Huffman Codes

Encoding messages

- Encode a message composed of a string of characters
- Codes used by computer systems
  - ASCII
    - uses 8 bits per character
    - can encode 256 characters
  - Unicode
    - 16 bits per character
    - can encode 65536 characters
    - includes all characters encoded by ASCII
- ASCII and Unicode are *fixed-length codes*
  - all characters represented by same number of bits

Problems

- Suppose that we want to encode a message constructed from the symbols **A**, **B**, **C**, **D**, and **E** using a fixed-length code
  - How many bits are required to encode each symbol?
    - at least 3 bits are required
    - 2 bits are not enough (can only encode four symbols)
  - How many bits are required to encode the message **DEAACAAAAABA**?
    - there are twelve symbols, each requires 3 bits
    - 12*3 = 36 bits are required

Drawbacks of fixed-length codes

- Wasted space
  - Unicode uses twice as much space as ASCII
    - inefficient for plain-text messages containing only ASCII characters
- Same number of bits used to represent all characters
  - ‘a’ and ‘e’ occur more frequently than ‘q’ and ‘z’

- **Potential solution**: use variable-length codes
  - variable number of bits to represent characters when frequency of occurrence is known
  - short codes for characters that occur frequently

Advantages of variable-length codes

- The advantage of variable-length codes over fixed-length is short codes can be given to characters that occur frequently
  - on average, the length of the encoded message is less than fixed-length encoding
- **Potential problem:** how do we know where one character ends and another begins?
    - not a problem if number of bits is fixed!

A = 00
B = 01
C = 10
D = 11

0010110111001111111111

A C D B A D D D D D

Prefix property

- A code has the **prefix property** if no character code is the prefix (start of the code) for another character
- Example:

Symbol	Code
P	000
Q	11
R	01
S	001
T	10

01001101100010

R S T Q P T

- 000 is not a prefix of 11, 01, 001, or 10
- 11 is not a prefix of 000, 01, 001, or 10 …

Code without prefix property

- The following code does **not** have prefix property

Symbol	Code
P	0
Q	1
R	01
S	10
T	11

- The pattern **1110** can be decoded as **QQQP**, **QTP**, **QQS**, or **TS**

Problem

- Design a variable-length prefix-free code such that the message **DEAACAAAAABA** can be encoded using 22 bits
- Possible solution:
  - **A** occurs eight times while **B**, **C**, **D**, and **E** each occur once
  - represent **A** with a one bit code, say 0
    - remaining codes cannot start with 0
  - represent **B** with the two bit code 10
    - remaining codes cannot start with 0 or 10
  - represent **C** with 110
  - represent **D** with 1110
  - represent **E** with 11110

Encoded message

DEAACAAAAABA

Symbol	Code
A	0
B	10
C	110
D	1110
E	11110

1110111100011000000100 22 bits

Another possible code

DEAACAAAAABA

Symbol	**Code**
A	0
B	100
C	101
D	1101
E	1111

1101111100101000001000 22 bits

Better code

DEAACAAAAABA

Symbol	**Code**
A	0
B	100
C	101
D	110
E	111

11011100101000001000 20 bits

What code to use?

- Question: Is there a variable-length code that makes the most efficient use of space?

Answer: Yes!

Huffman coding tree

- Binary tree
  - each leaf contains symbol (character)
  - label edge from node to left child with 0
  - label edge from node to right child with 1
- Code for any symbol obtained by following path from root to the leaf containing symbol
- Code has prefix property
  - leaf node cannot appear on path to another leaf
  - *note*: fixed-length codes are represented by a complete Huffman tree and clearly have the prefix property

Building a Huffman tree

- Find frequencies of each symbol occurring in message
- Begin with a forest of single node trees
  - each contain symbol and its frequency
- Do recursively
  - select two trees with smallest frequency at the root
  - produce a new binary tree with the selected trees as children and store the sum of their frequencies in the root
- Recursion ends when there is one tree
  - this is the Huffman coding tree

Example

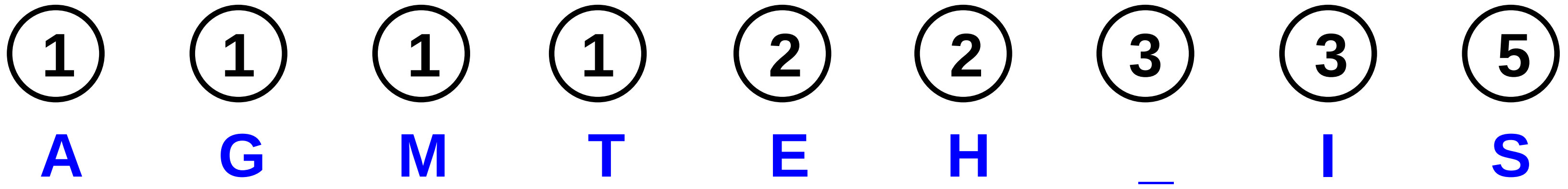
- Build the Huffman coding tree for the message

