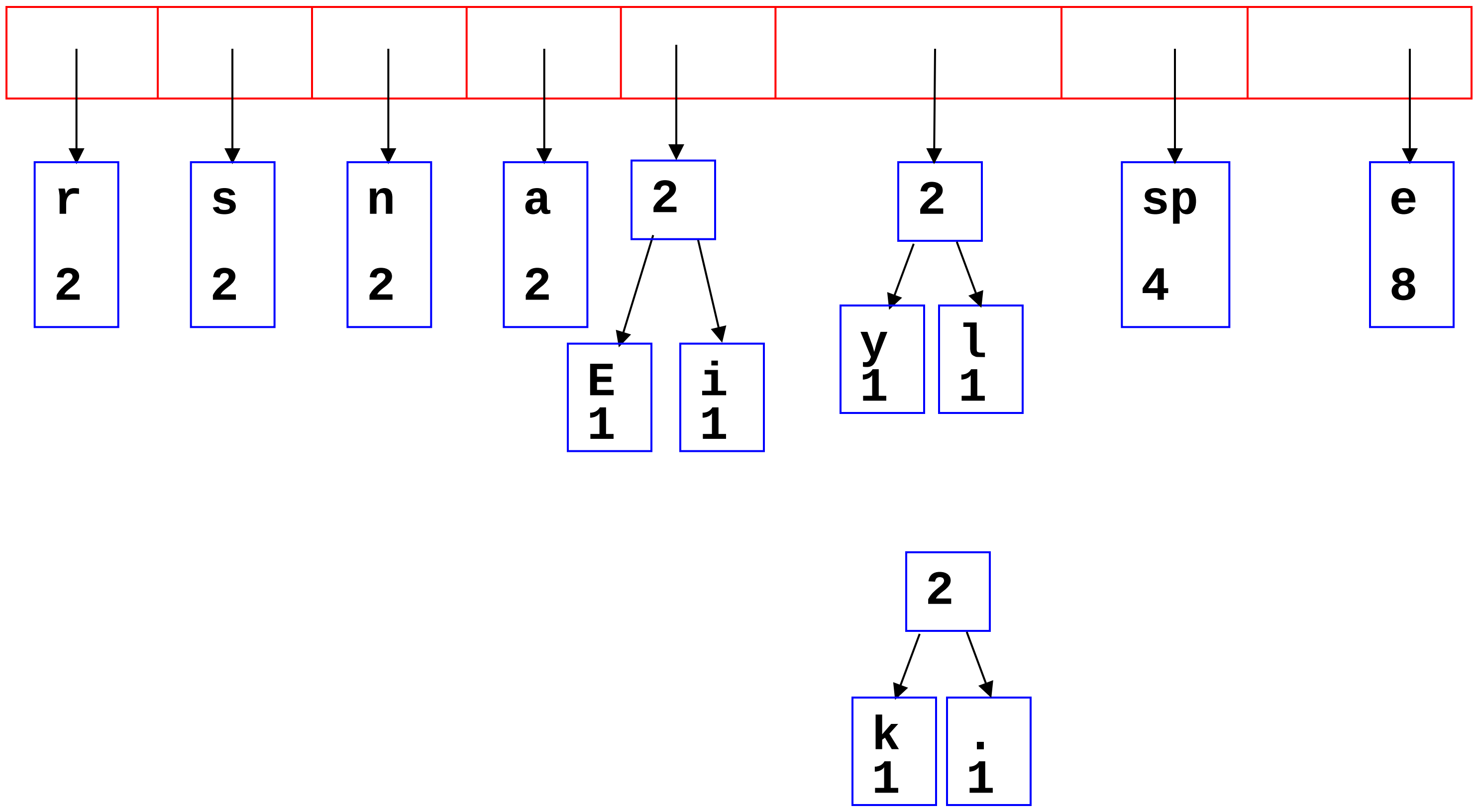
Building a Tree



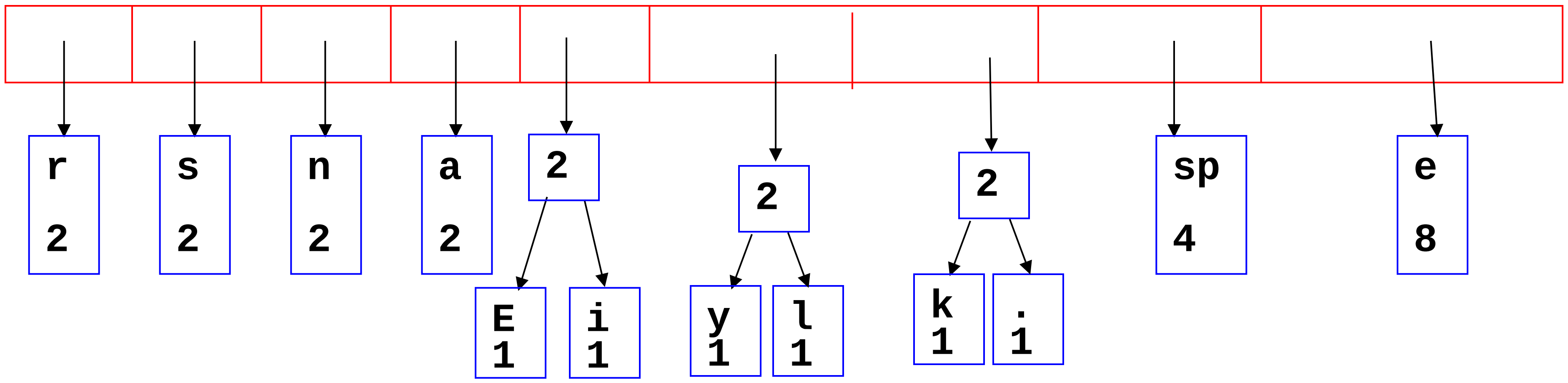
Building a Tree



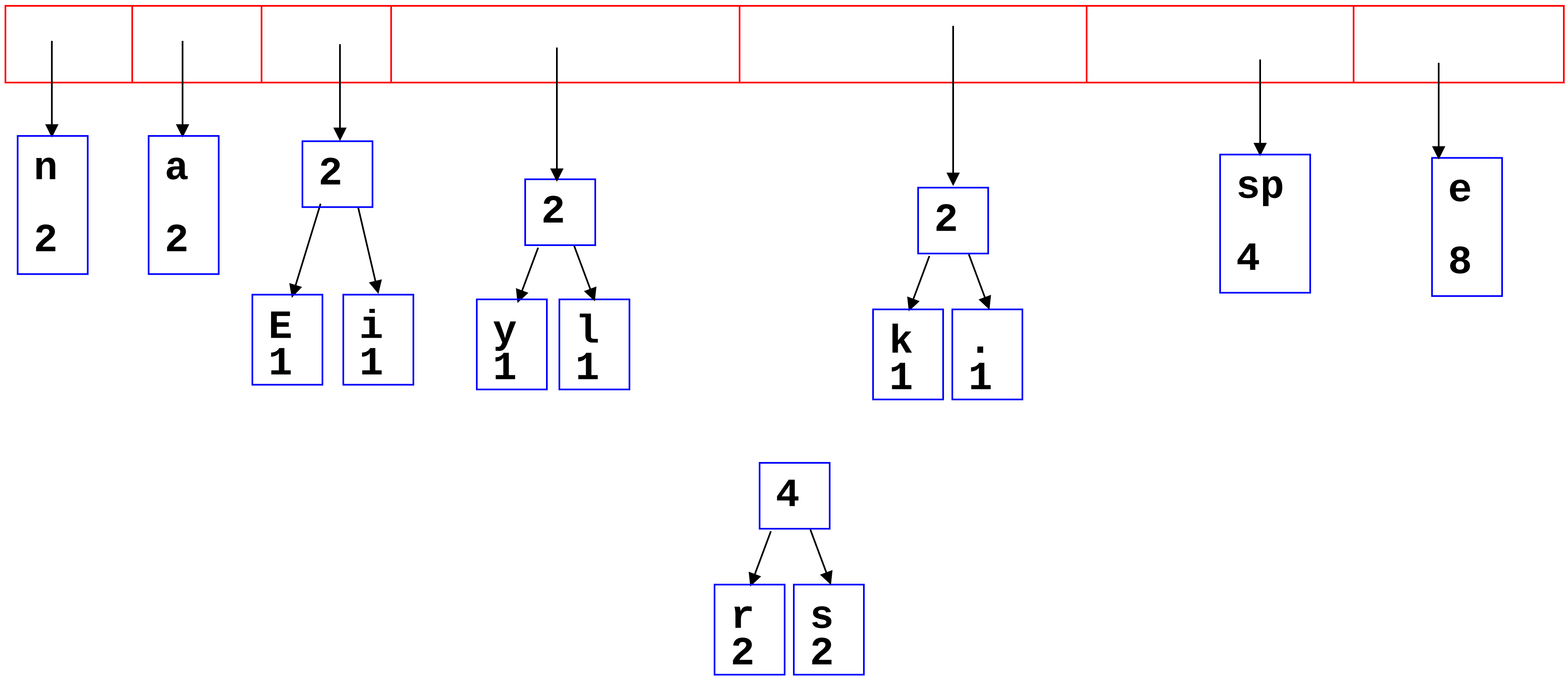
Building a Tree



