Data Structures and Algorithms

Lecture 12 – Huffman Coding Trees

Forests

Miao Zhang

Lecture Objectives

- **To learn how to use a Huffman tree to encode characters using fewer bytes than ASCII or Unicode, resulting in smaller files and reduced storage requirements**

Huffman Coding

- Huffman codes can be used to compress information
  - Like WinZip – although WinZip doesn't use the Huffman algorithm
  - JPEGs do use Huffman as part of their compression process
- The basic idea is that instead of storing each character in a file as an 8-bit ASCII value, we will store the more frequently occurring characters using fewer bits and less frequently occurring characters using more bits
  - On average this should decrease the filesize (usually ½)

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 (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 (1)

- 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 (1)

Eerie eyes seen near lake.

- **What characters are present?**

Eeri space ysnarlk.

Building a Tree
Scan the original text (1)

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		

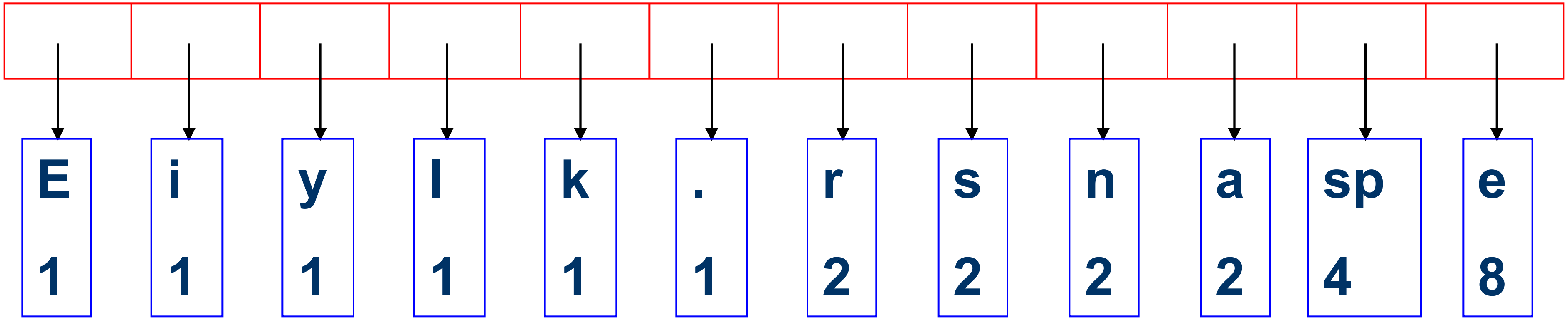
Building a Tree
Prioritize characters (2)