  This is his message

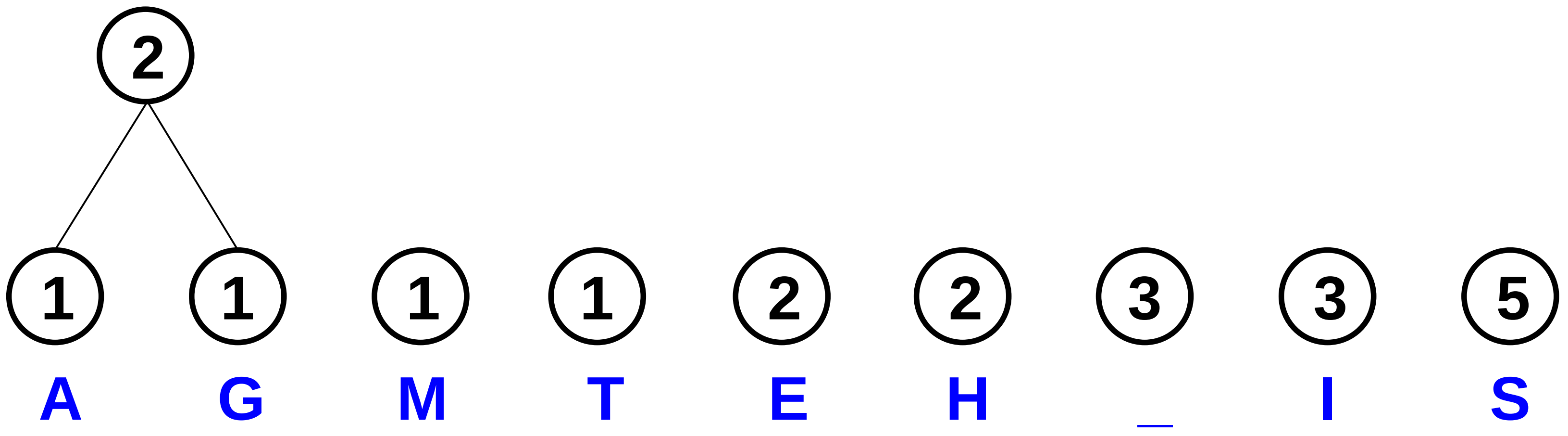
- Character frequencies

A	G	M	T	E	H	_	I	S
1	1	1	1	2	2	3	3	5

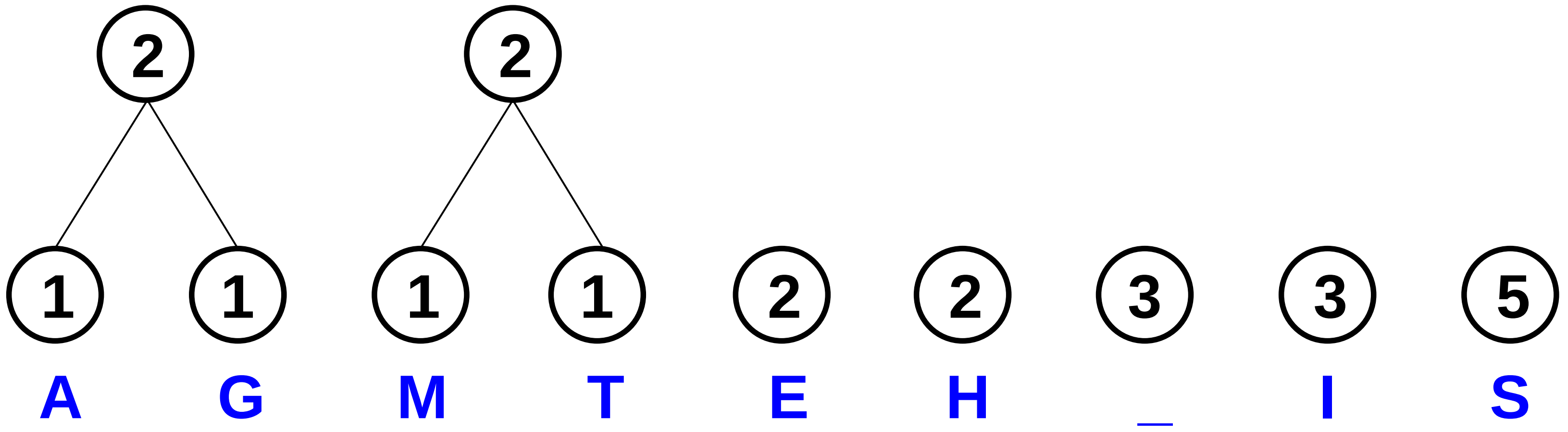
- Begin with forest of single trees



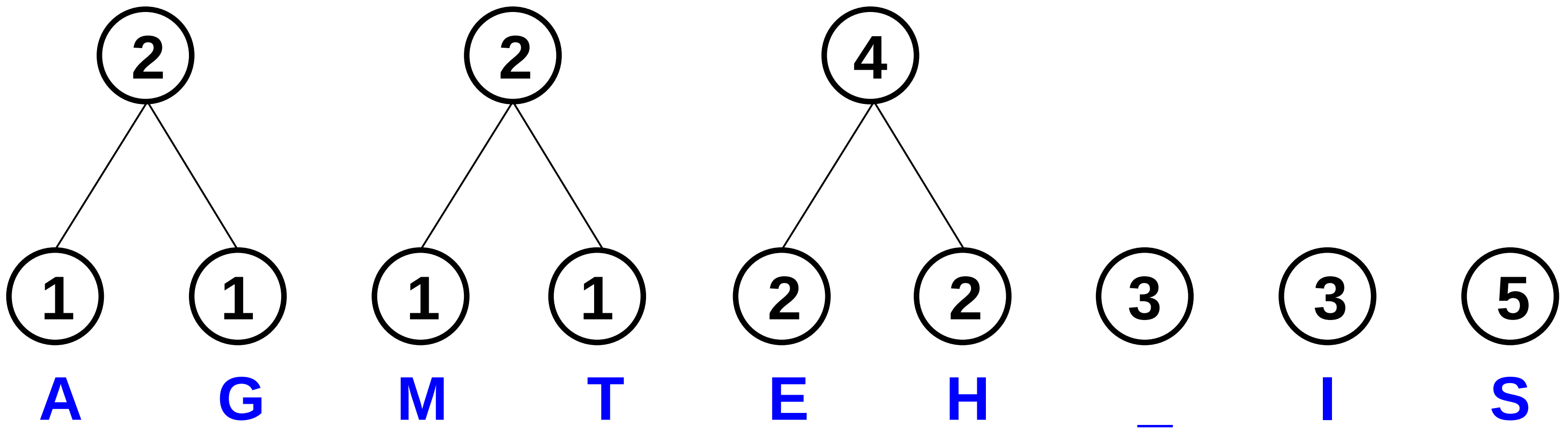
Step 1



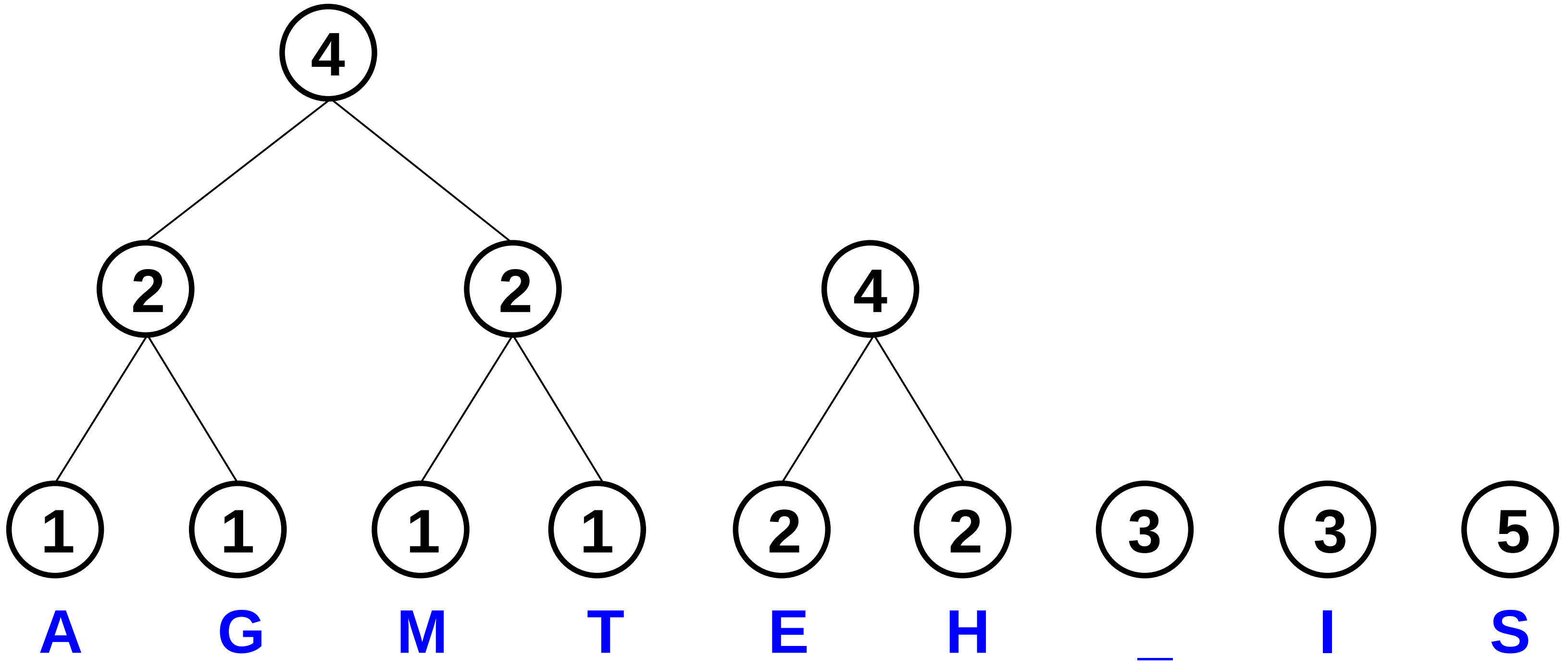
Step 2



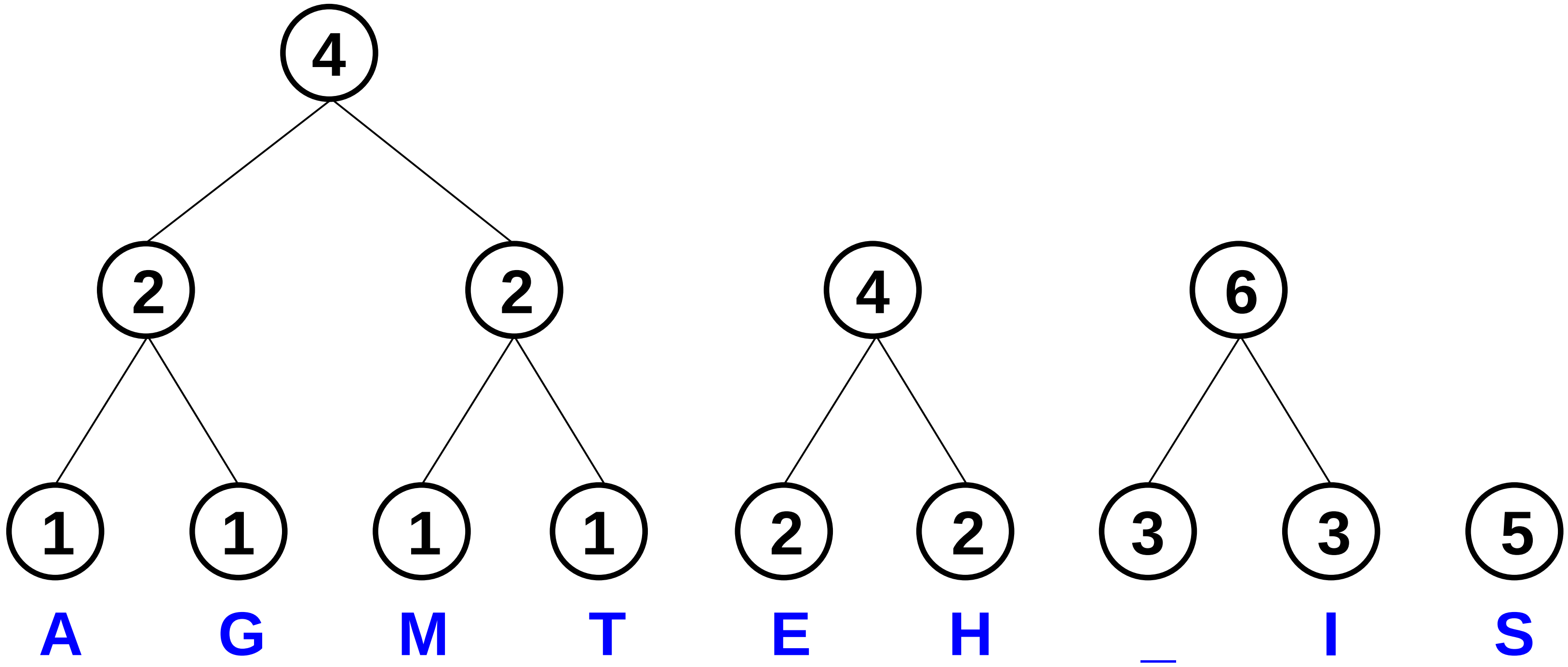
Step 3



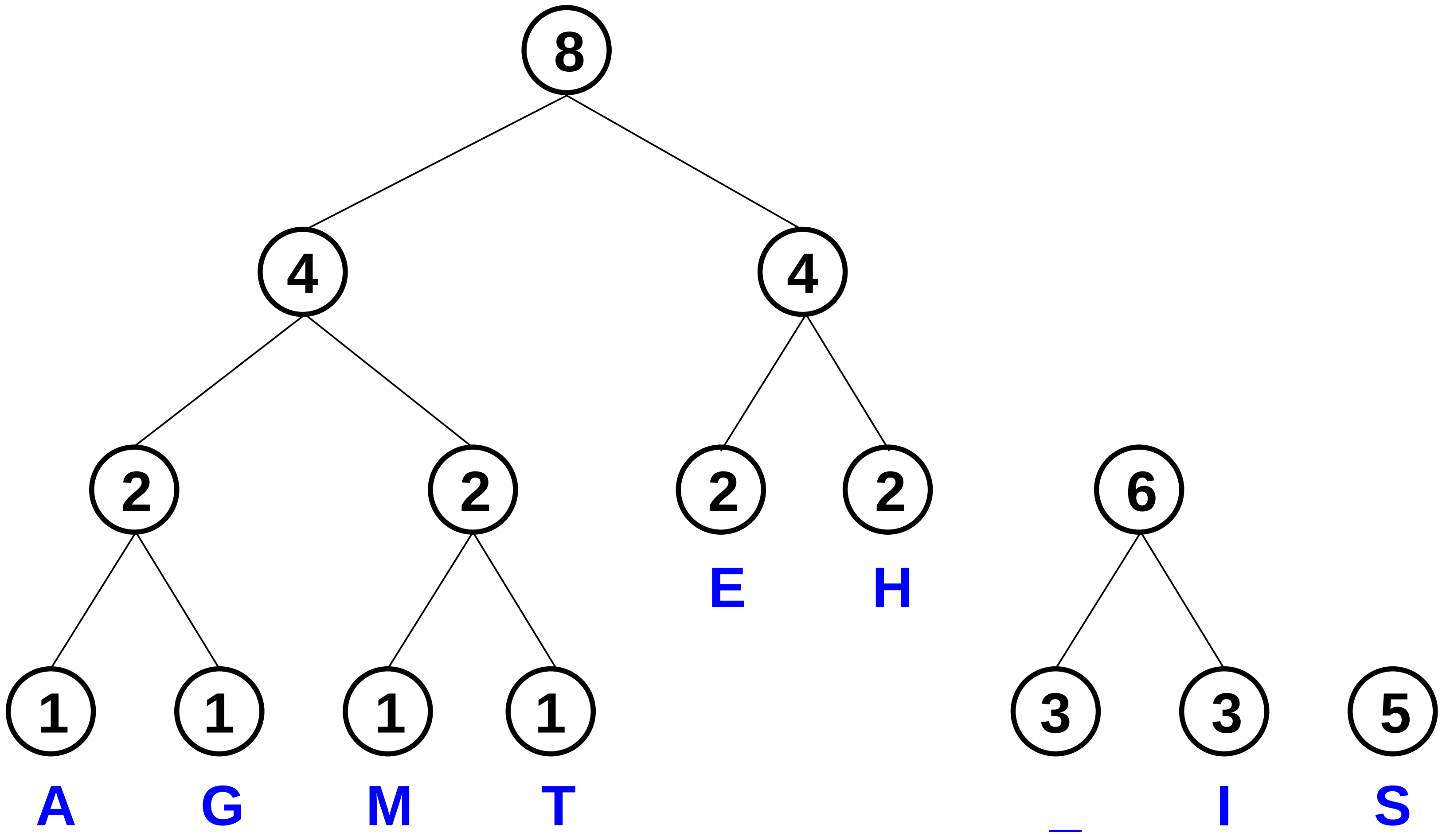
Step 4



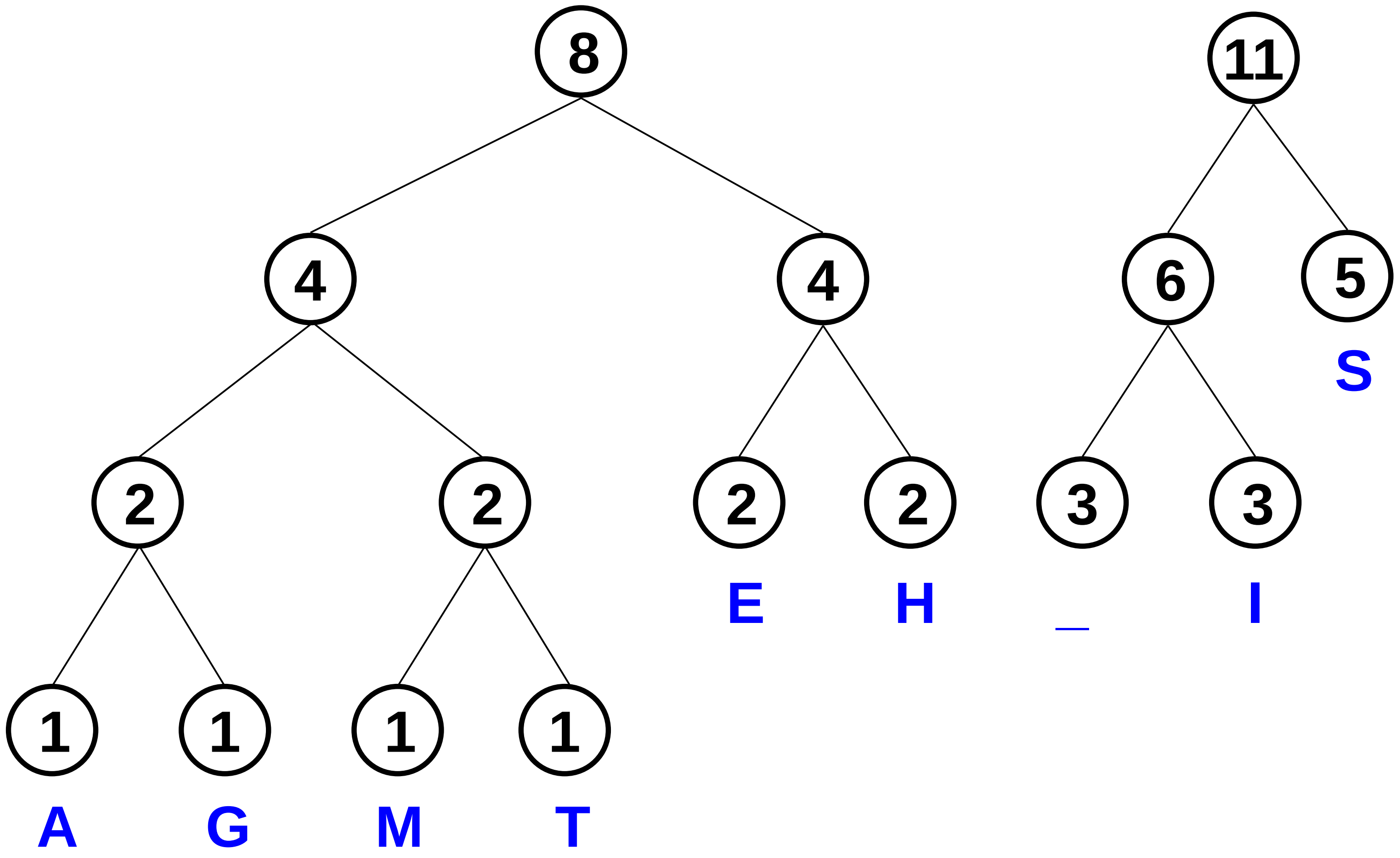
Step 5



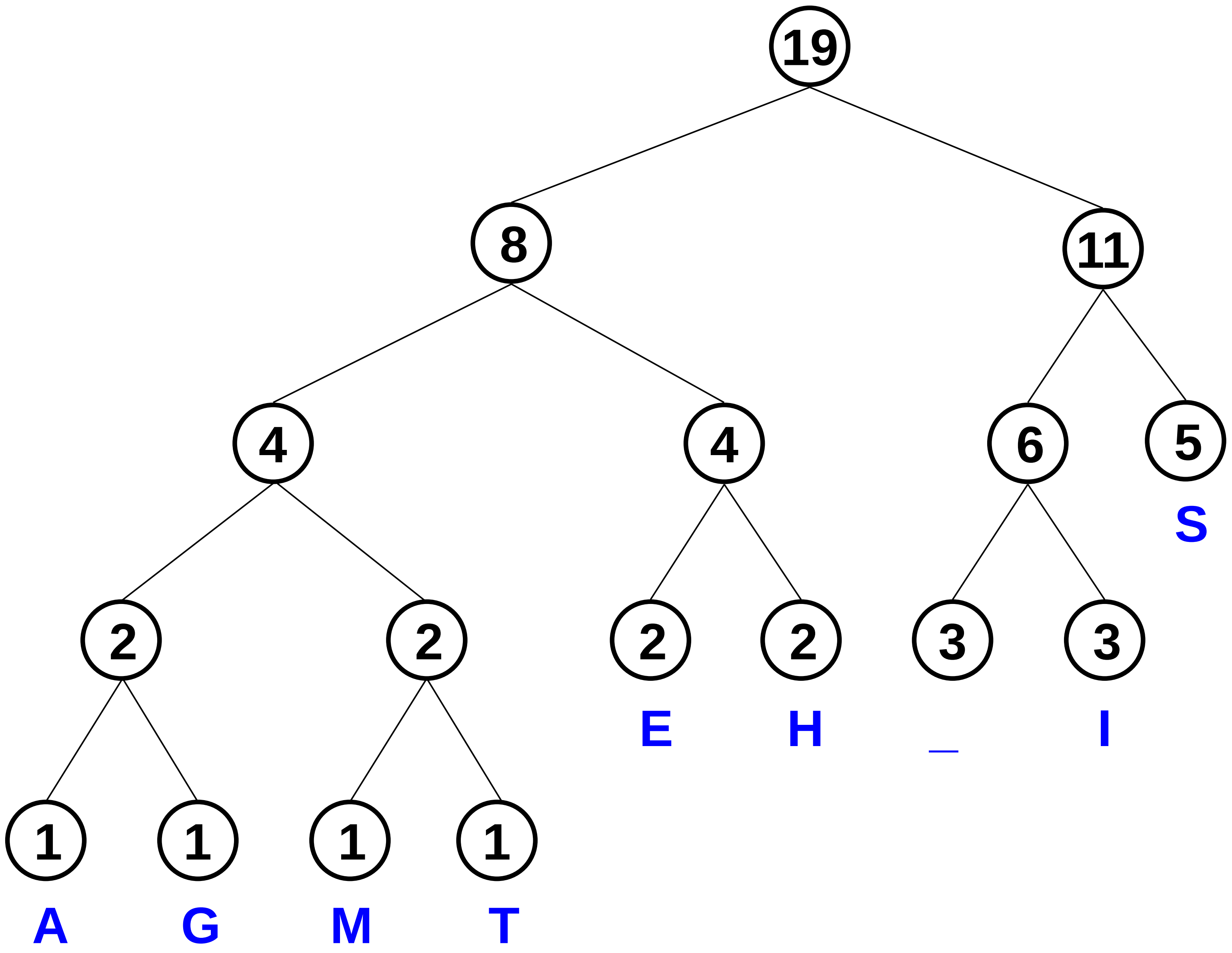
Step 6



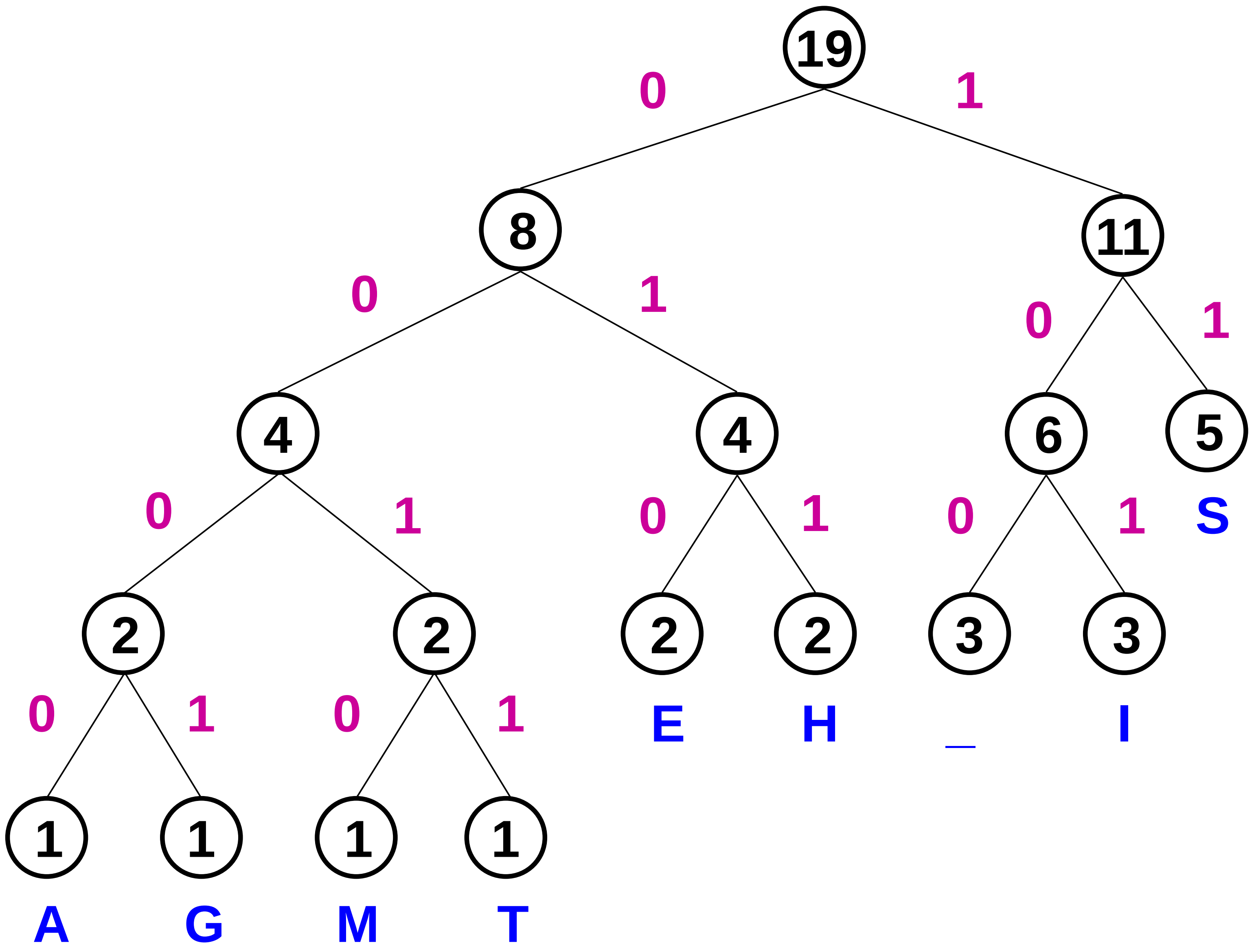