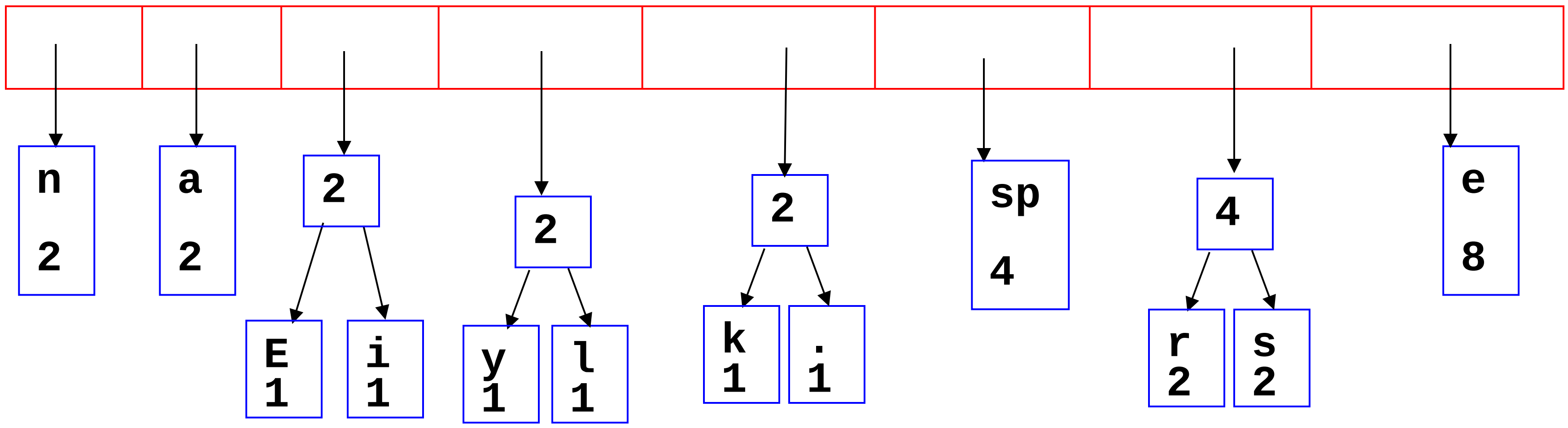
Building a Tree



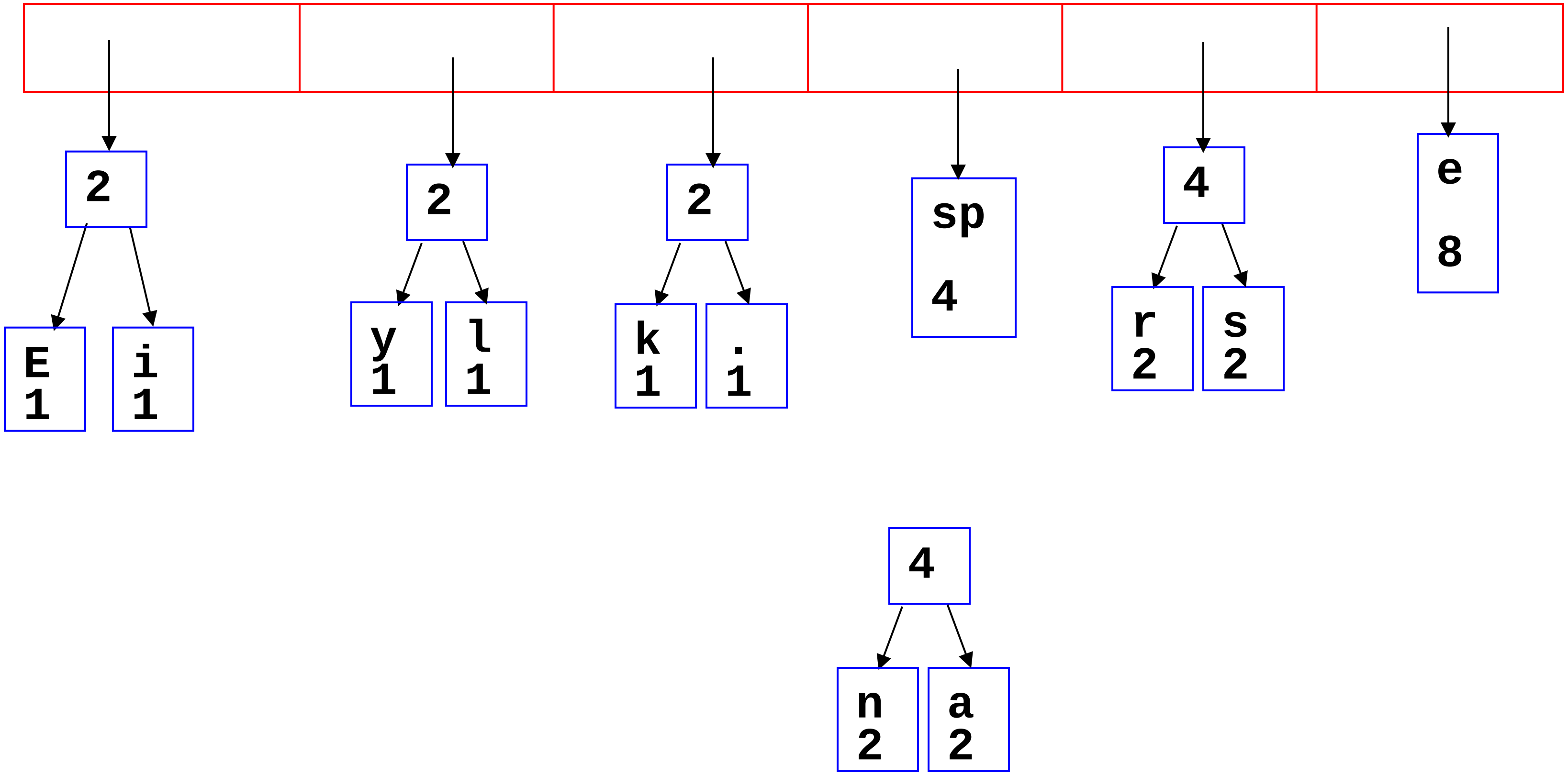
Building a Tree



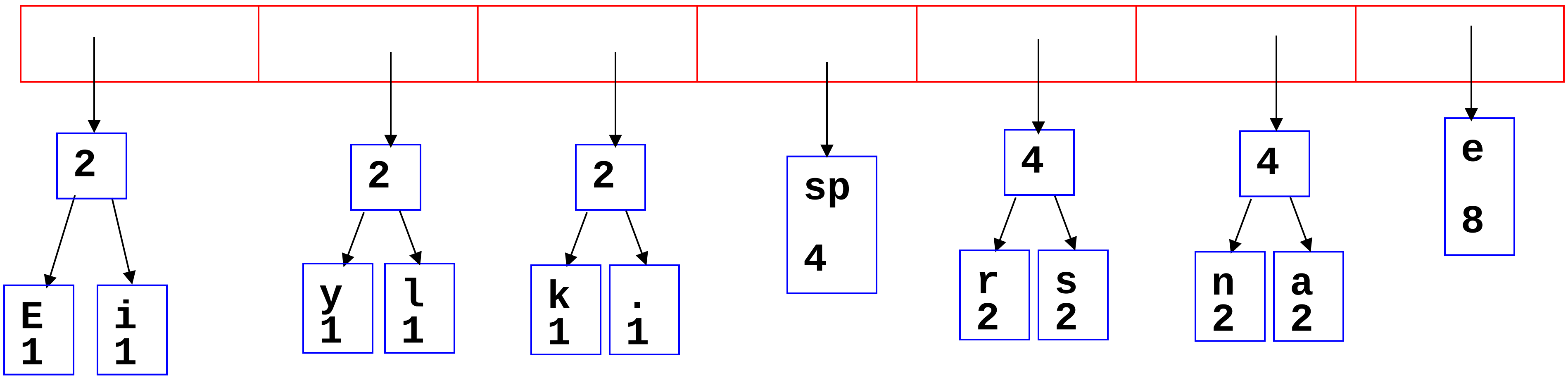
Building a Tree



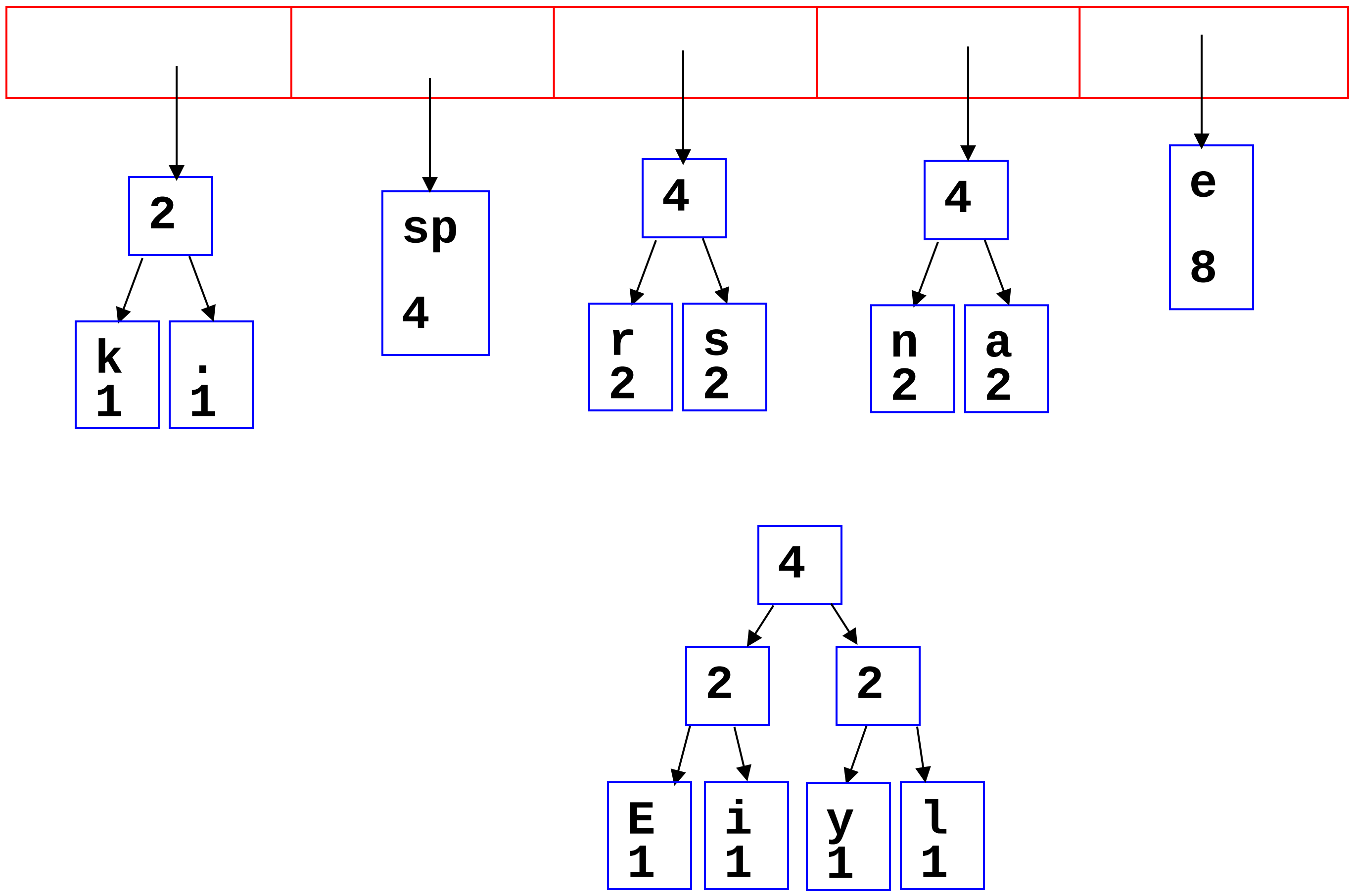