- 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

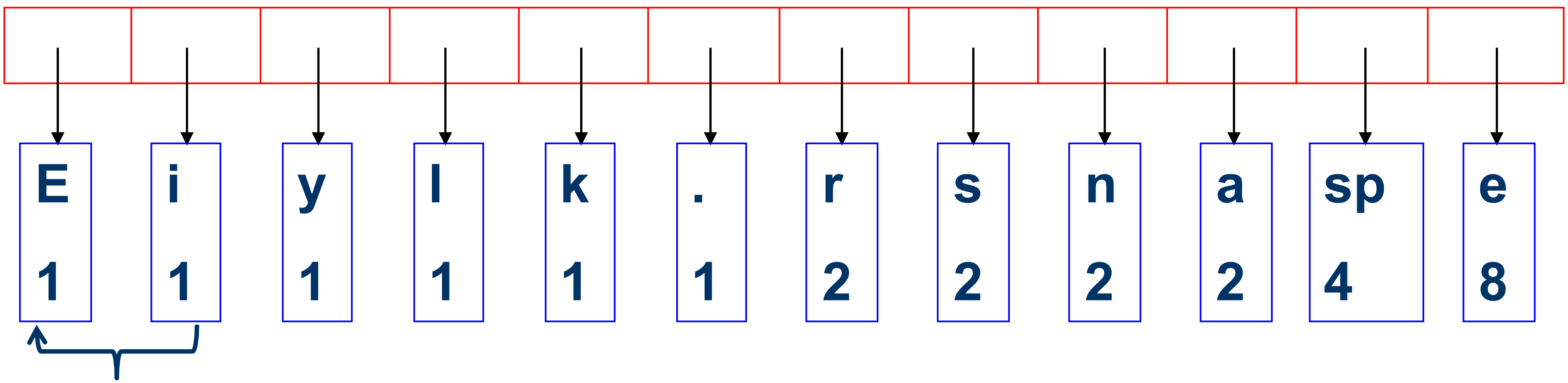


- Null Pointers are not shown