Step 7



Step 8



Label edges



Huffman code & encoded message

This is his message

S	11
E	010
H	011
_	100
I	101
A	0000
G	0001
M	0010
T	0011

00110111011110010111100011101111000010010111100000001010

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
  - 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

Prioritize characters

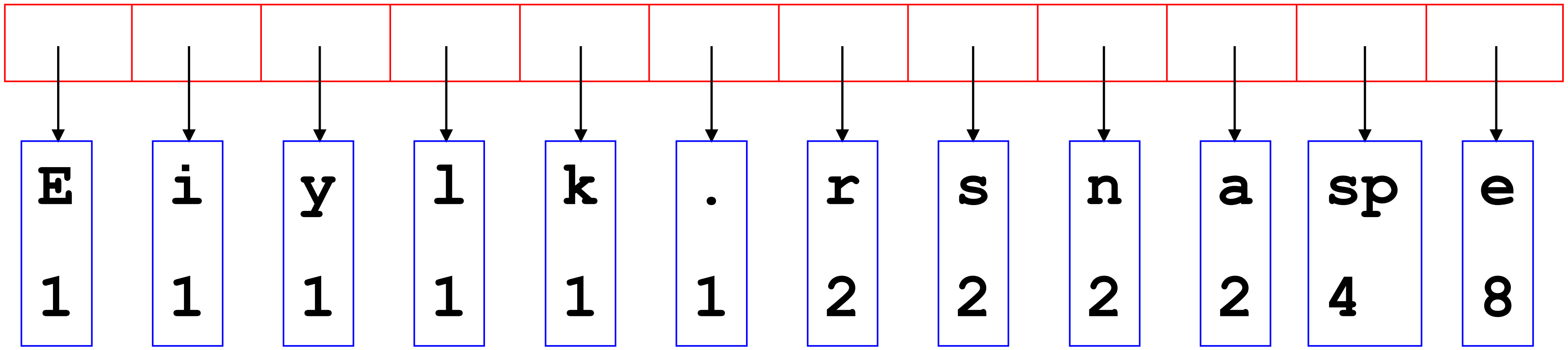
- Uses binary tree nodes

```
public class HuffNode
{
  public char myChar;
  public int myFrequency;
  public HuffNode myLeft, myRight;
}

priorityQueue myQueue;
```