Building a Tree



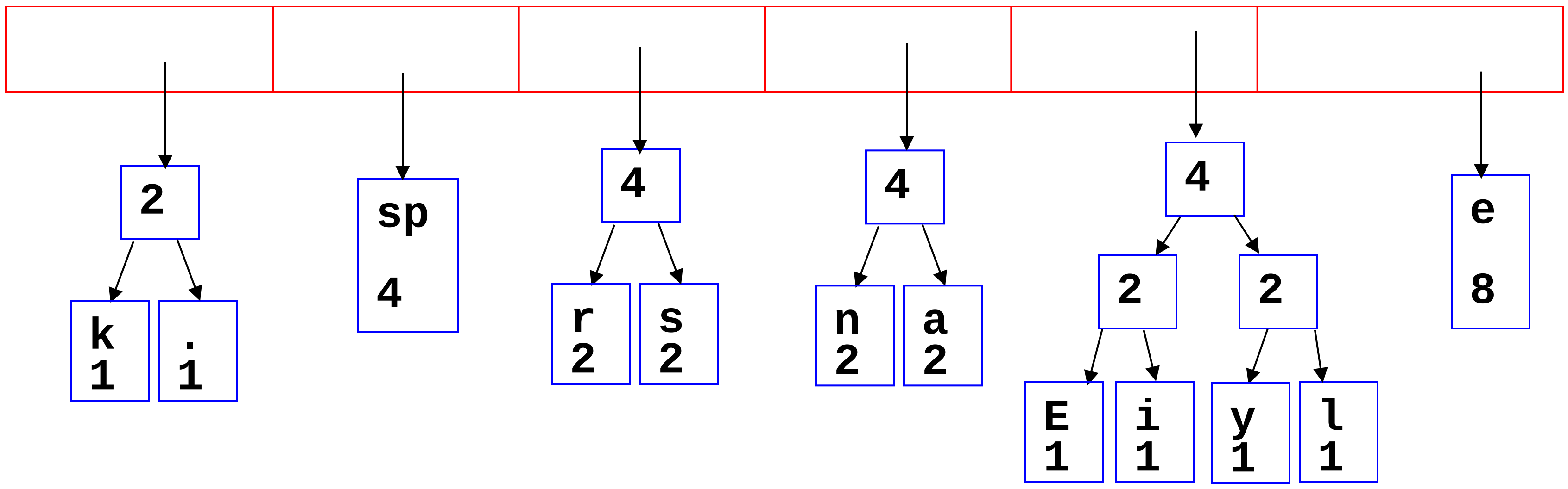
Building a Tree



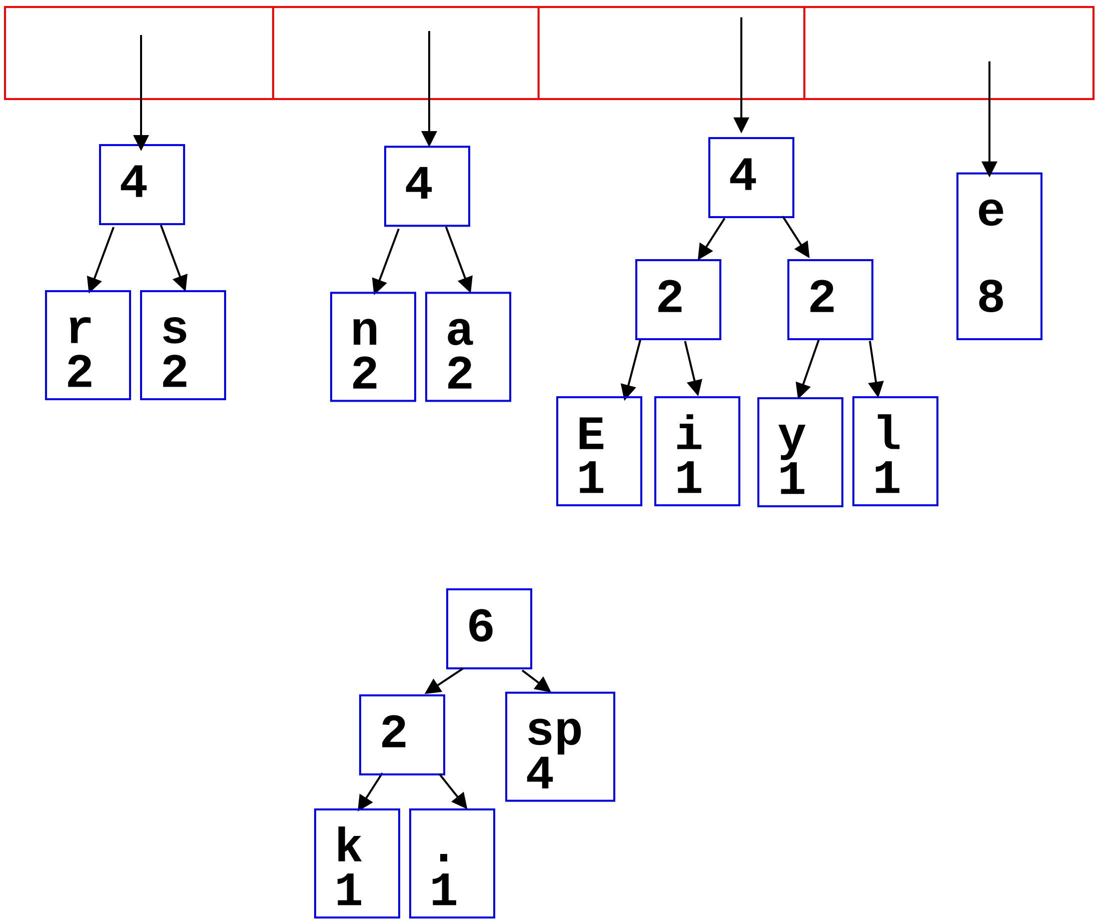
Building a Tree



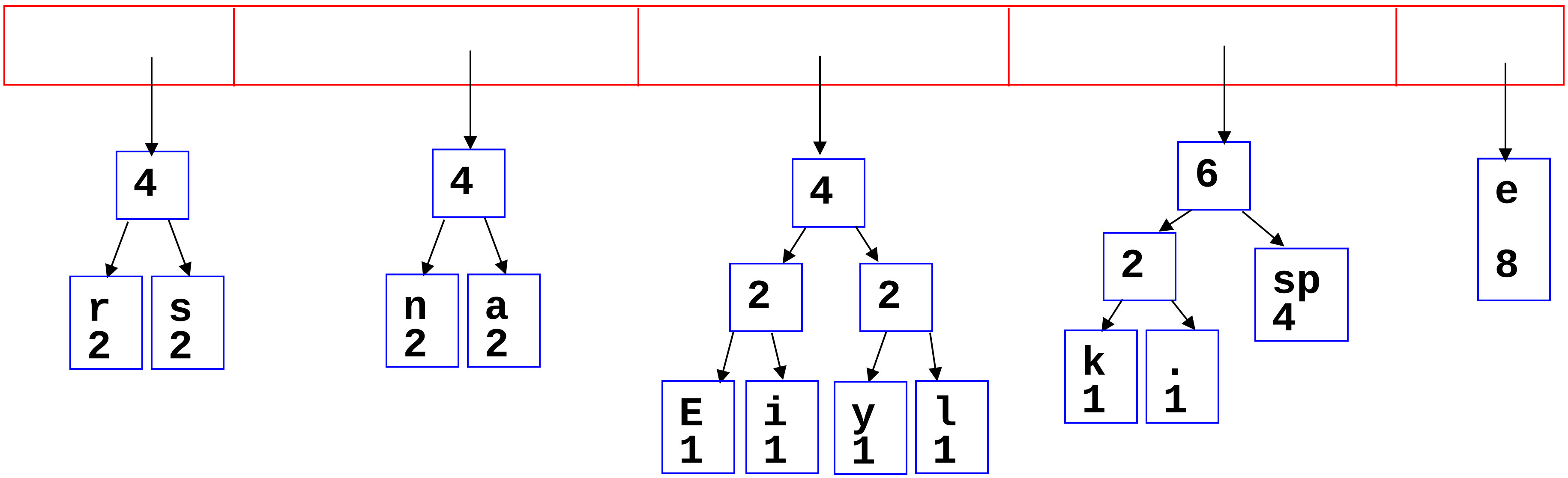
Building a Tree