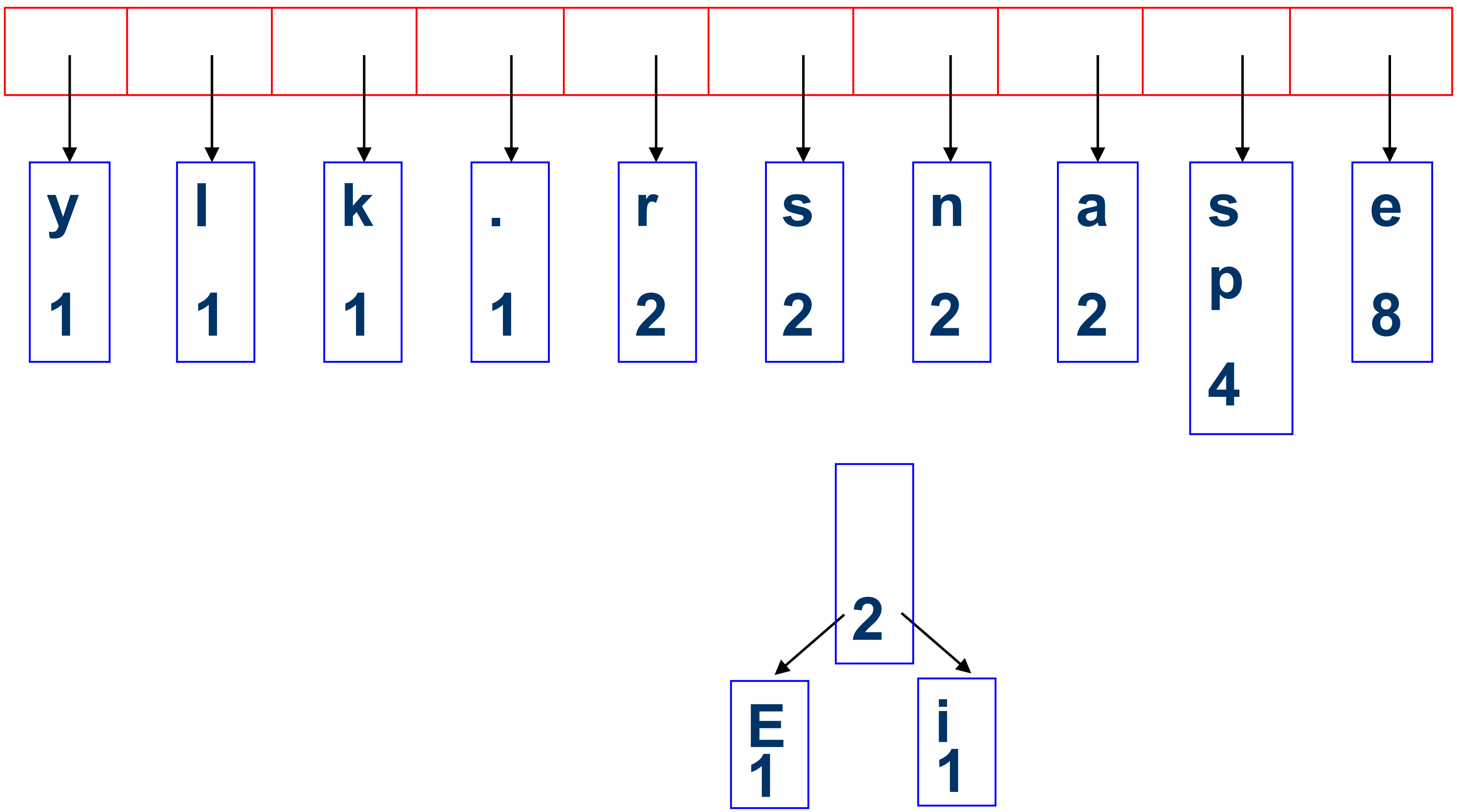
Building a Tree

- 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

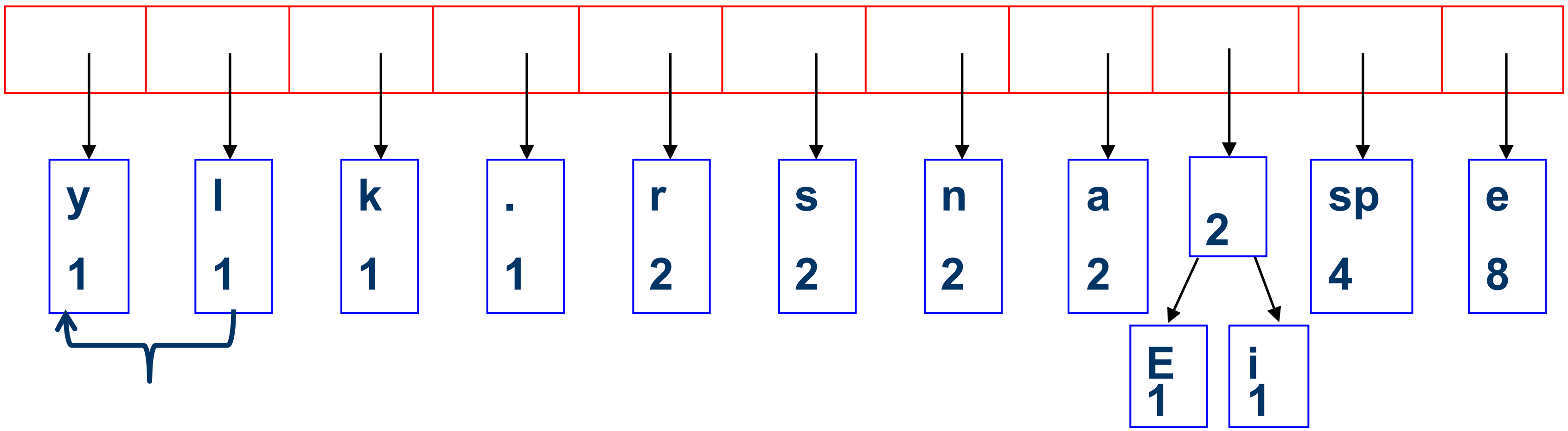
Building a Tree