Building a Tree

- The queue after inserting all nodes

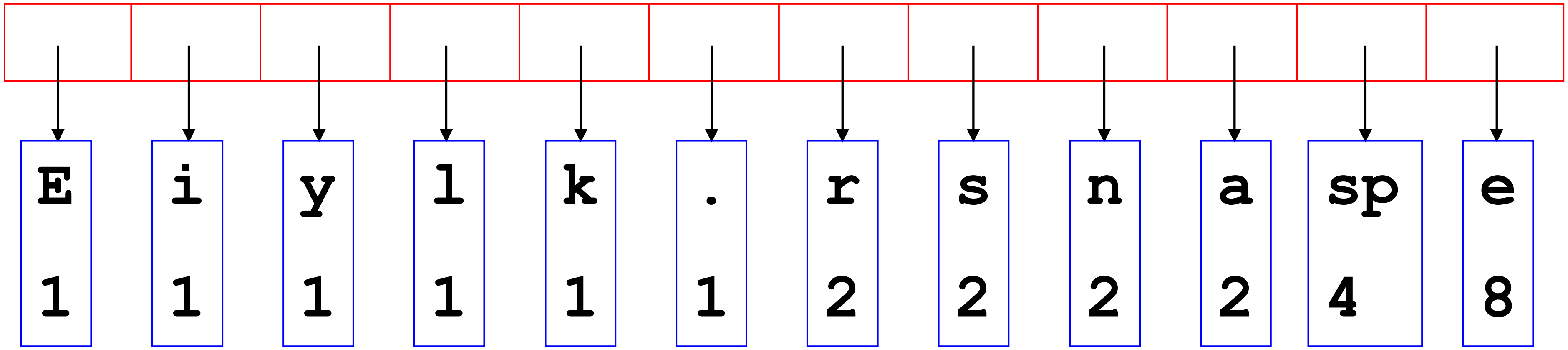


- Null Pointers are not shown

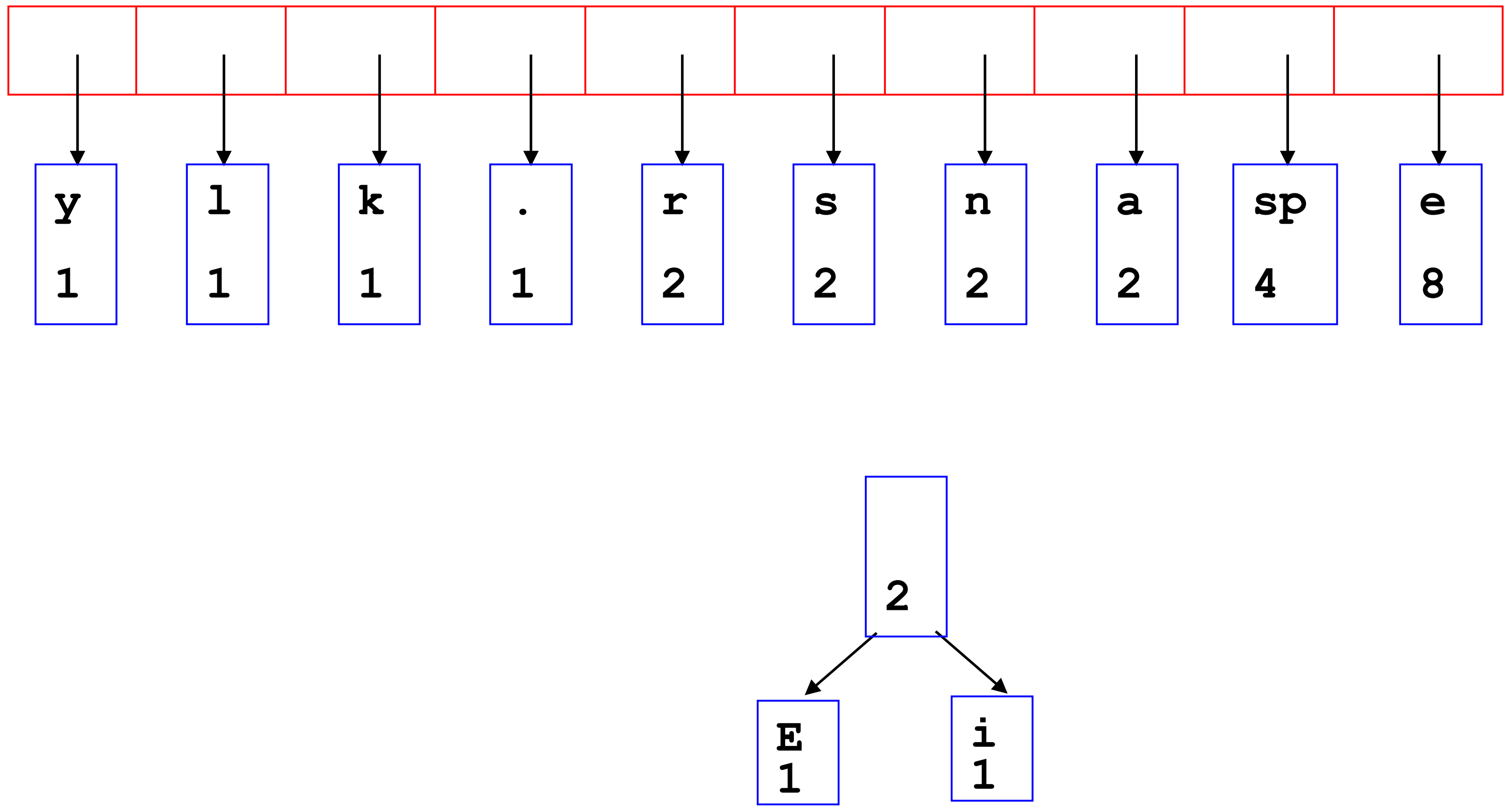
Building a Tree

- 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

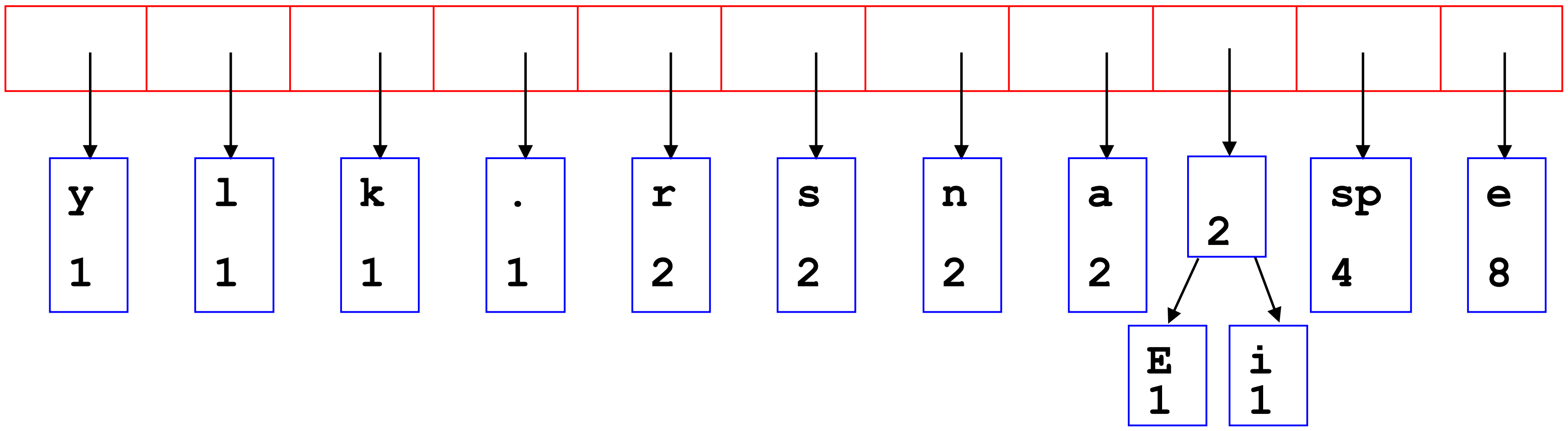
Building a Tree



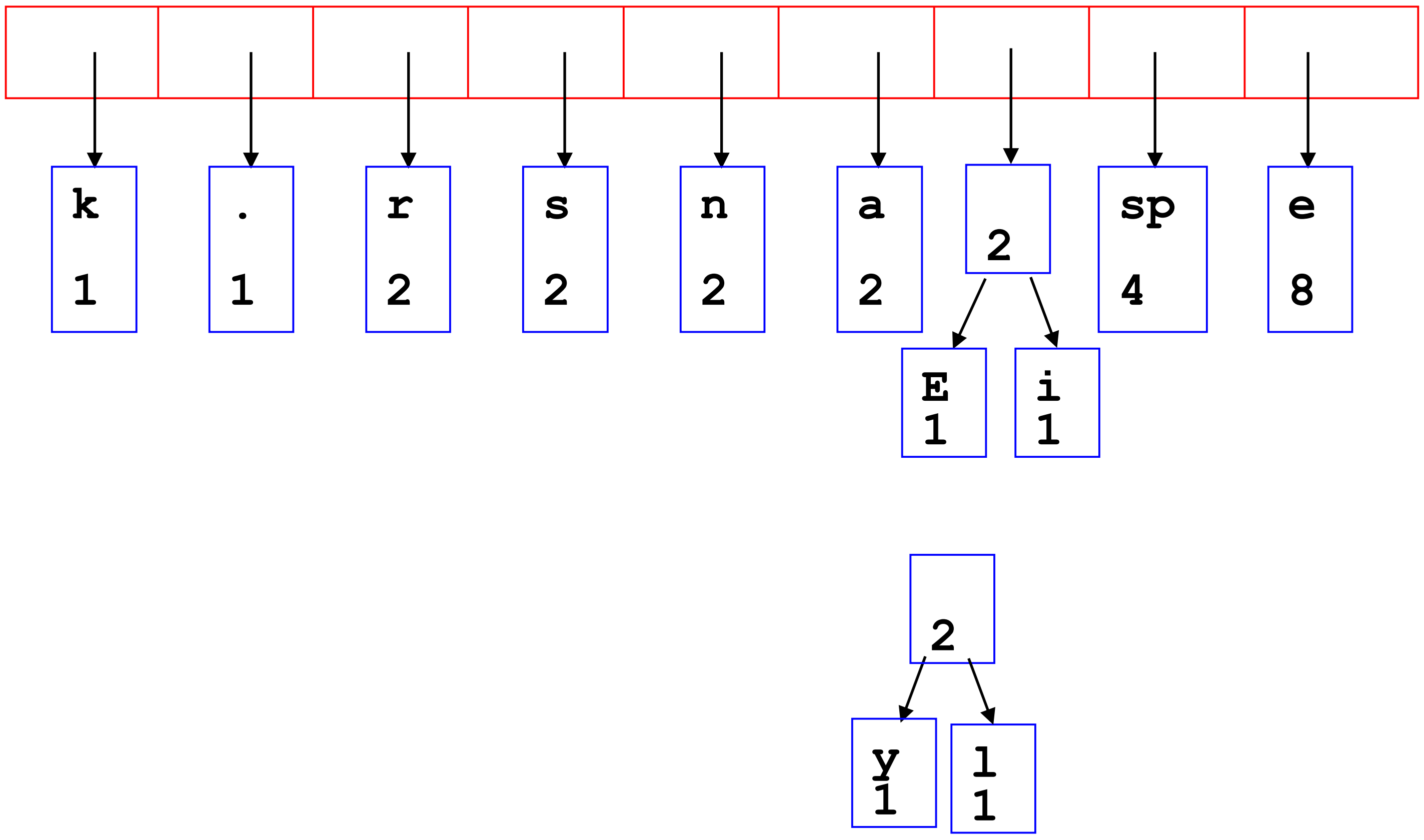
Building a Tree



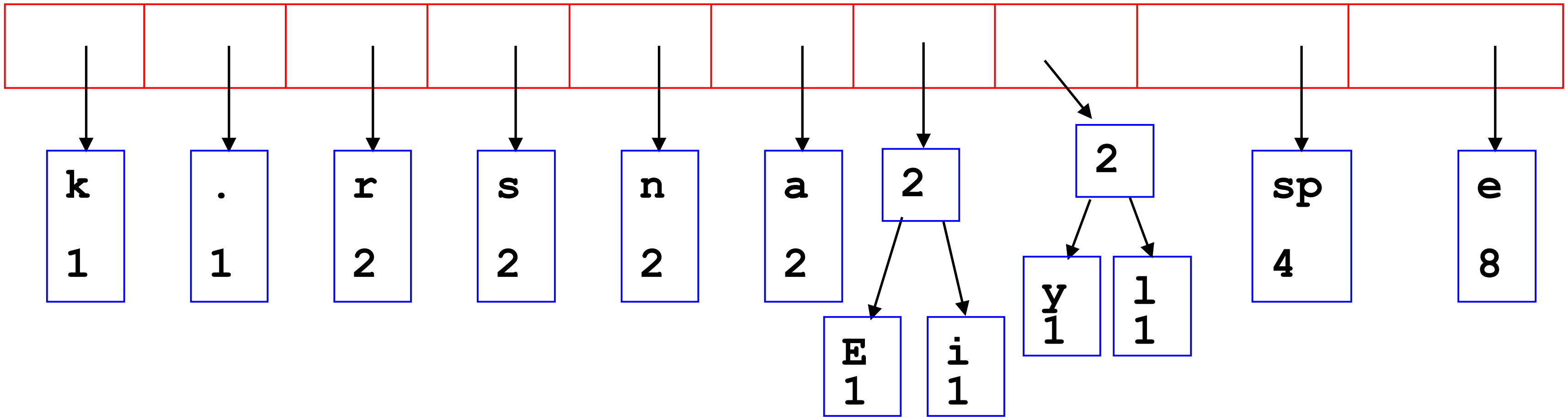