Building a Tree

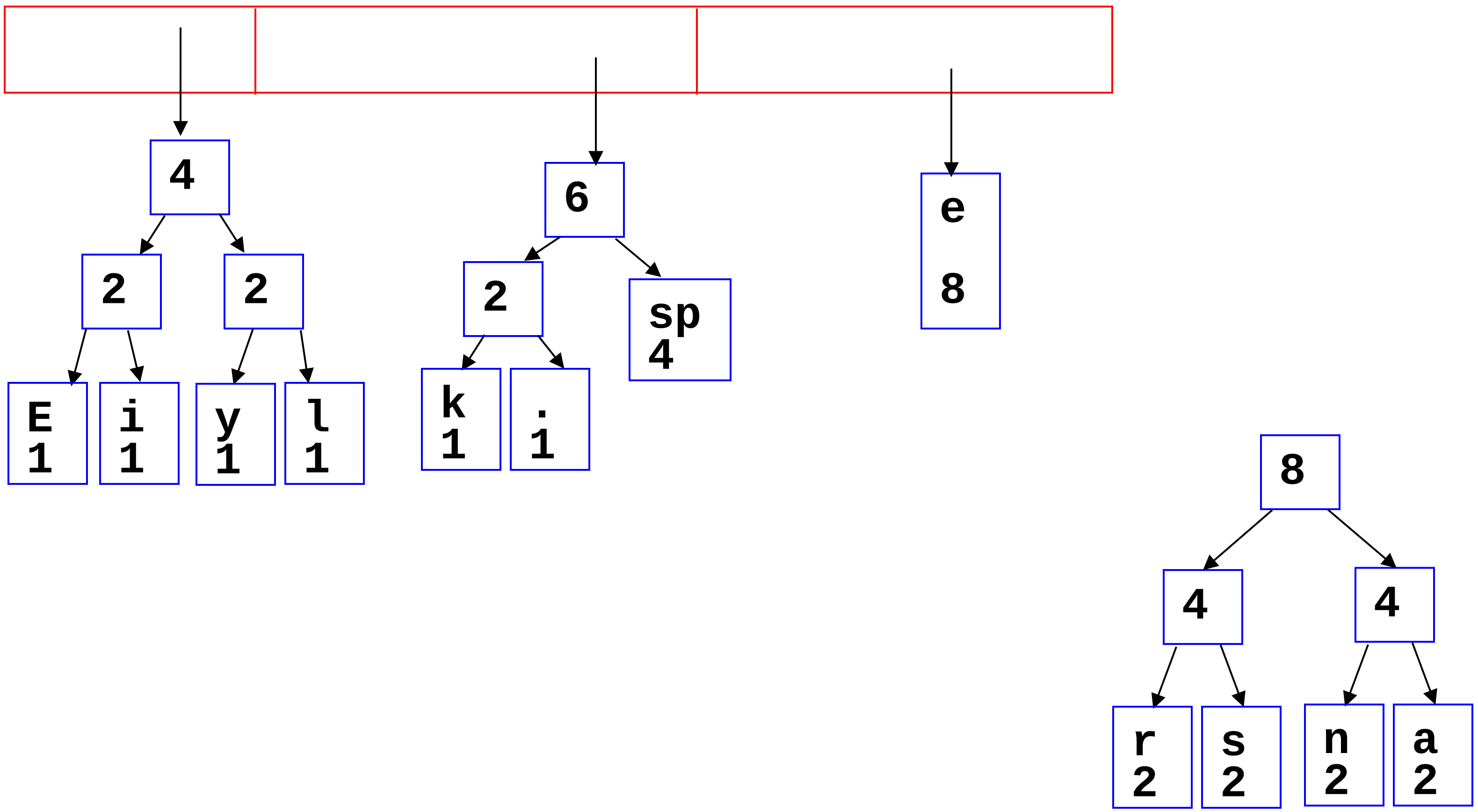


Building a Tree

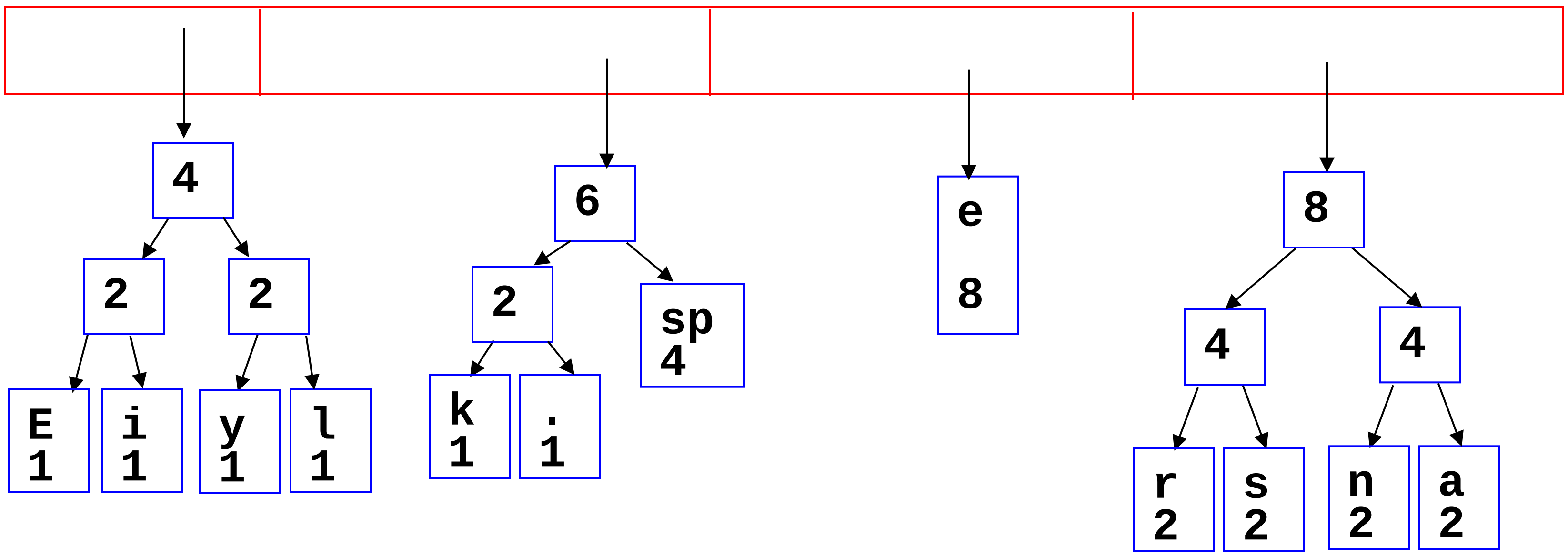


What is happening to the characters with a low number of occurrences?