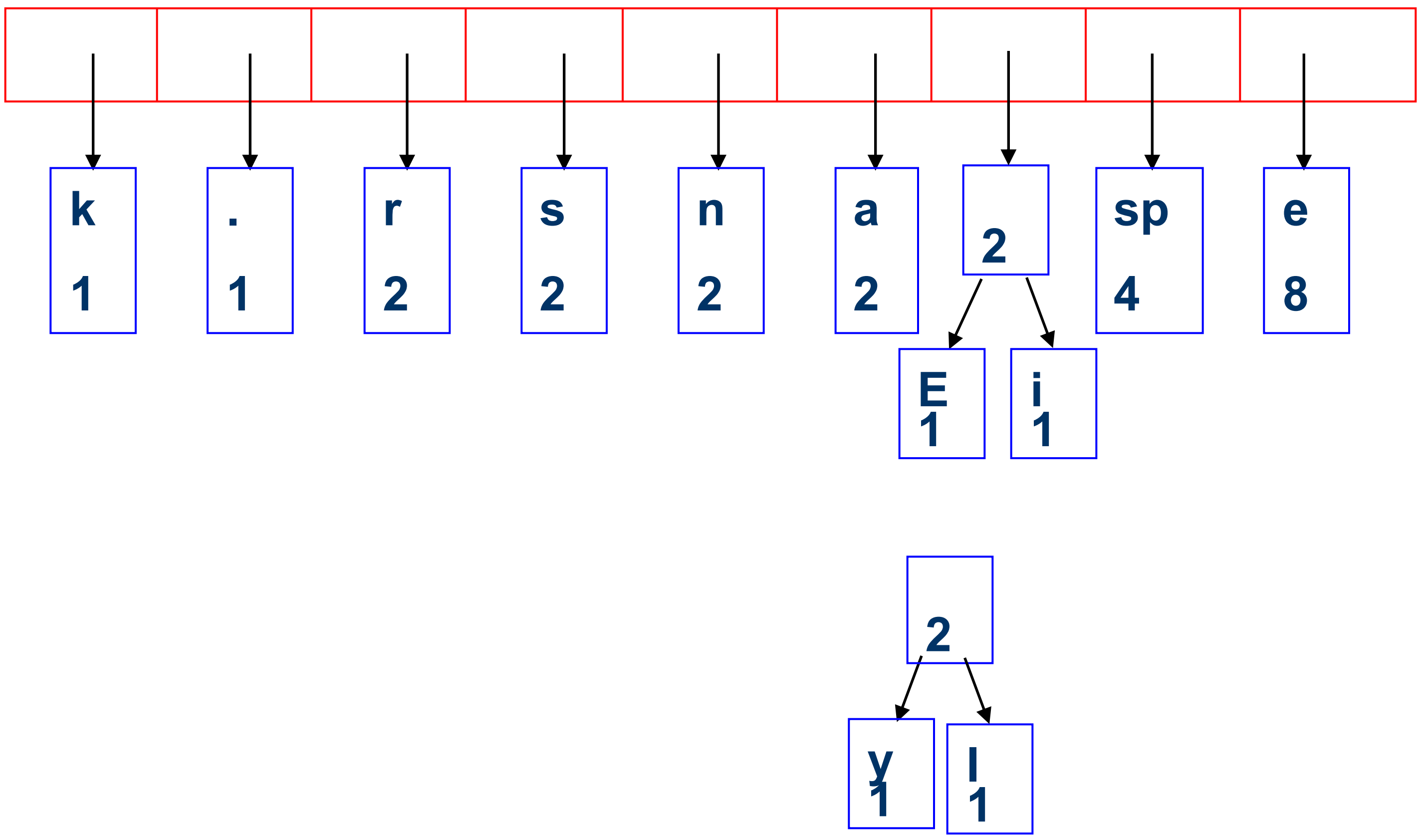
Building a Tree



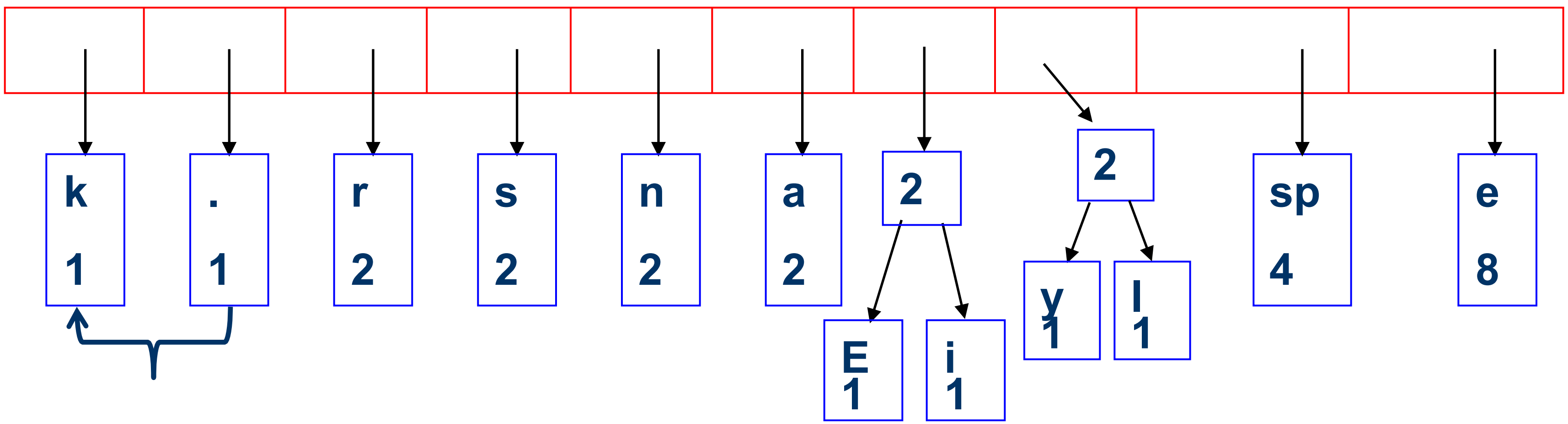
Building a Tree



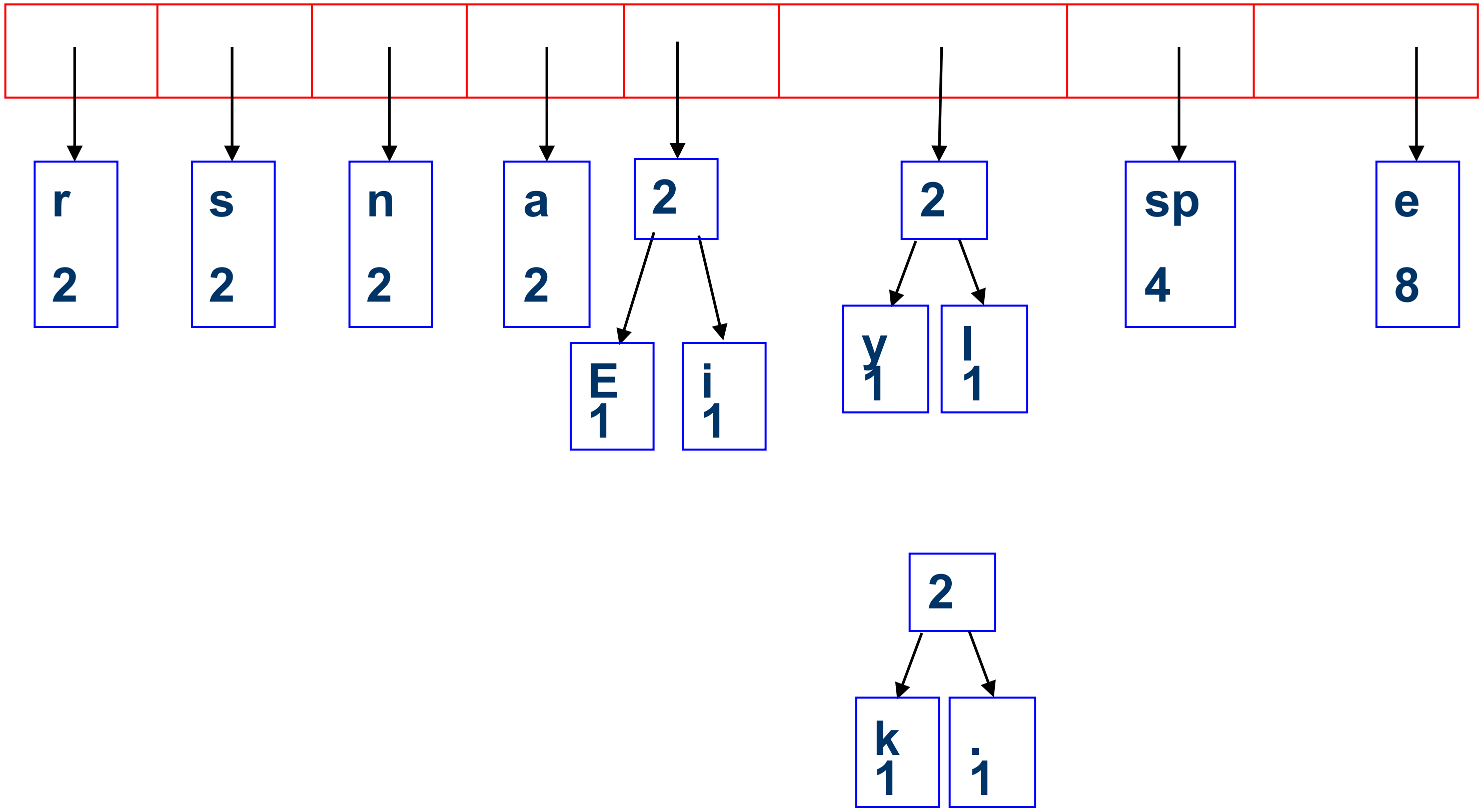