Building a Tree



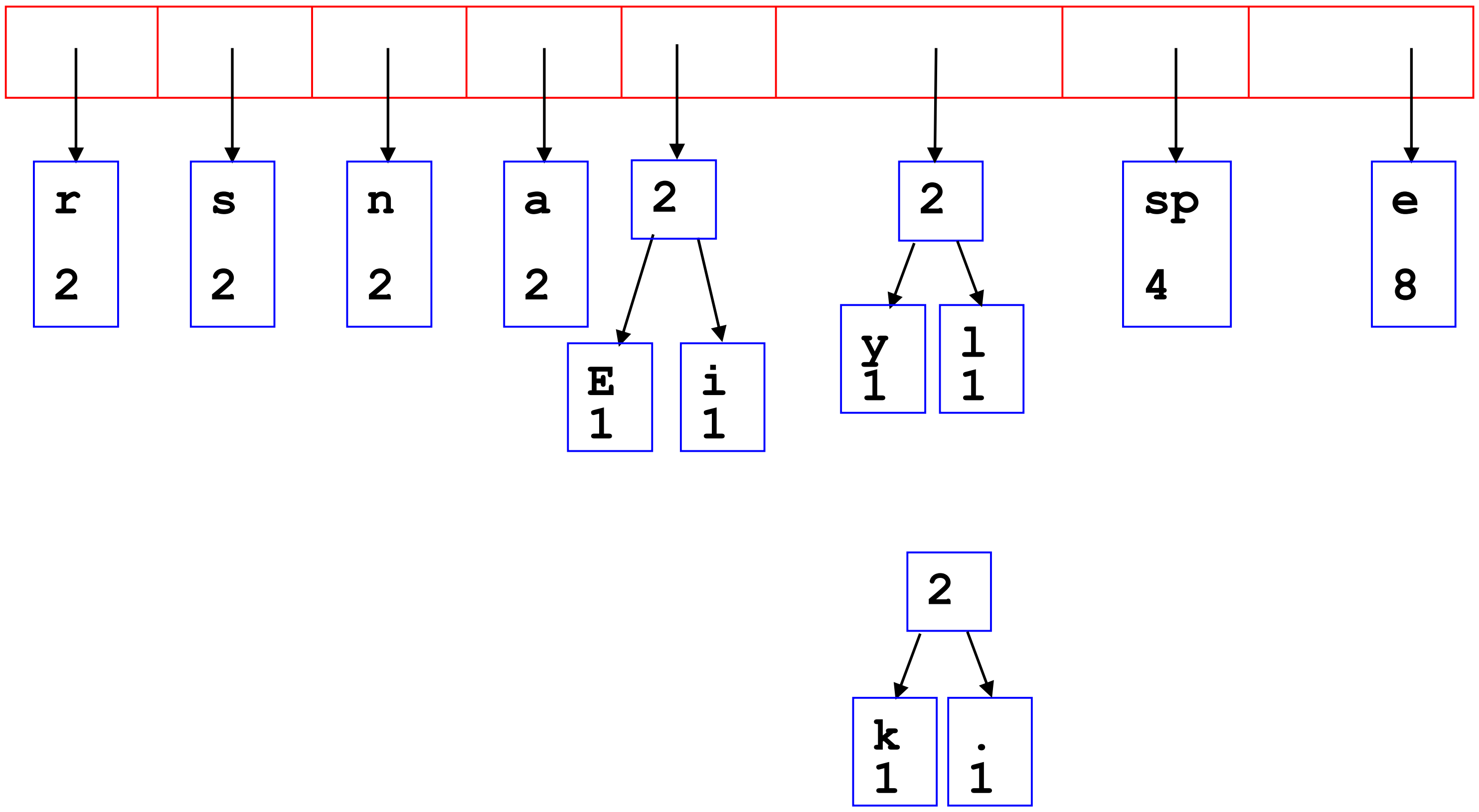
Building a Tree



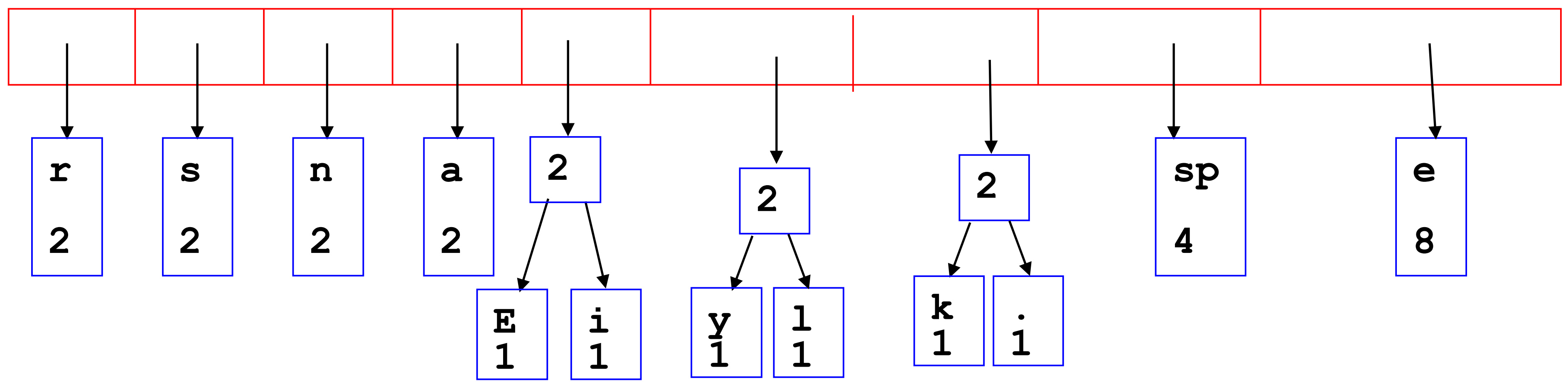
Building a Tree



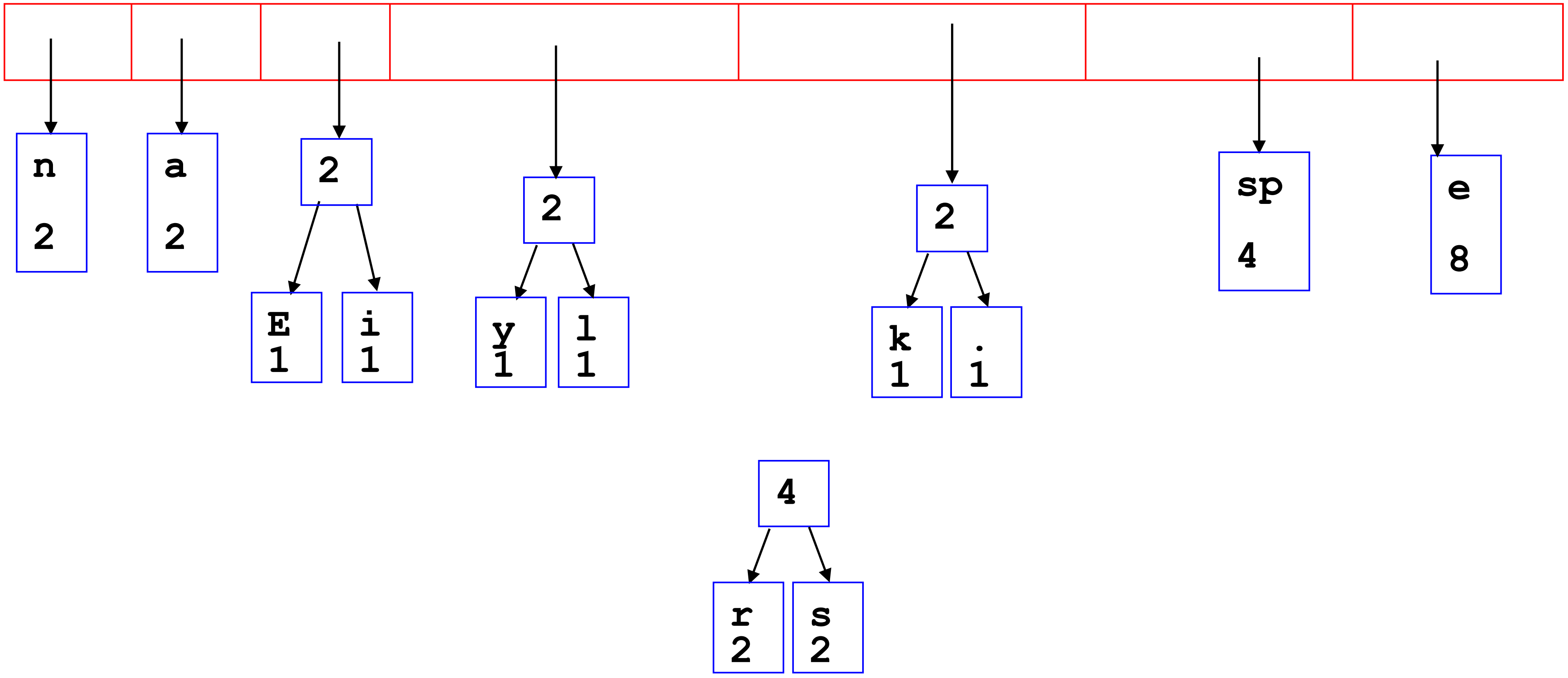
Building a Tree



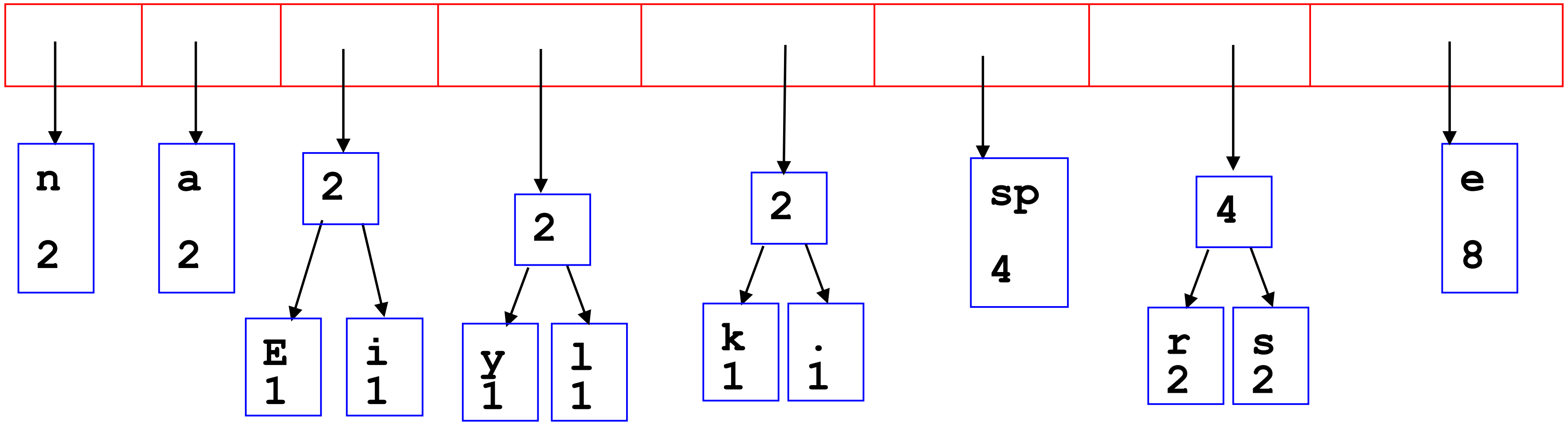
Building a Tree



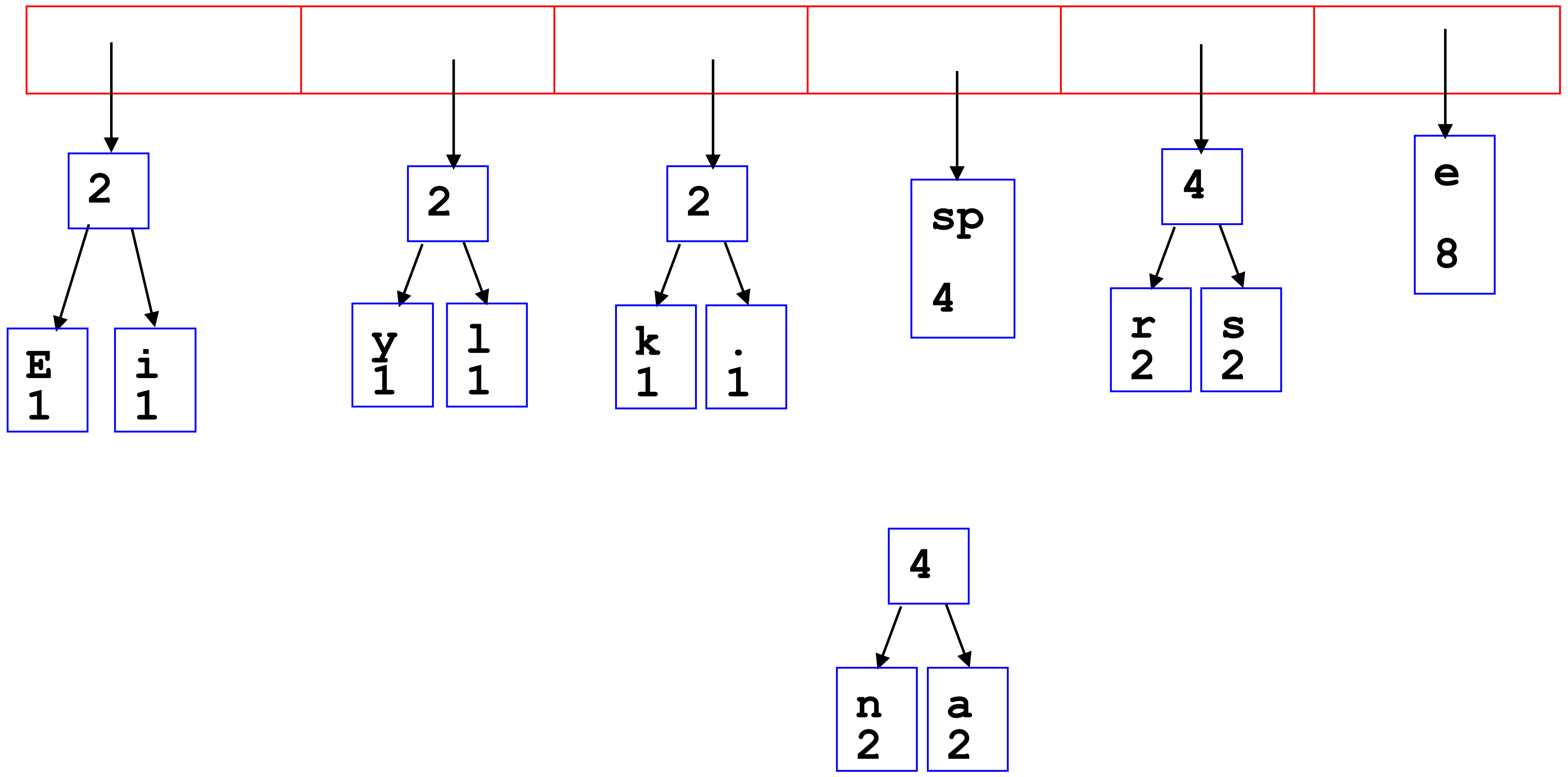
Building a Tree



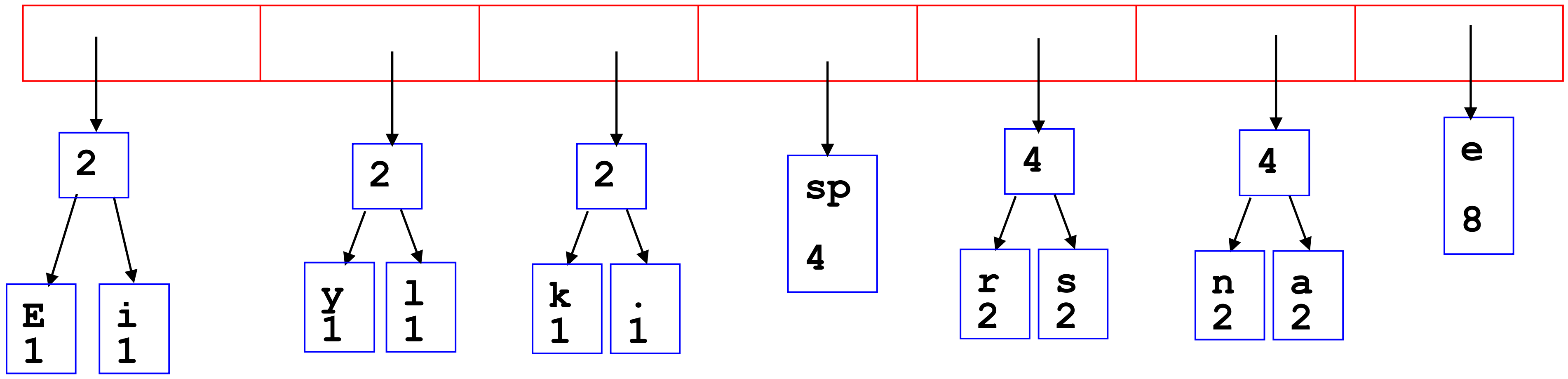
Building a Tree



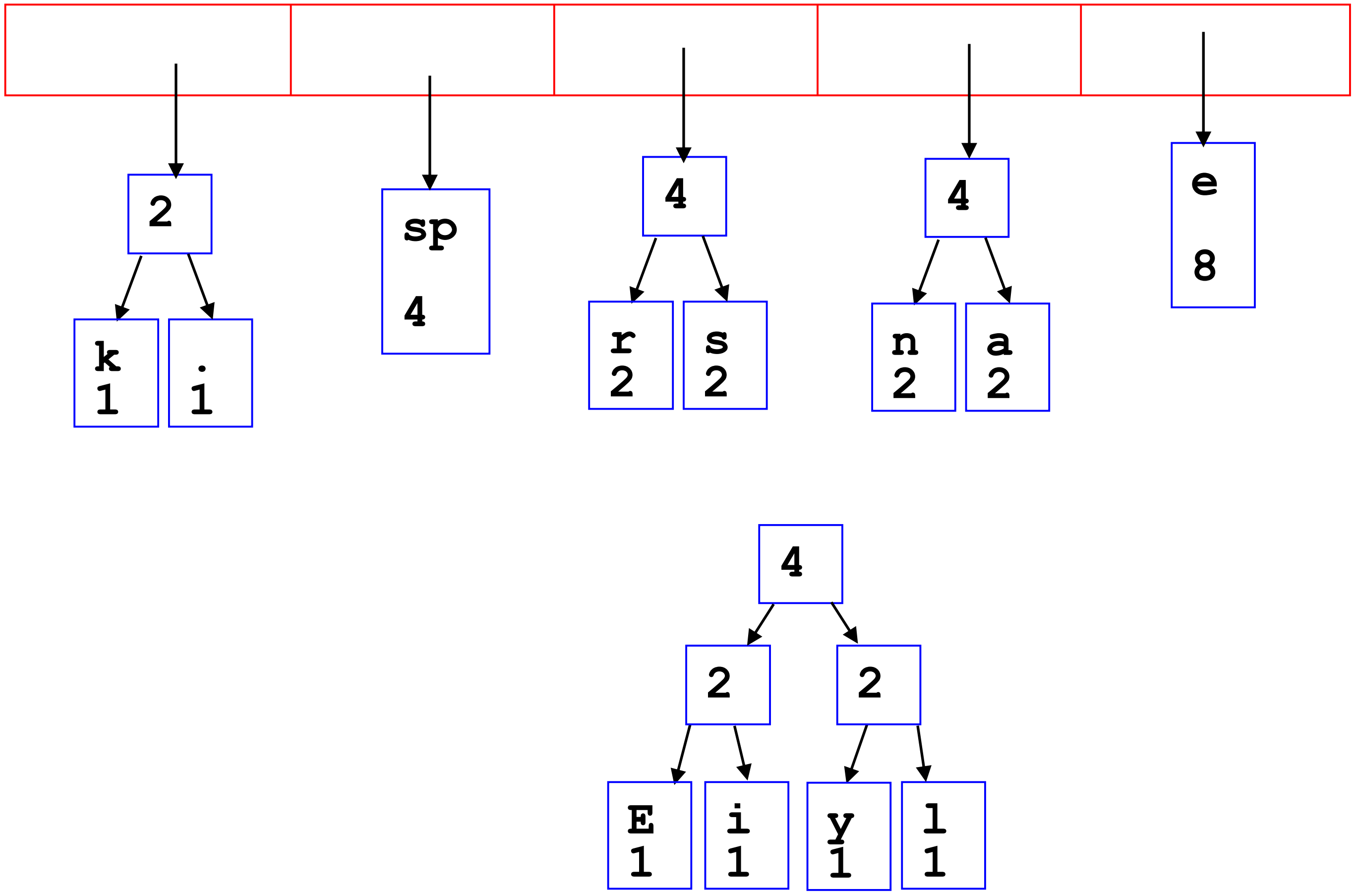