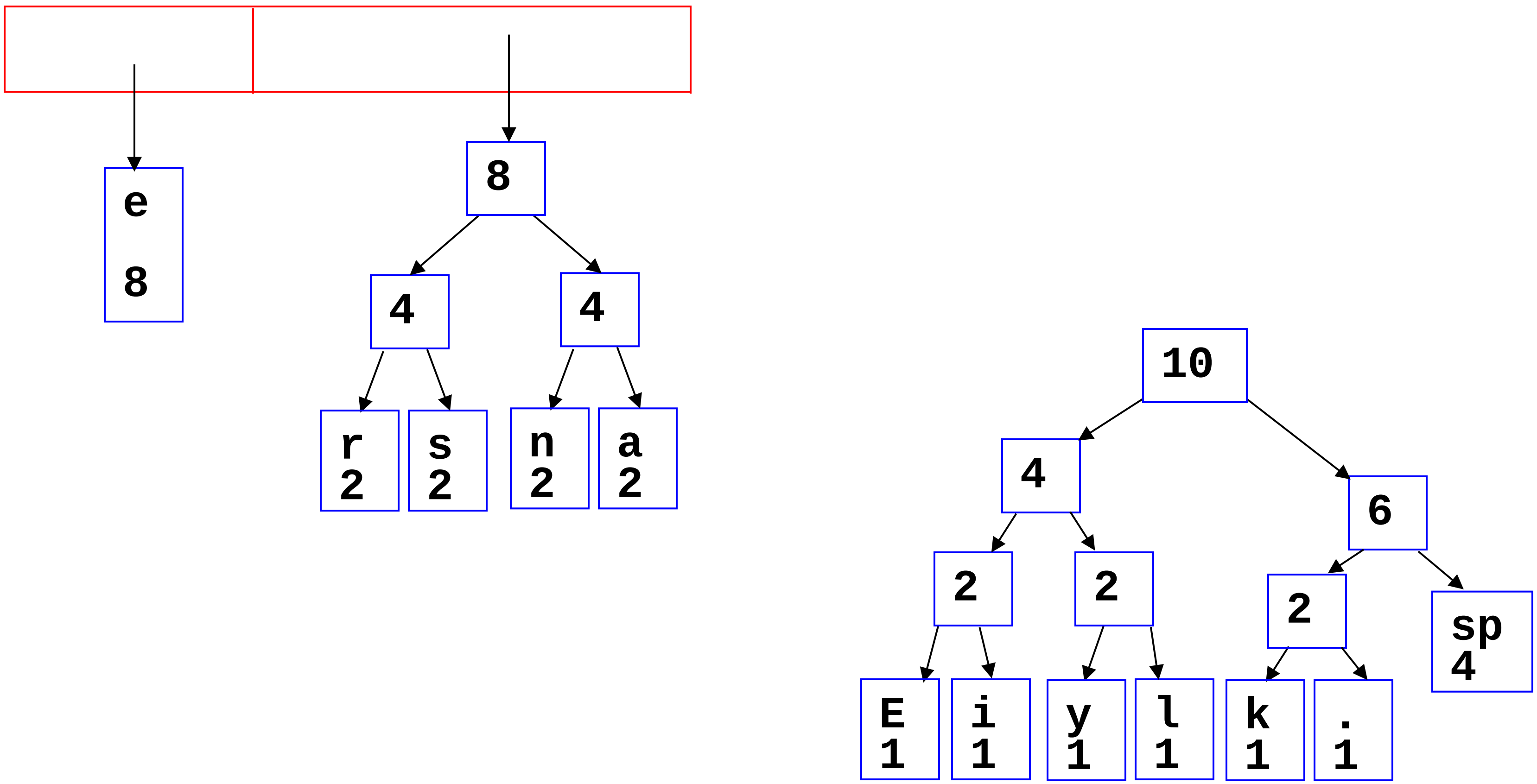
Building a Tree



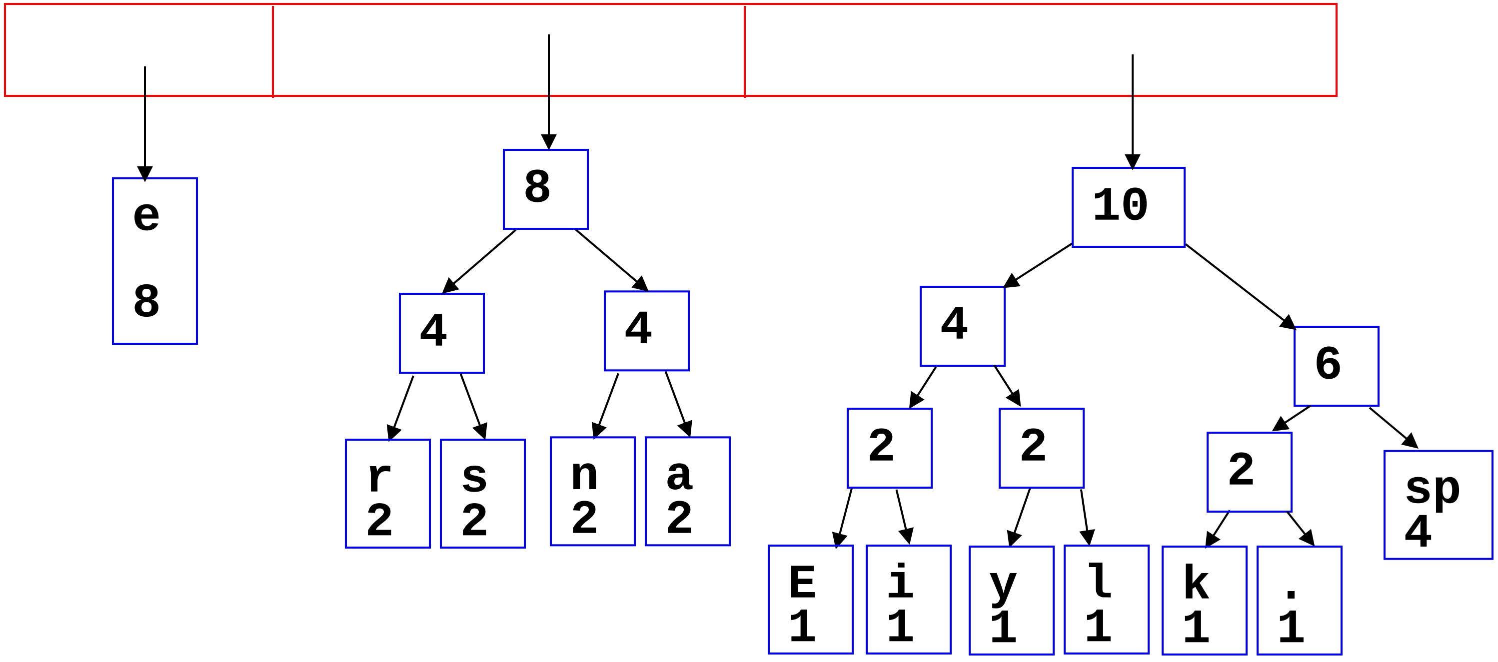
Building a Tree



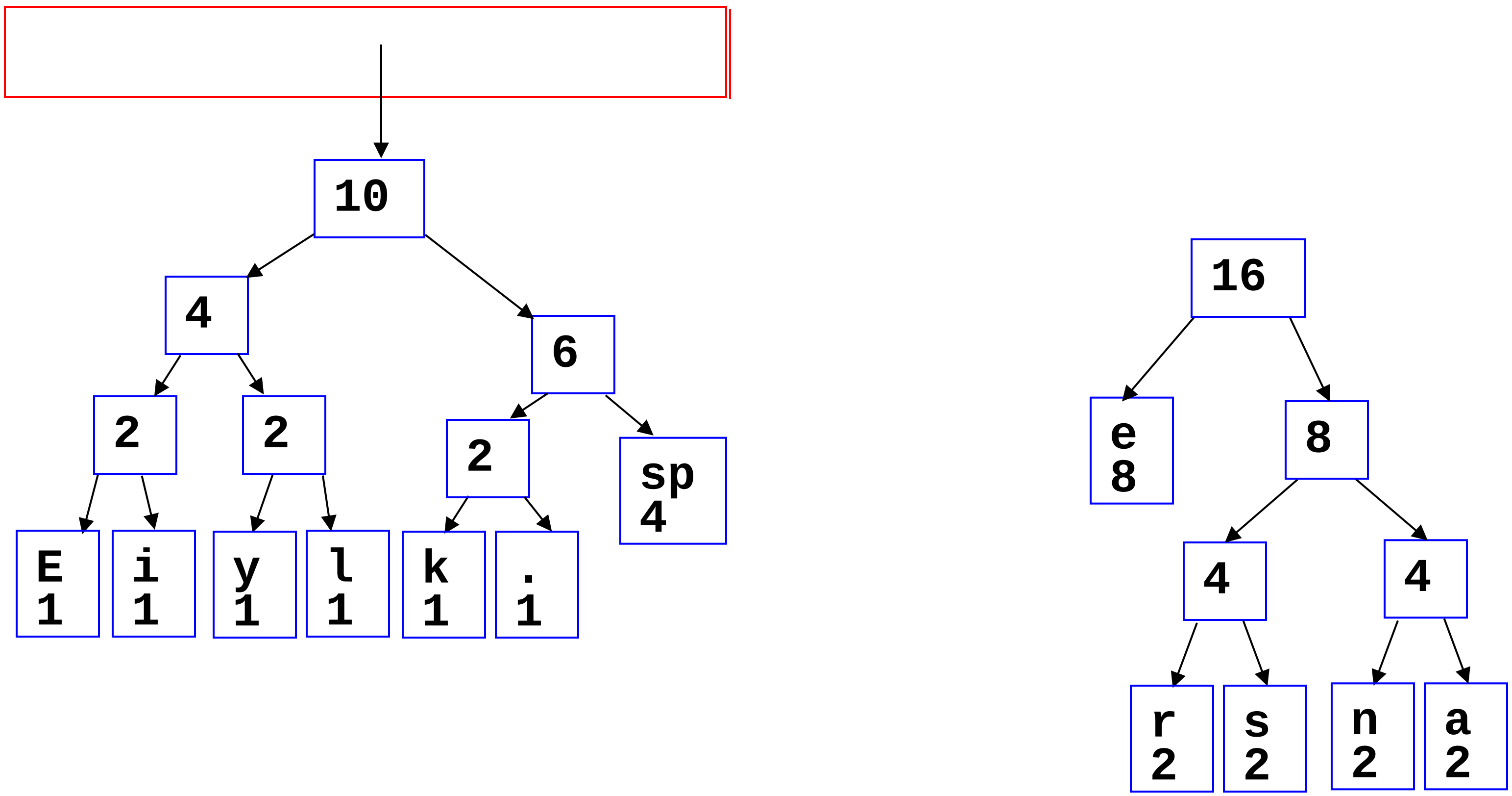
Building a Tree



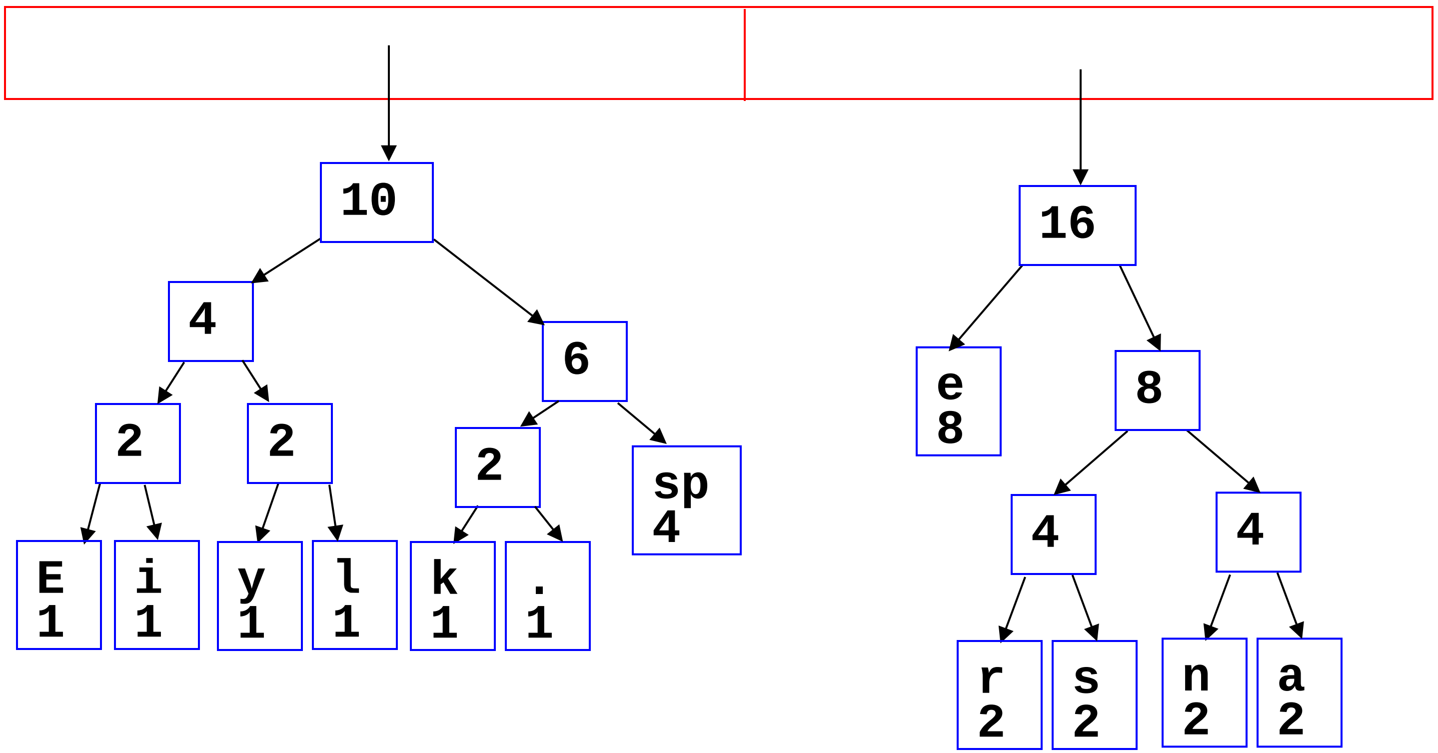
Building a Tree



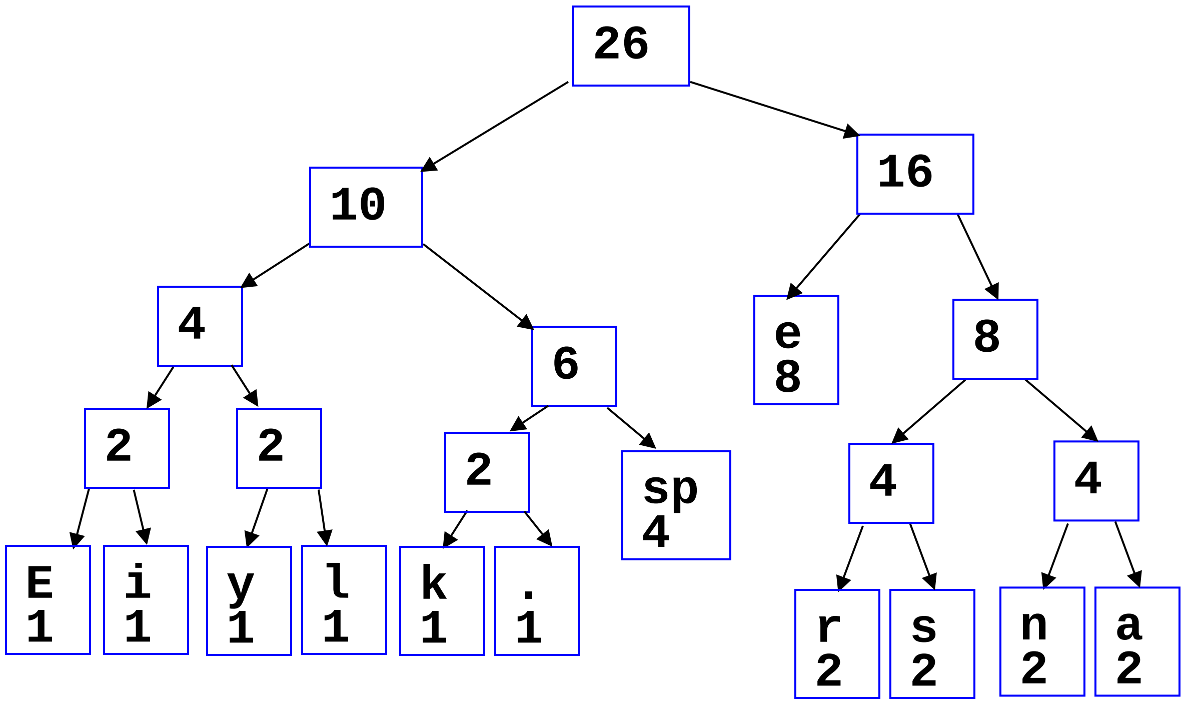
Building a Tree



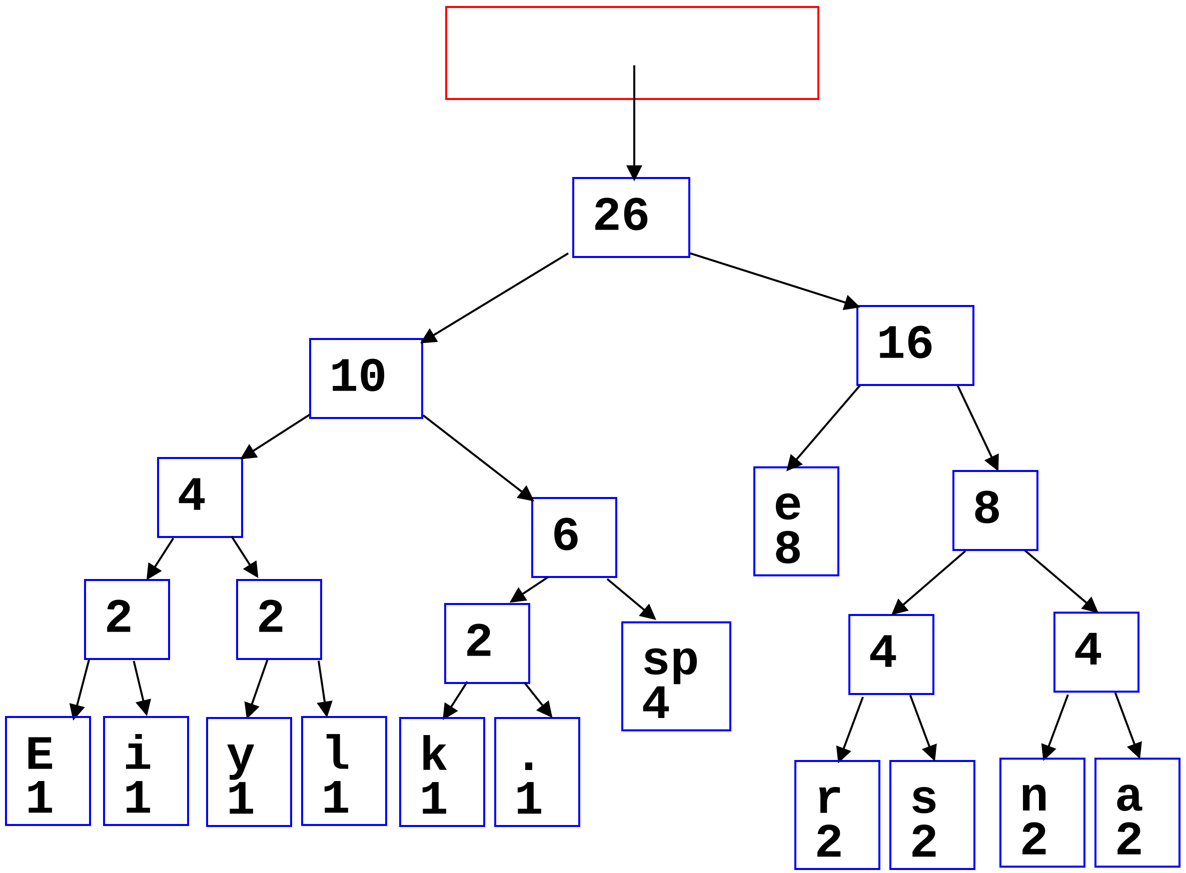
Building a Tree



Building a Tree



Building a Tree



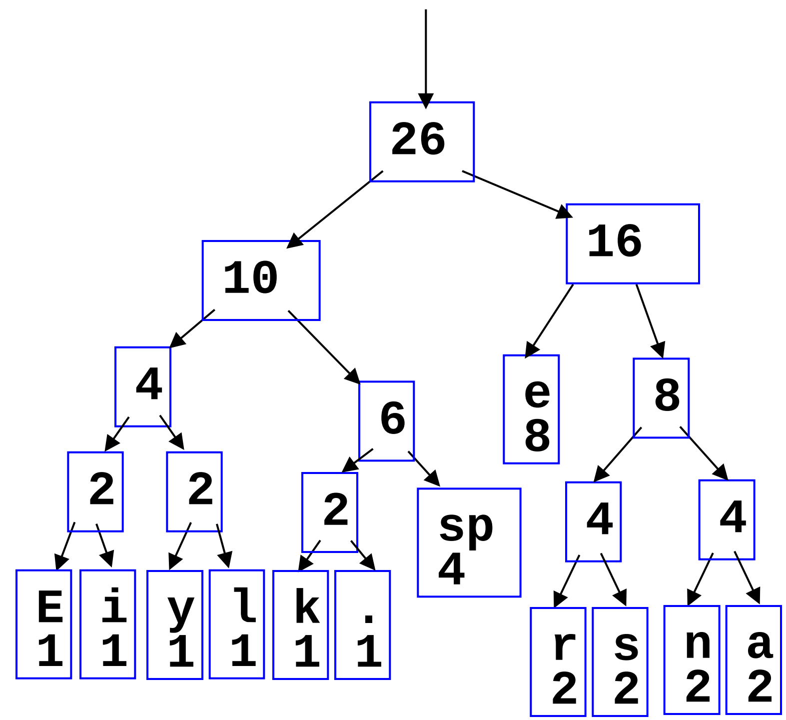