Building a Tree



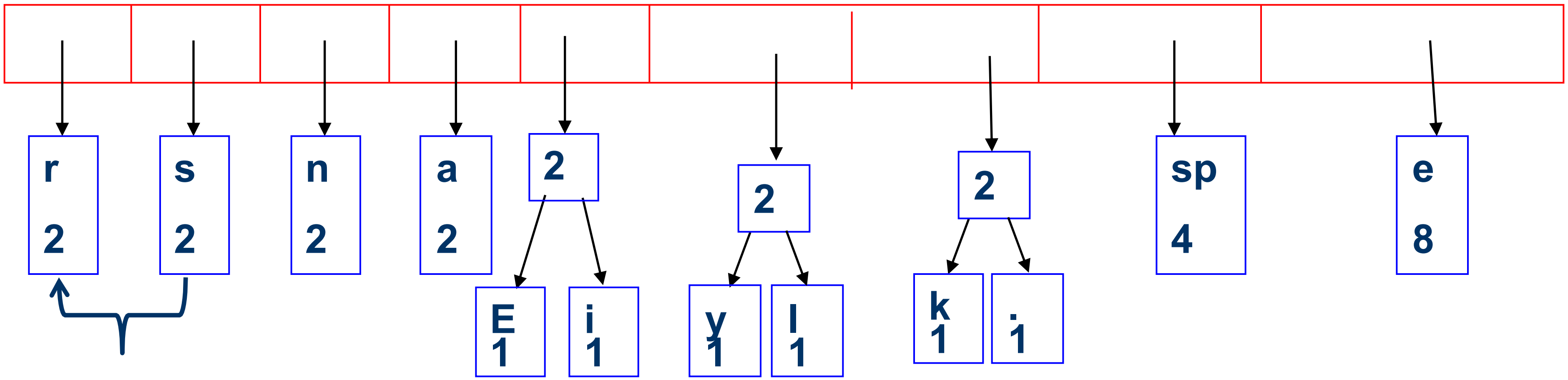
Building a Tree



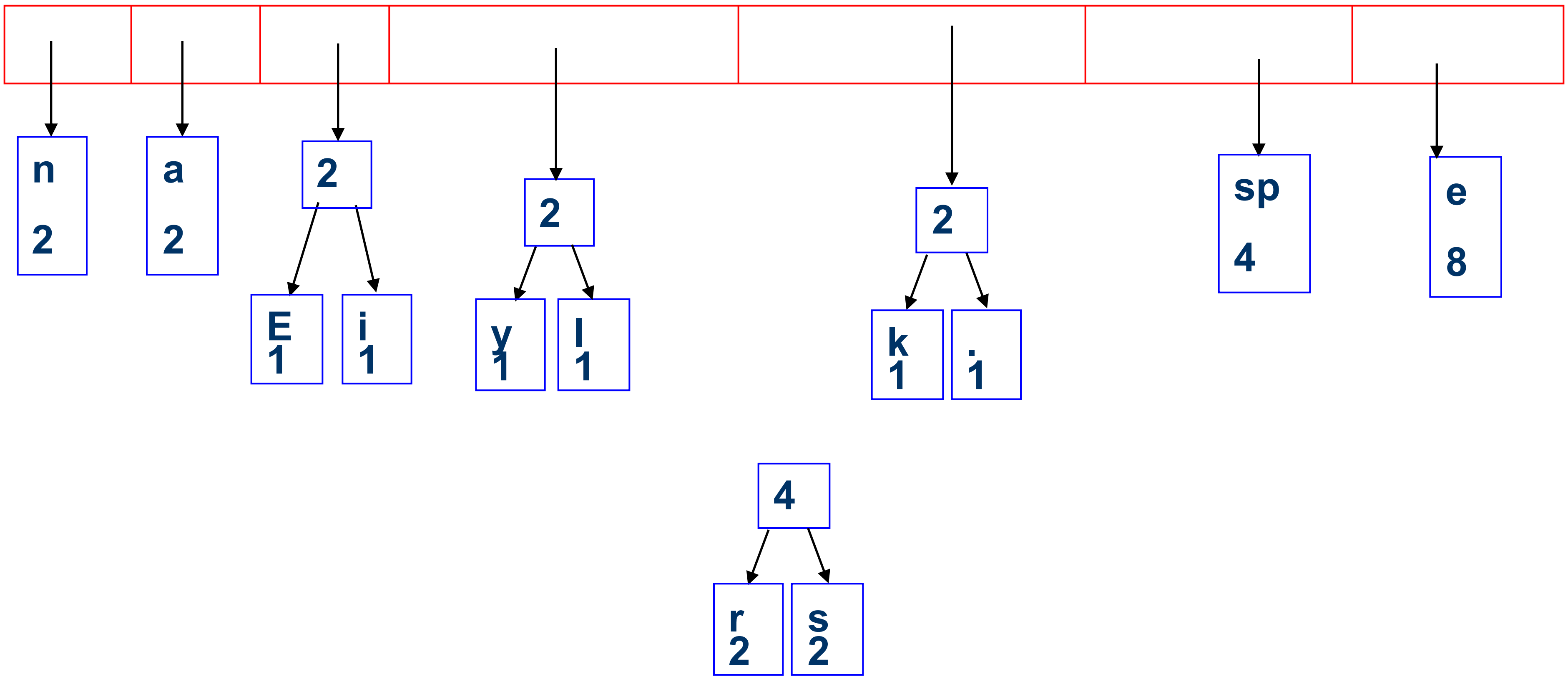