Building a Tree



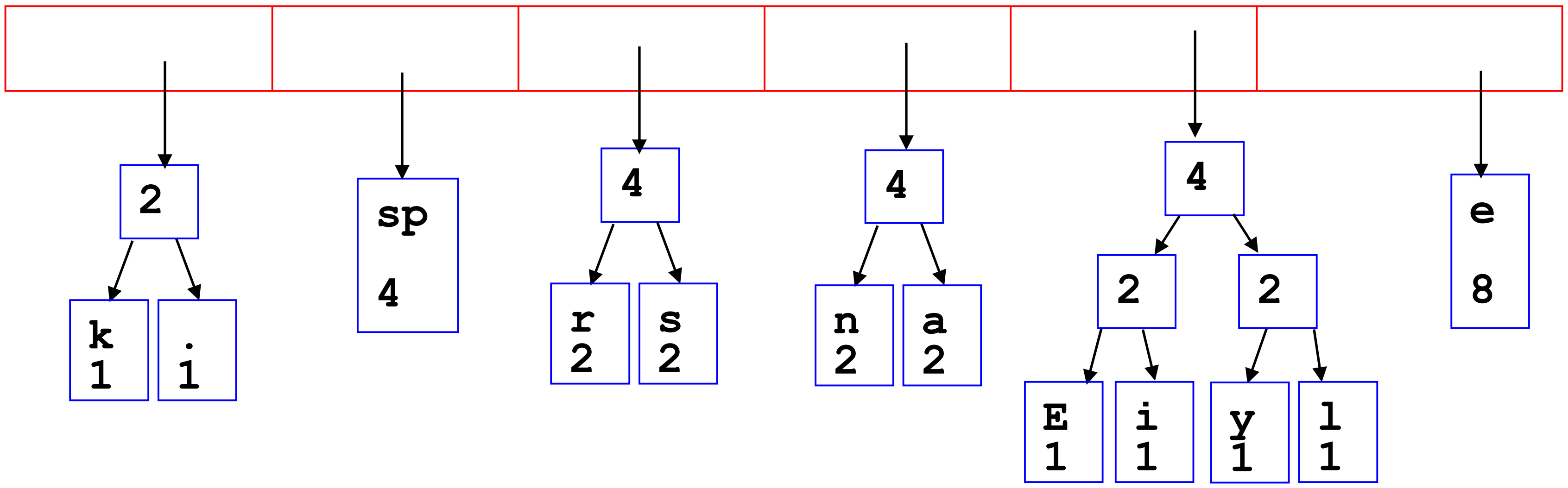
Building a Tree



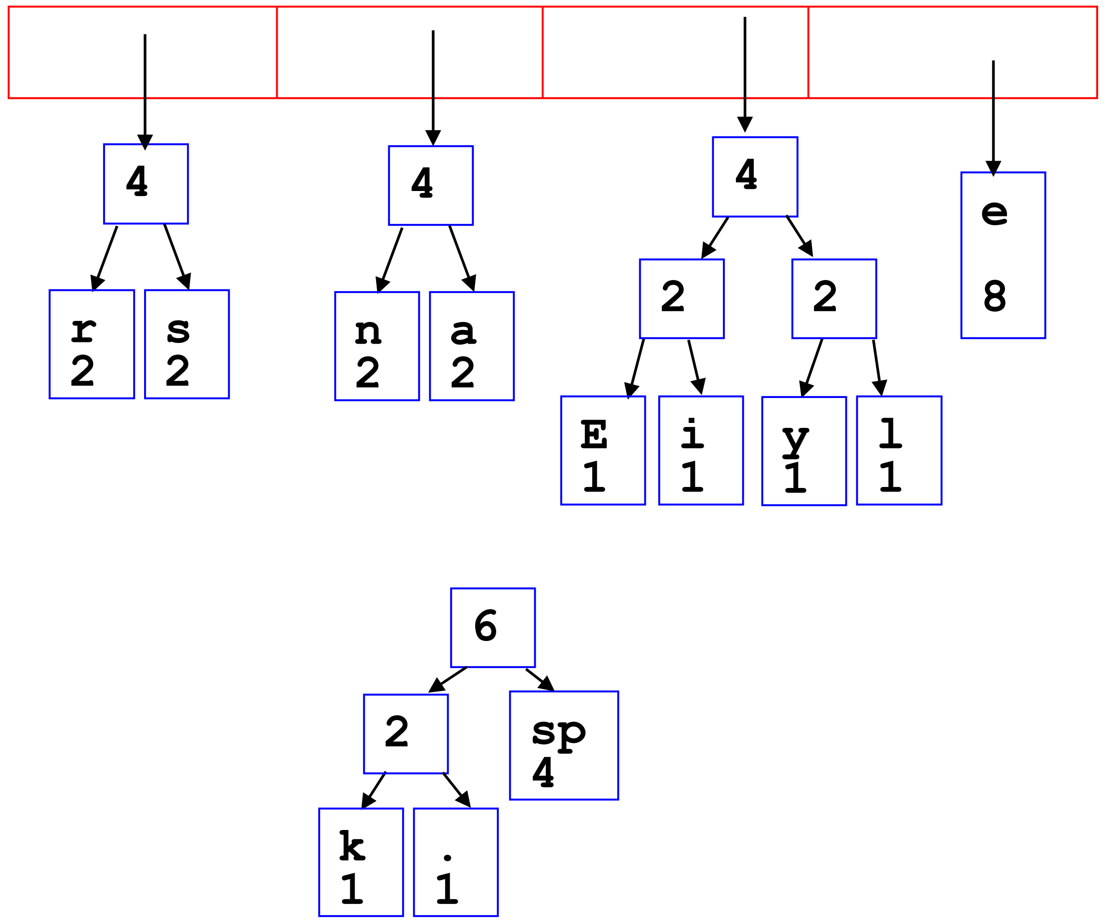
Building a Tree



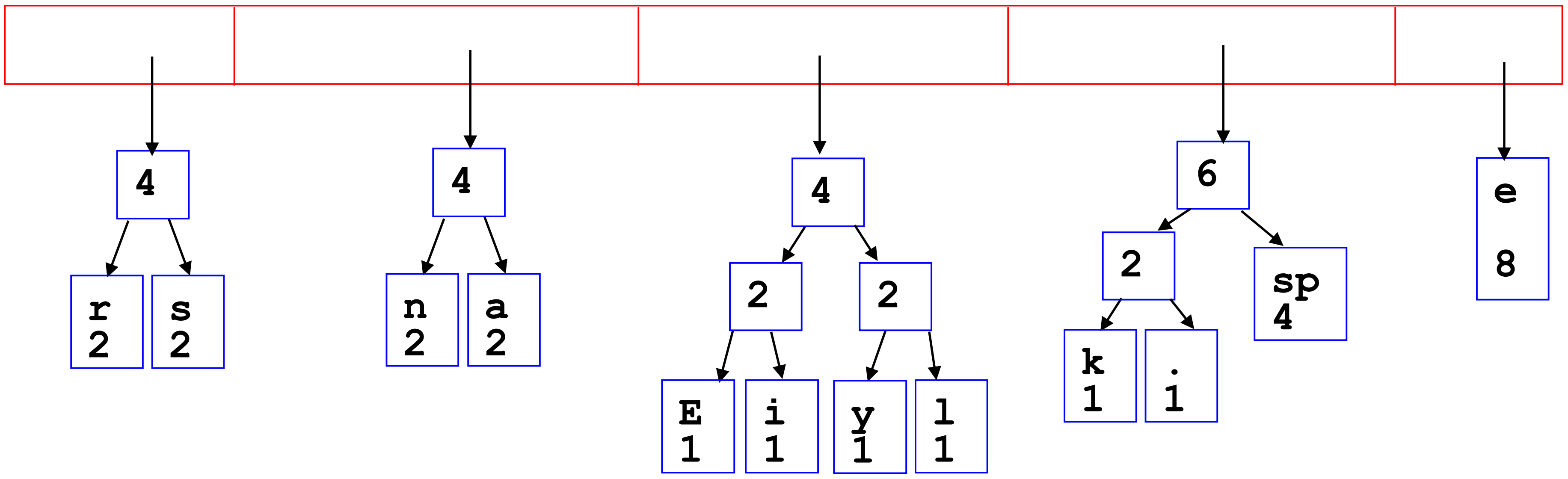
Building a Tree



Building a Tree

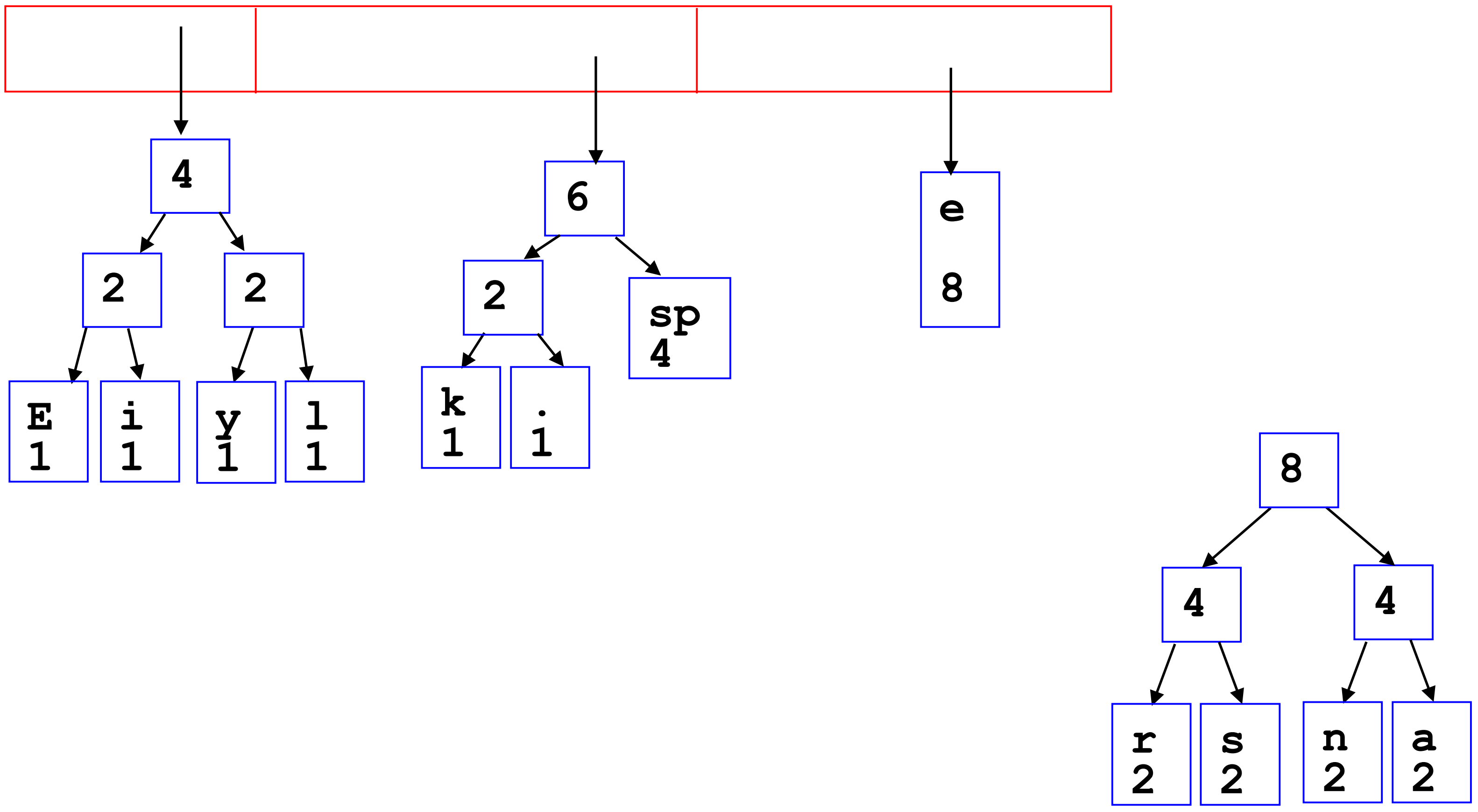


Building a Tree

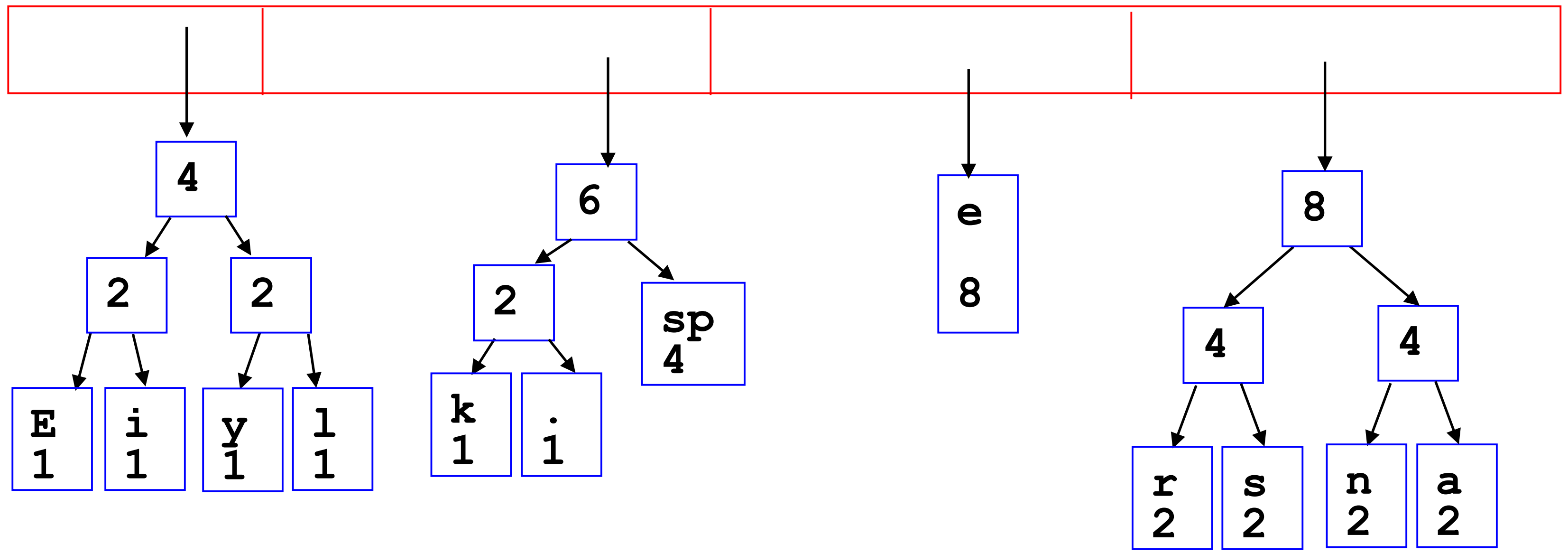


What is happening to the characters with a low number of occurrences?