- **After enqueueing this node there is only one node left in priority queue.**

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.

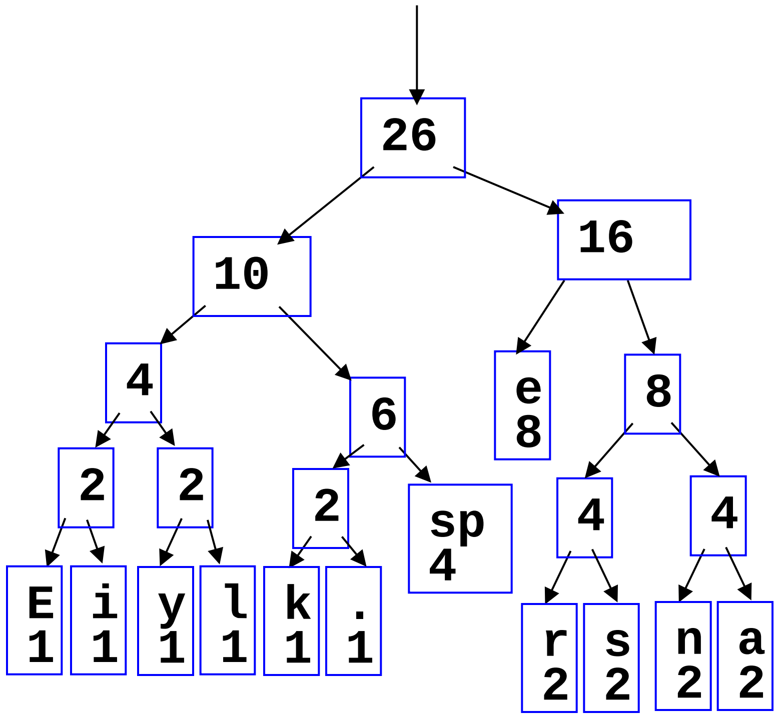


Eerie eyes seen near lake. 26 characters

Encoding the File

Traverse Tree for Codes

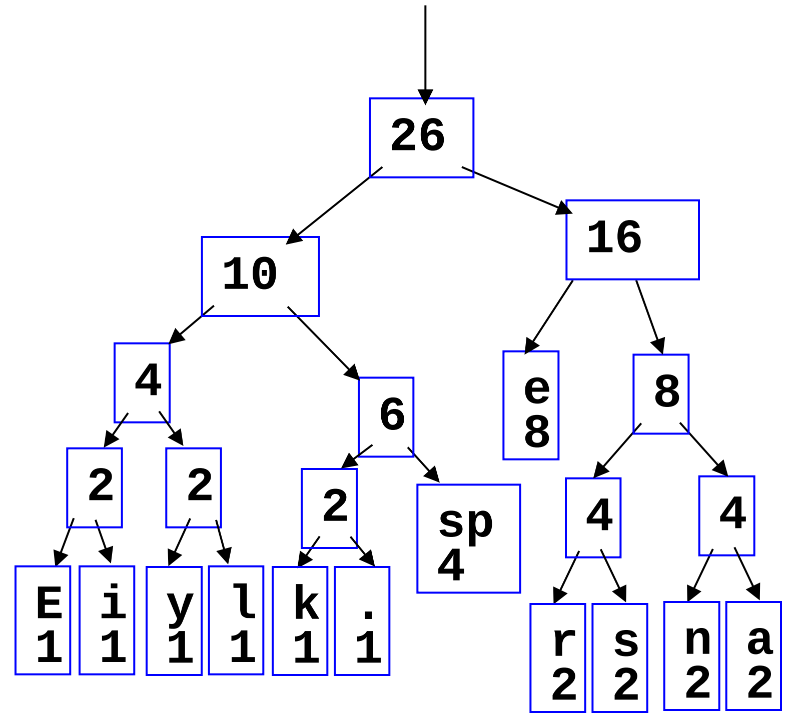
- Perform a traversal of the tree to obtain new code words
- Going left is a 0
  going right is a 1
- code word is only completed when a leaf node is reached



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