Building a Tree



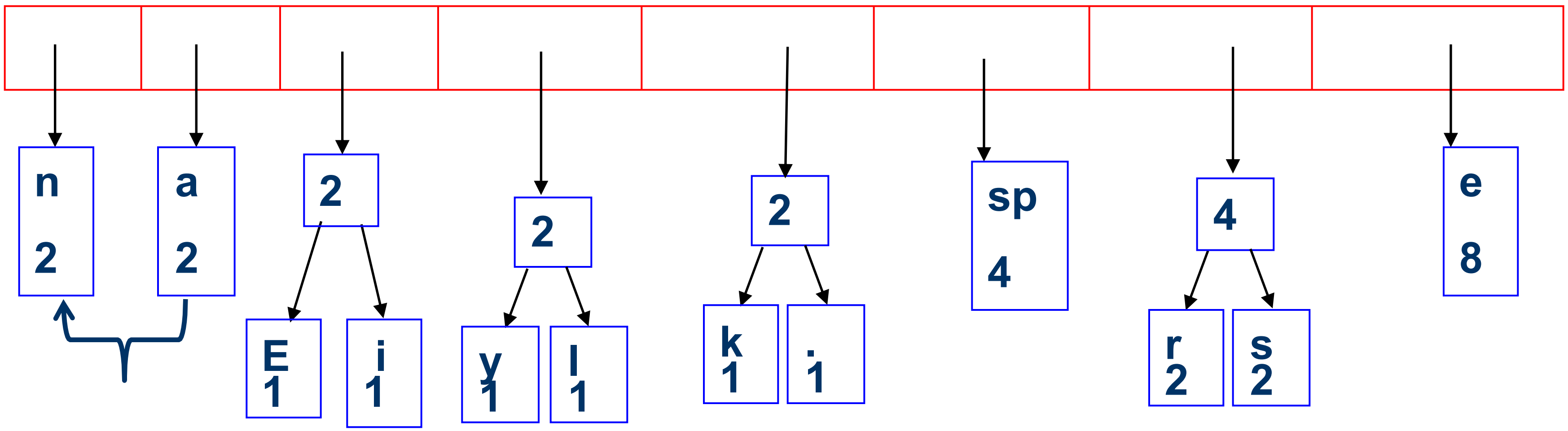
Building a Tree



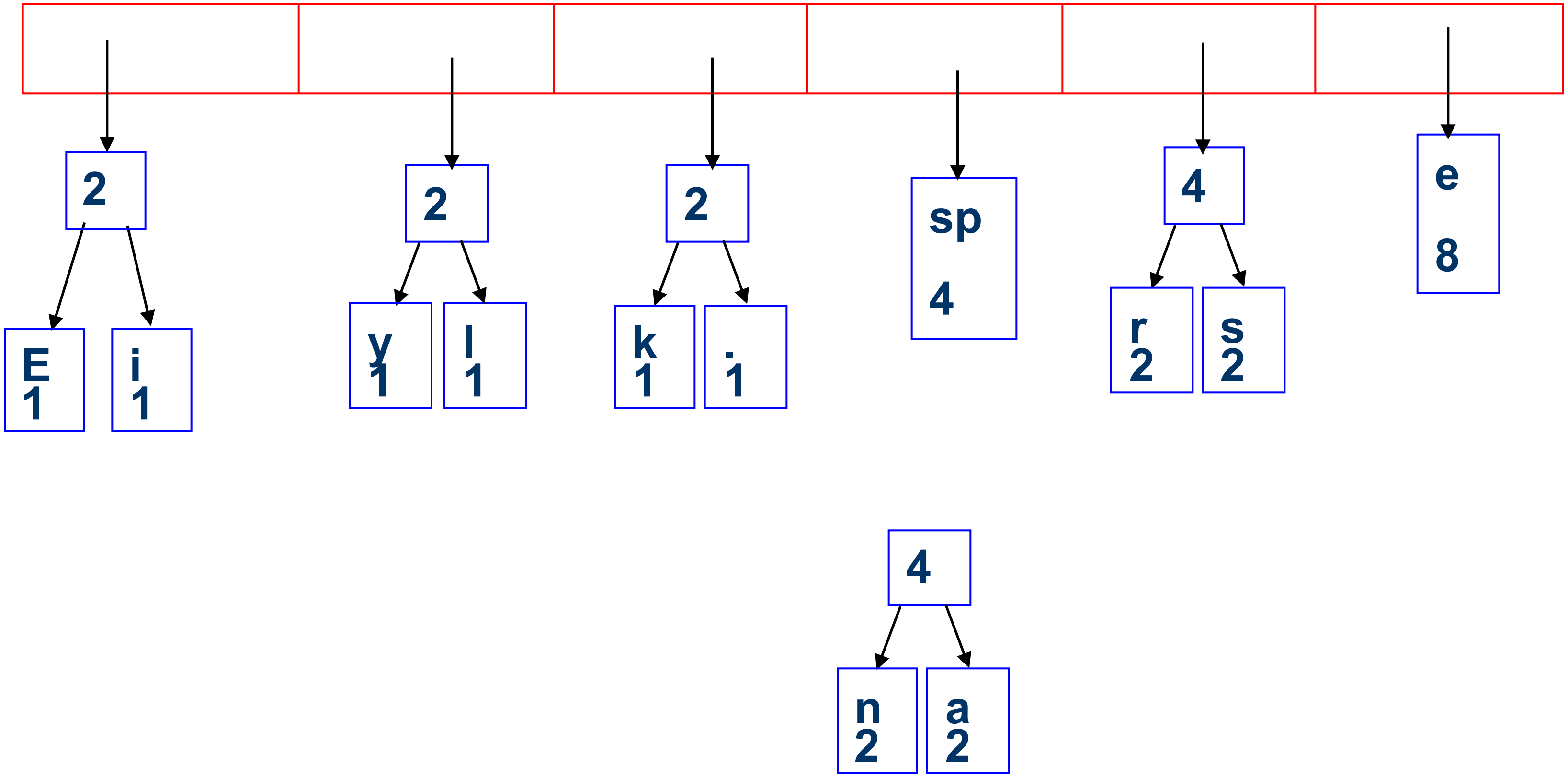
Building a Tree