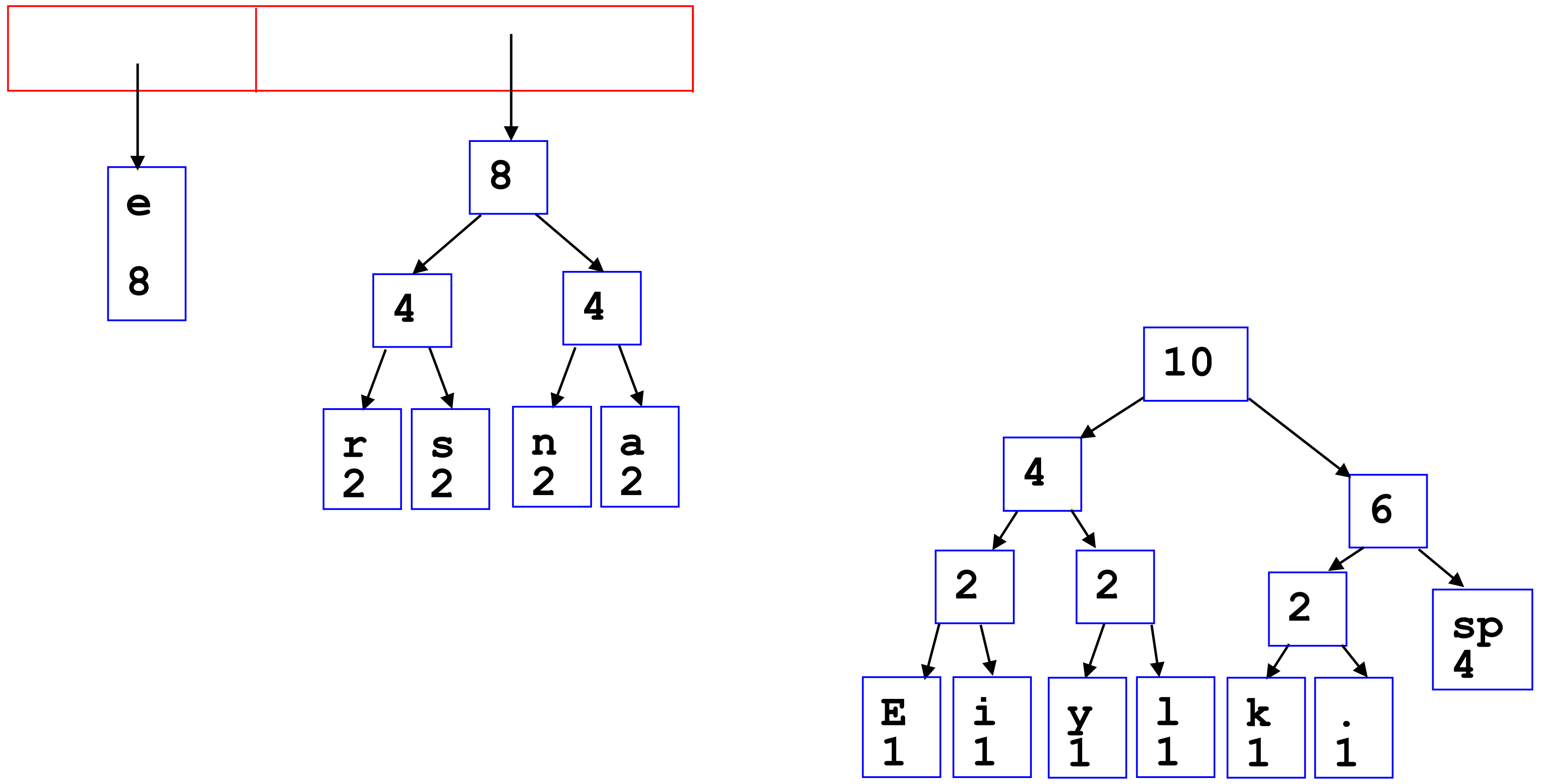
Building a Tree



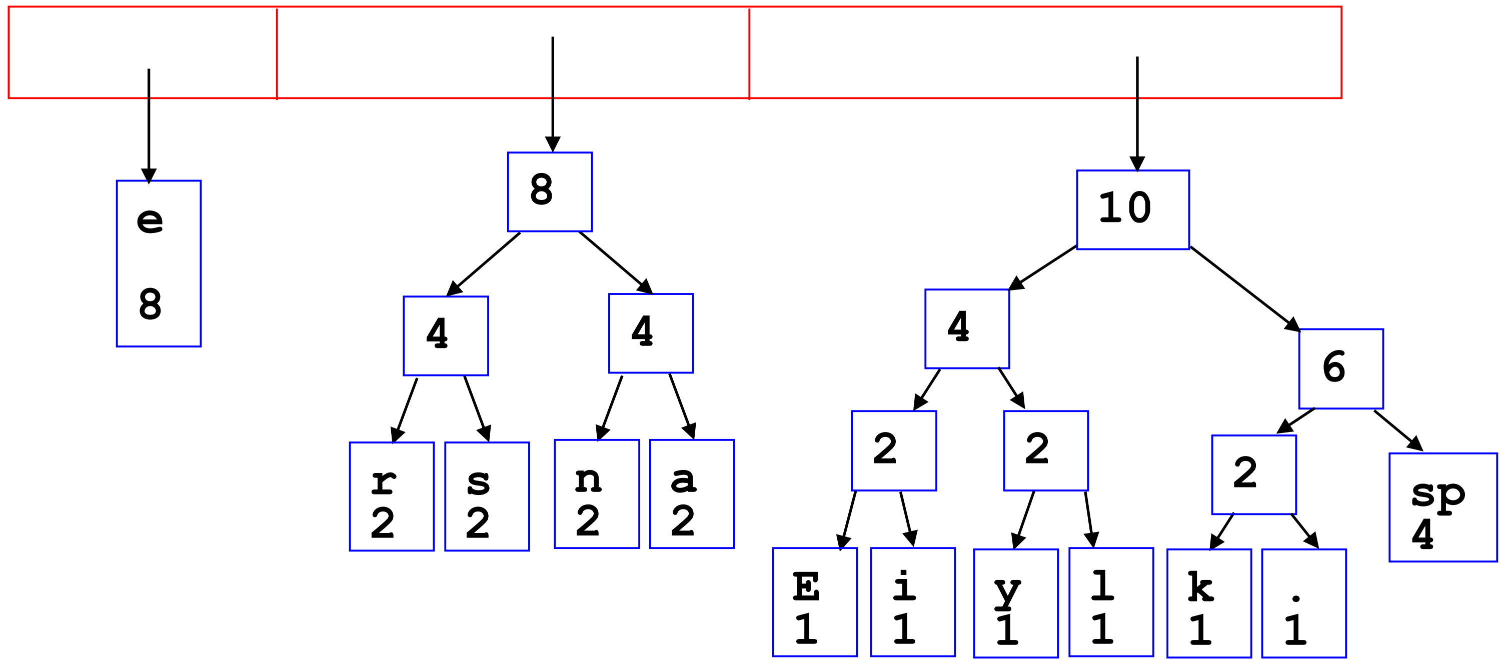
Building a Tree



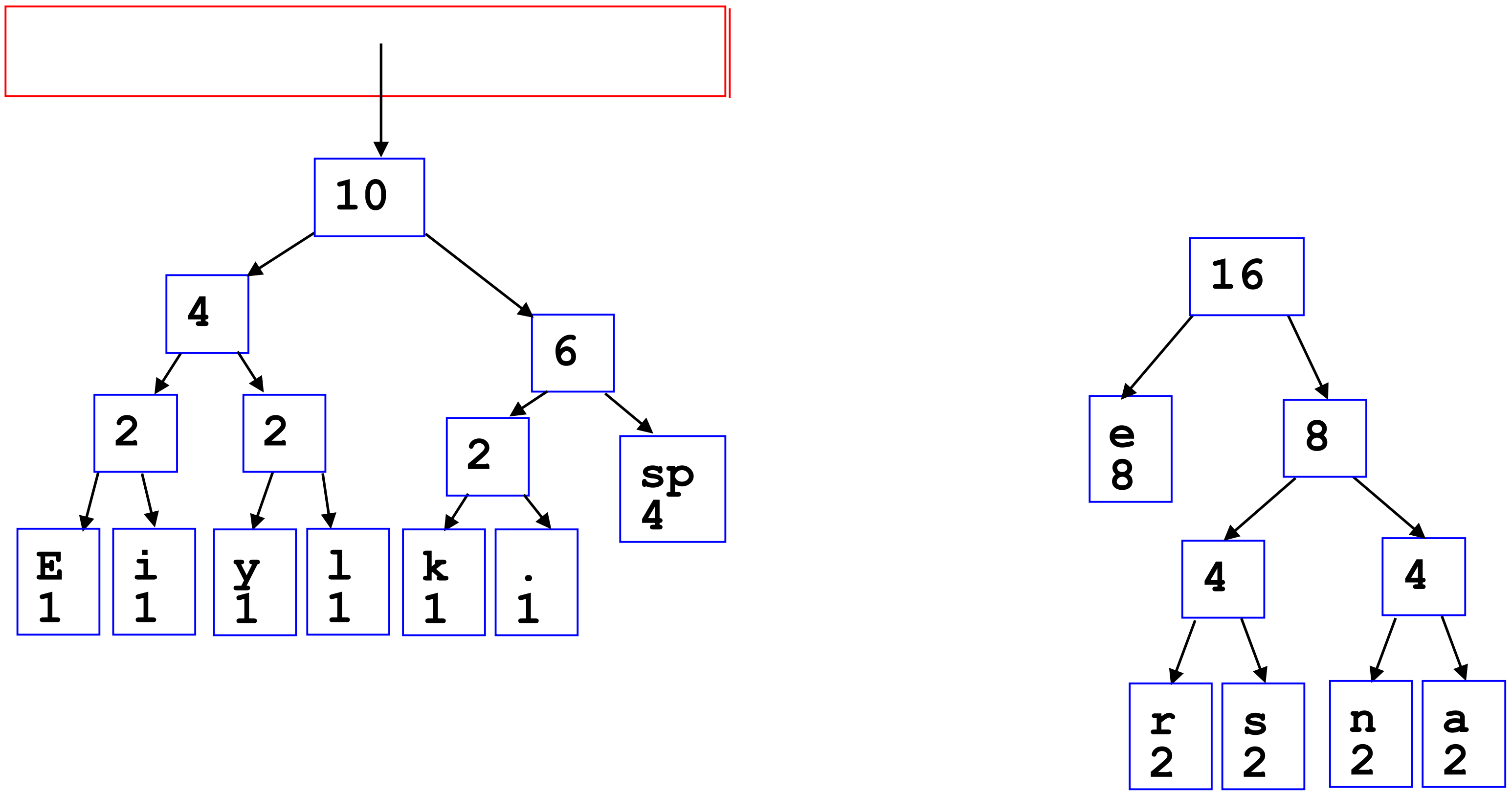
Building a Tree



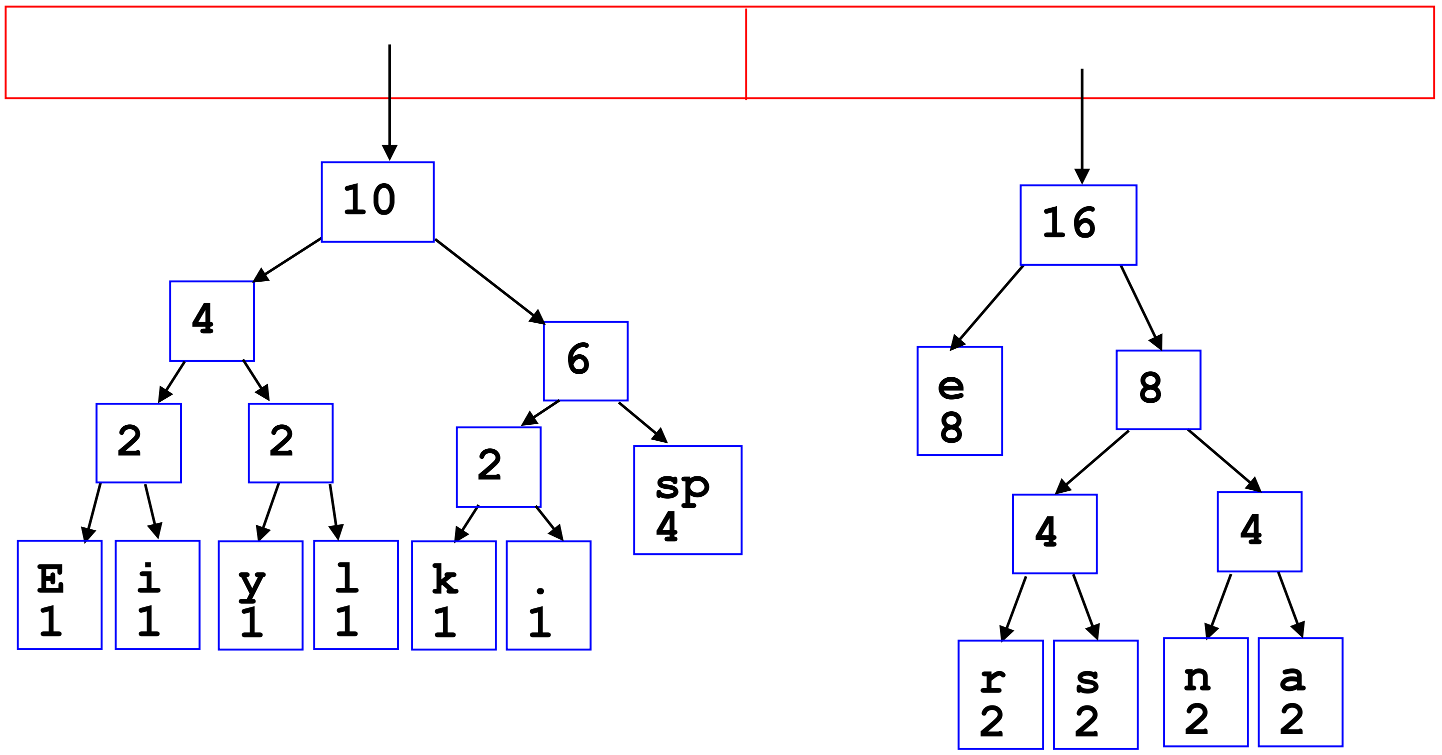
Building a Tree



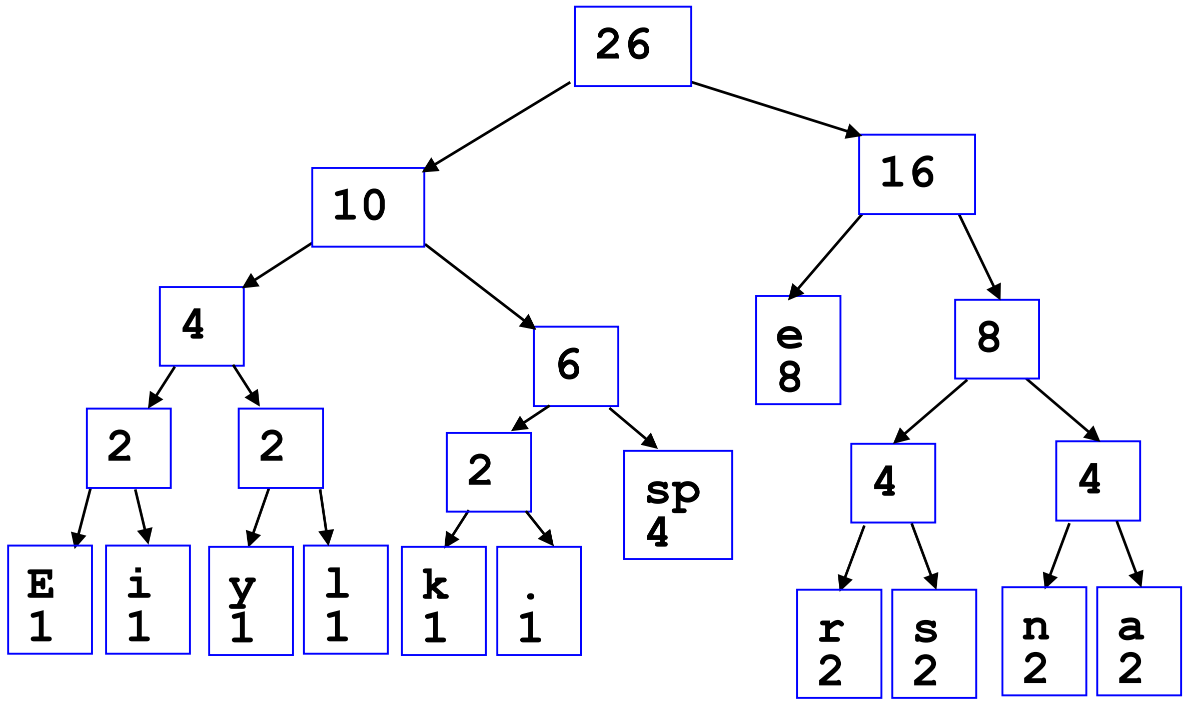
Building a Tree



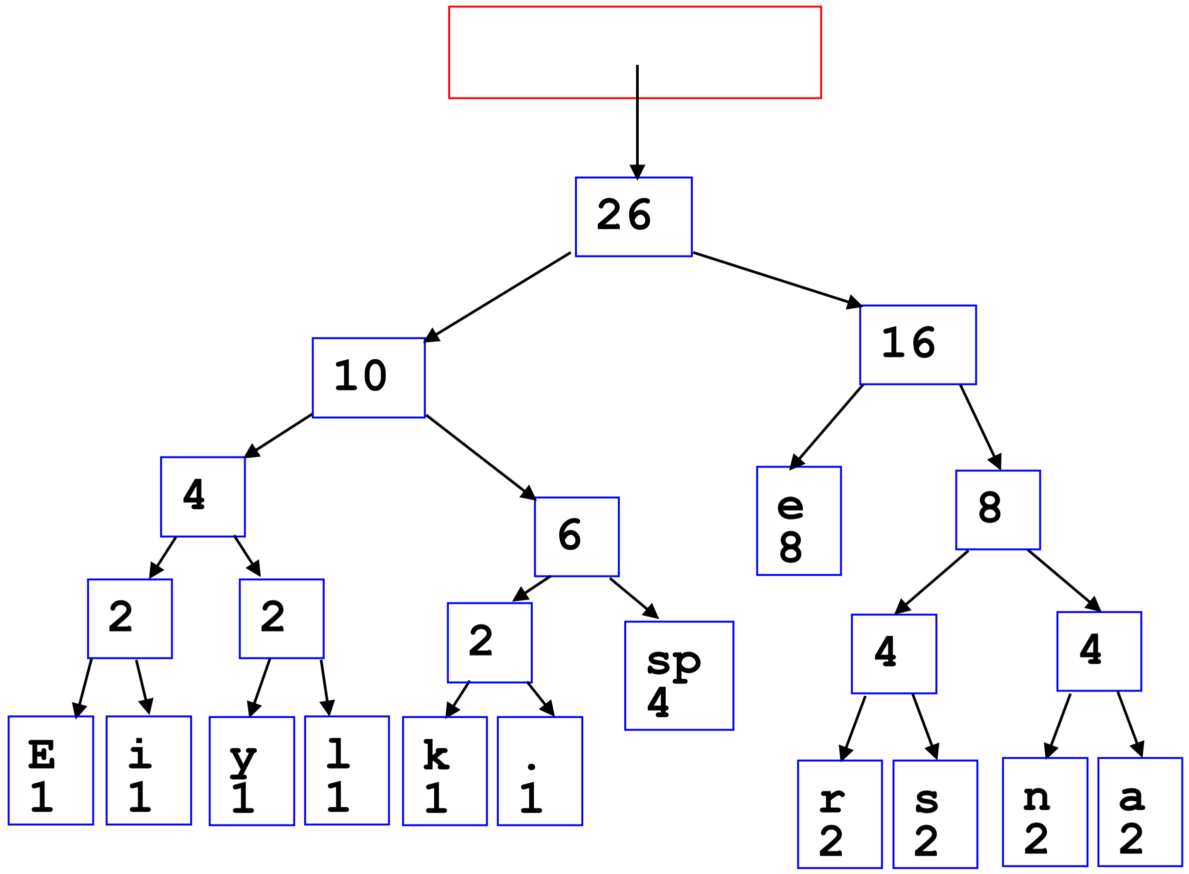
Building a Tree



Building a Tree



Building a Tree



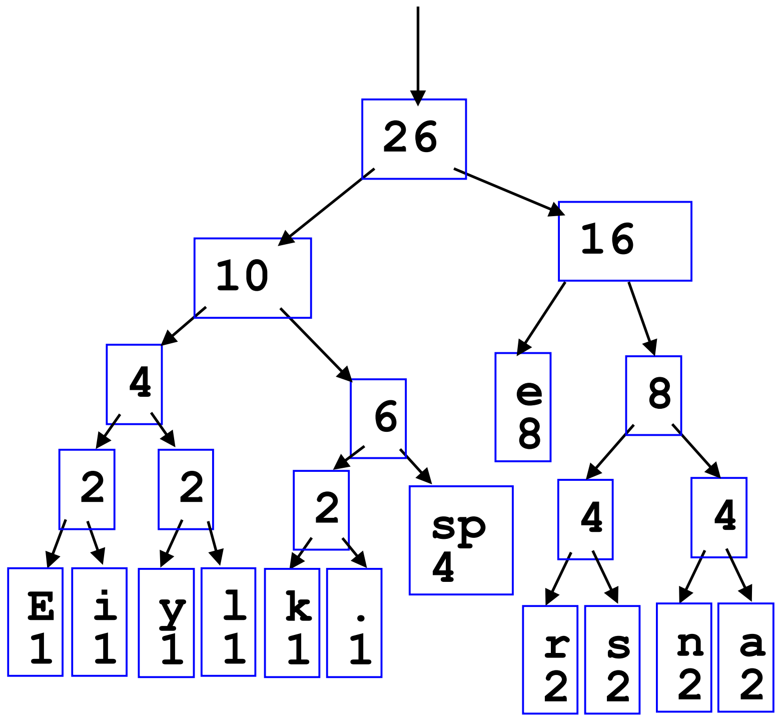
- After enqueueing this node there is only one node left in priority queue.

Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

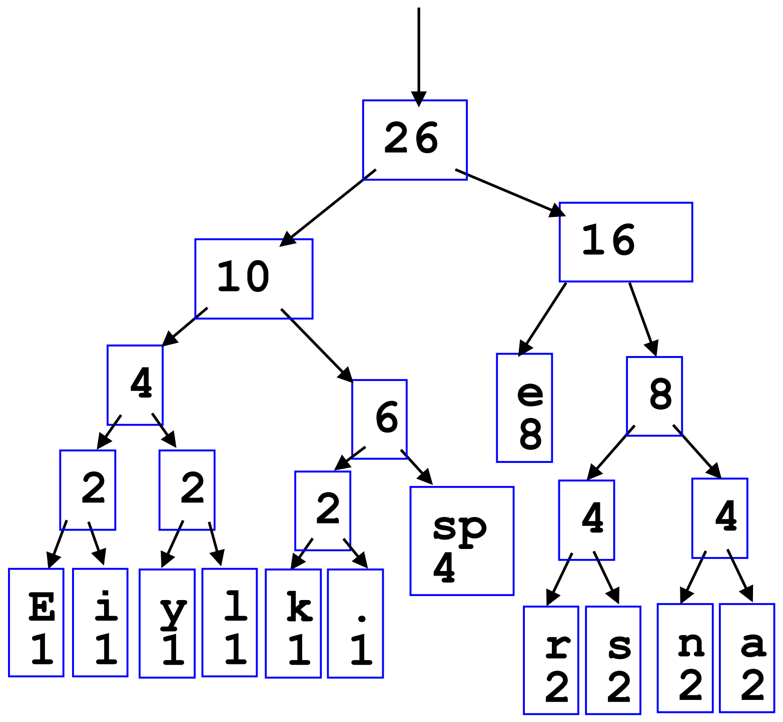
Frequency of root node should equal number of characters in text.



Eerie eyes seen near lake. 26 characters