```

- **Why is there no need for a separator character?**

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
- ASCII would take 8 * 26 = 208 bits

```
0000101100000110011
1000101011011010011
1110101111110001100
1111110100100101
```

- **`If modified code used 4 bits per character are needed.  Total bits 4 * 26 = 104.  Savings not as great.`**

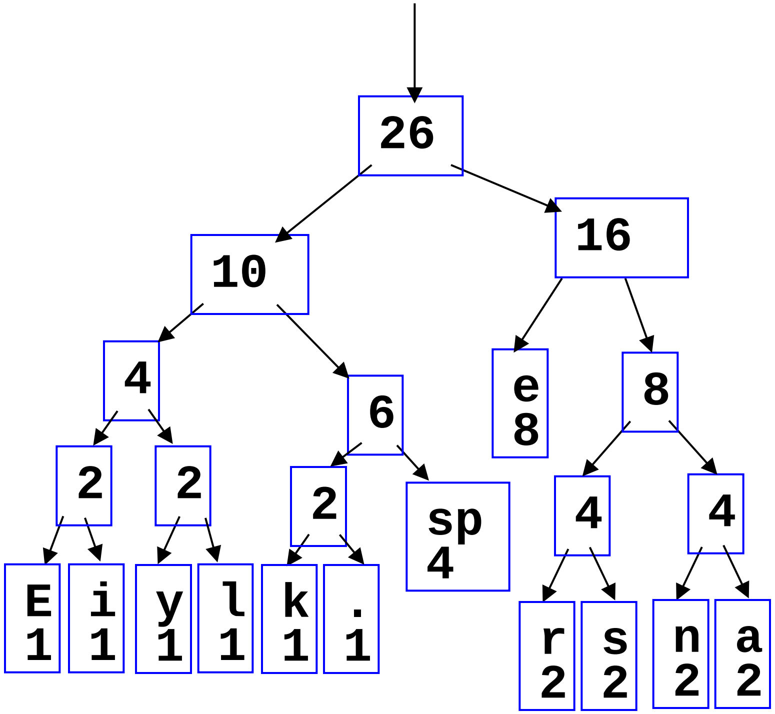
Decoding the File

- 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
- Tree predetermined
  - based on statistical analysis of text files or file types
- Data transmission is bit based versus byte based

Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 $\Rightarrow$ go left
- 1 $\Rightarrow$ go right

101000110111101111
011111110000110101



Summary

- Huffman coding is a technique used to compress files for transmission
- Uses statistical coding
  - more frequently used symbols have shorter code words
- Works well for text and fax transmissions