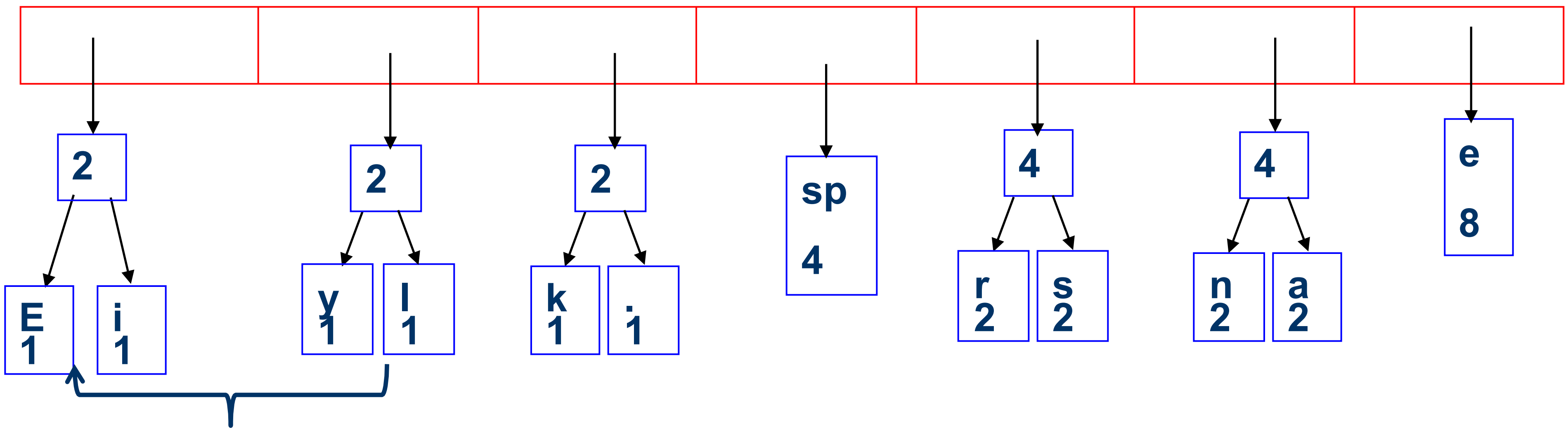
Building a Tree



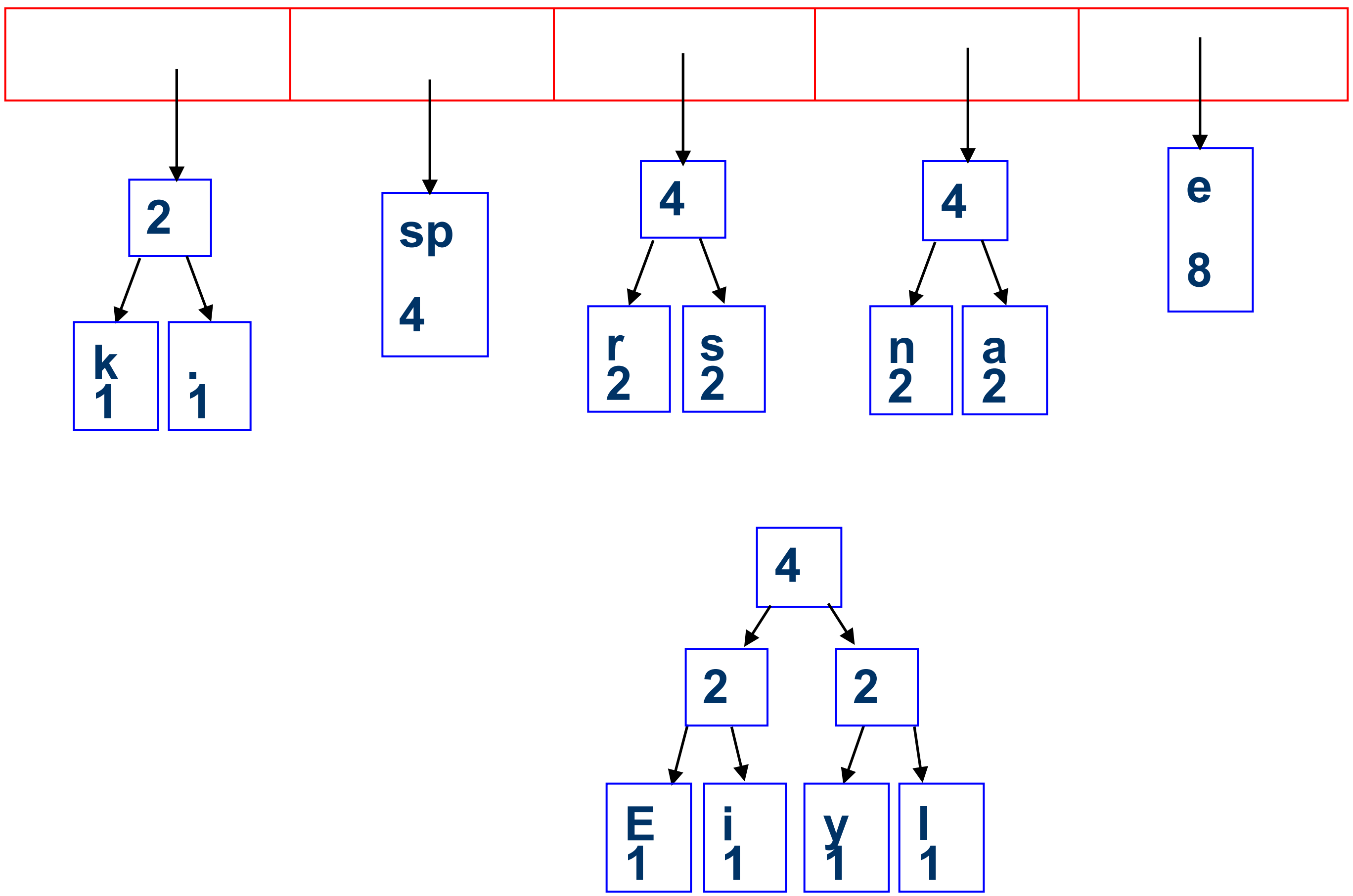
Building a Tree



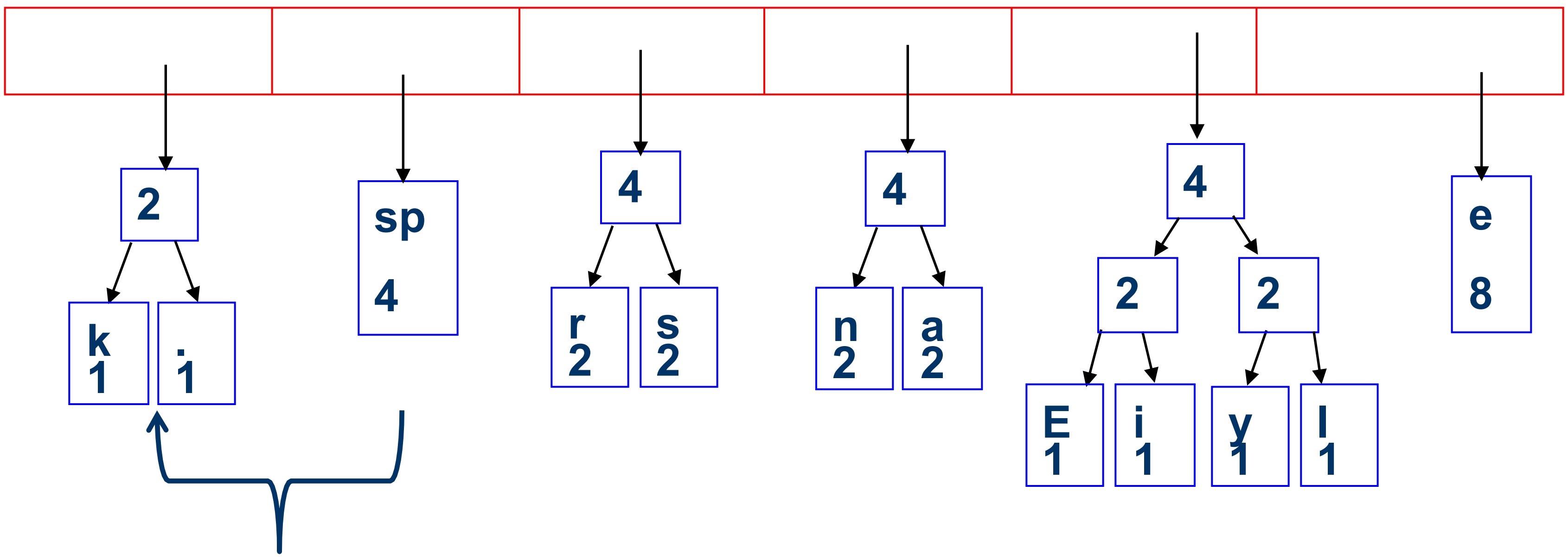