Encoding the File
Traverse Tree for Codes

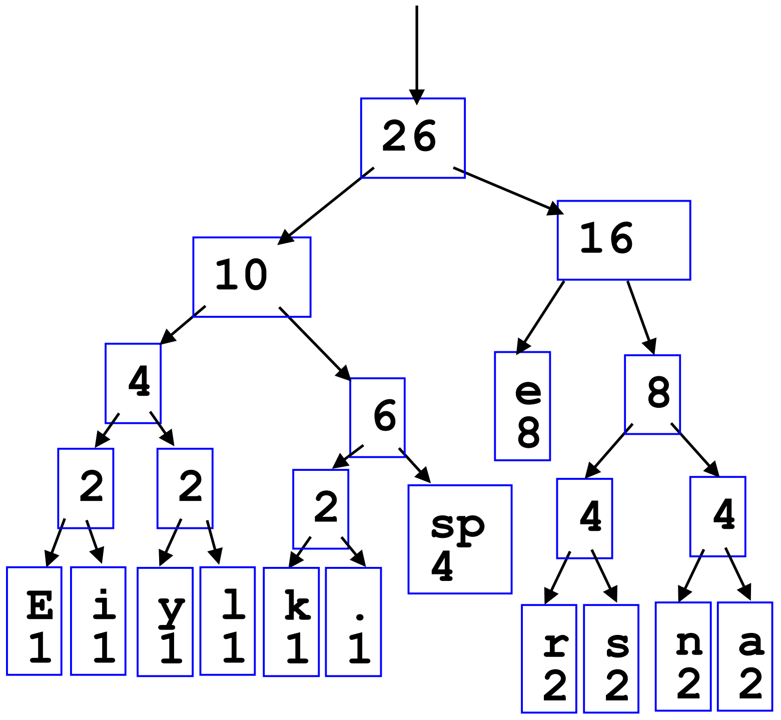
- Perform a traversal of the tree to obtain new code words
- Going left is a 0 going right is a 1
- code word is only completed when a leaf node is reached



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

```
0000101100000110011
1000101011011010011
1110101111110001100
111111010010010 1
```

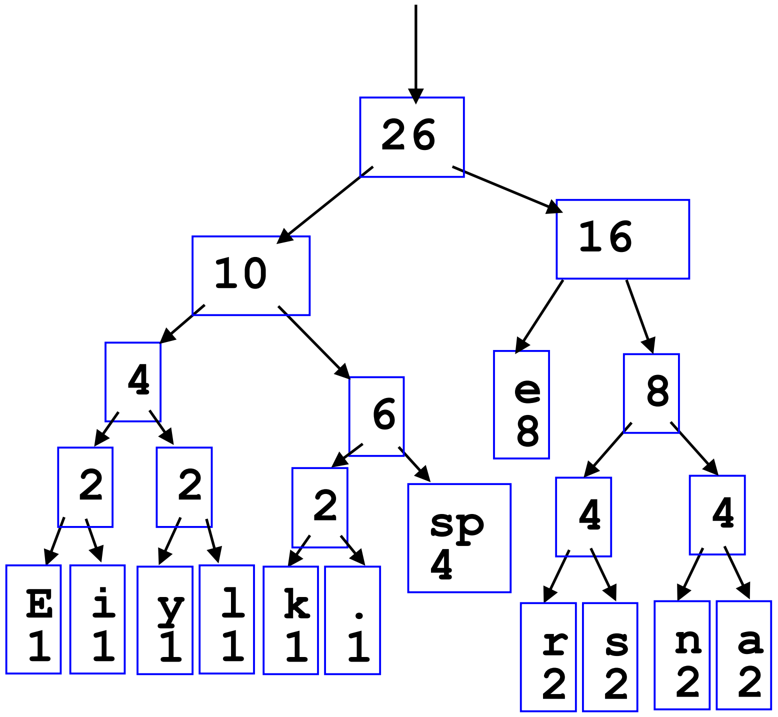
Decoding the File

- 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 ⇒ go left
- 1 ⇒ go right

```
1010001101111101111
0111111000001101 01
```



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
- An application that uses several data structures