Building a Tree



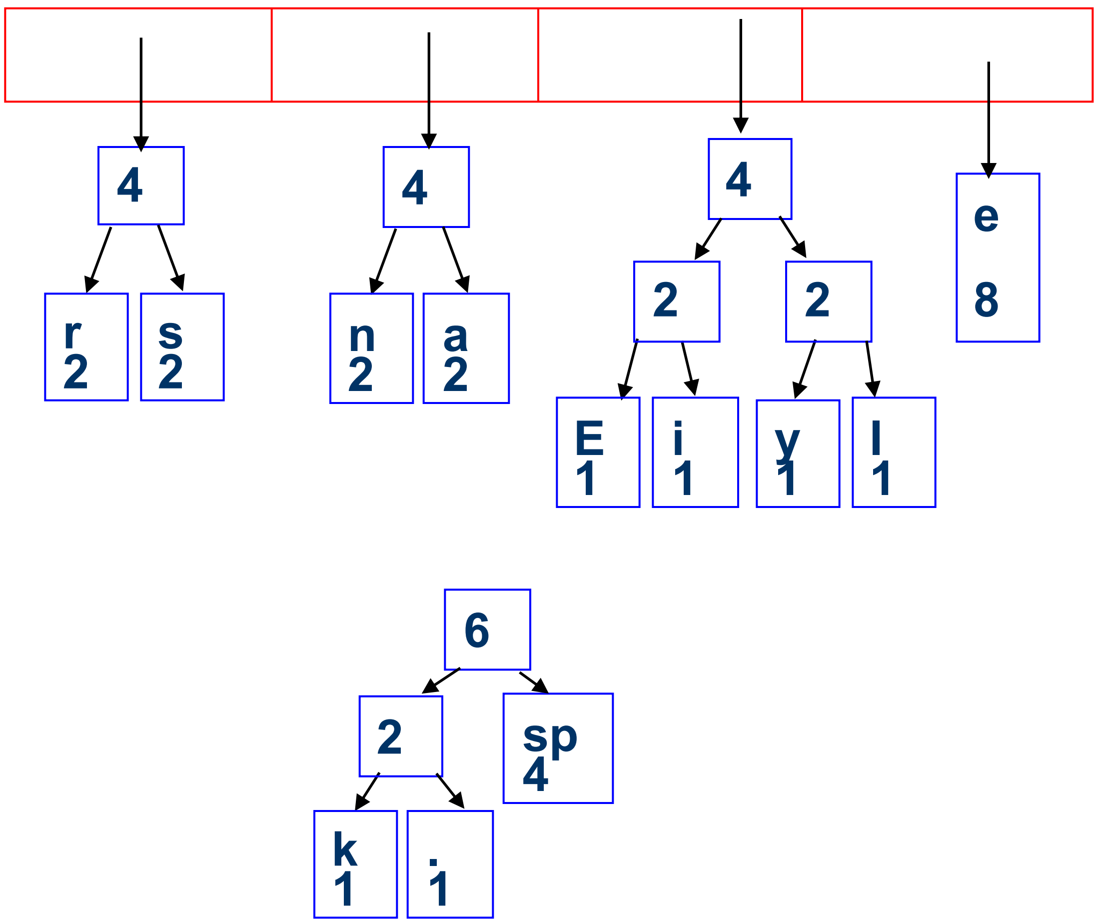
Building a Tree



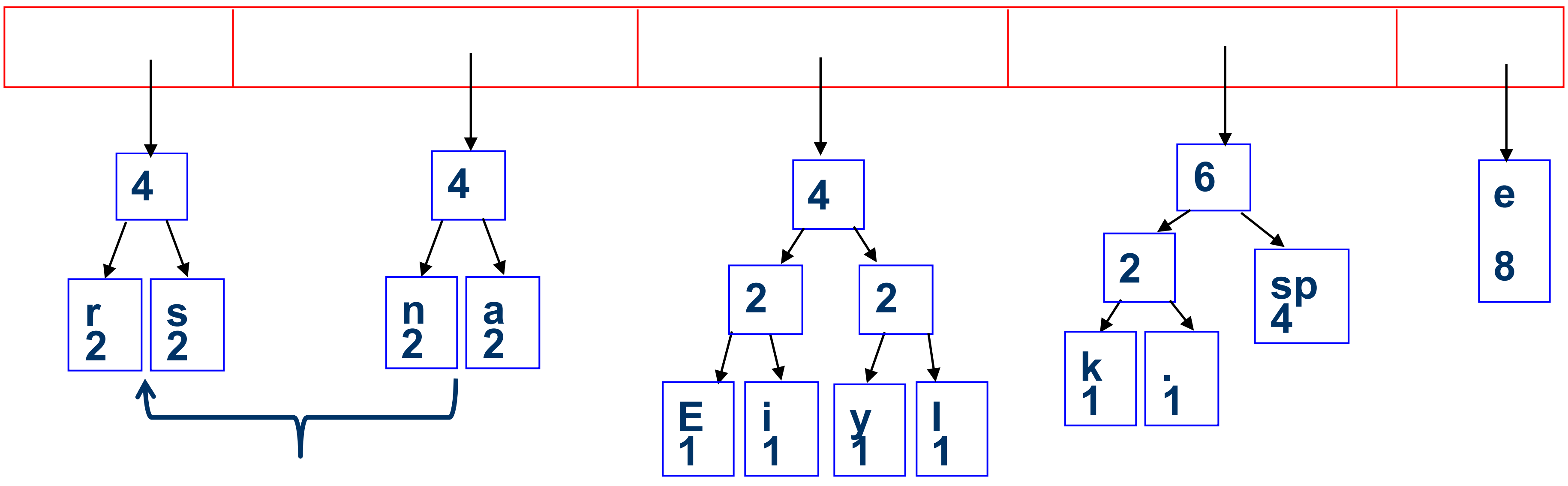
Building a Tree



Building a Tree

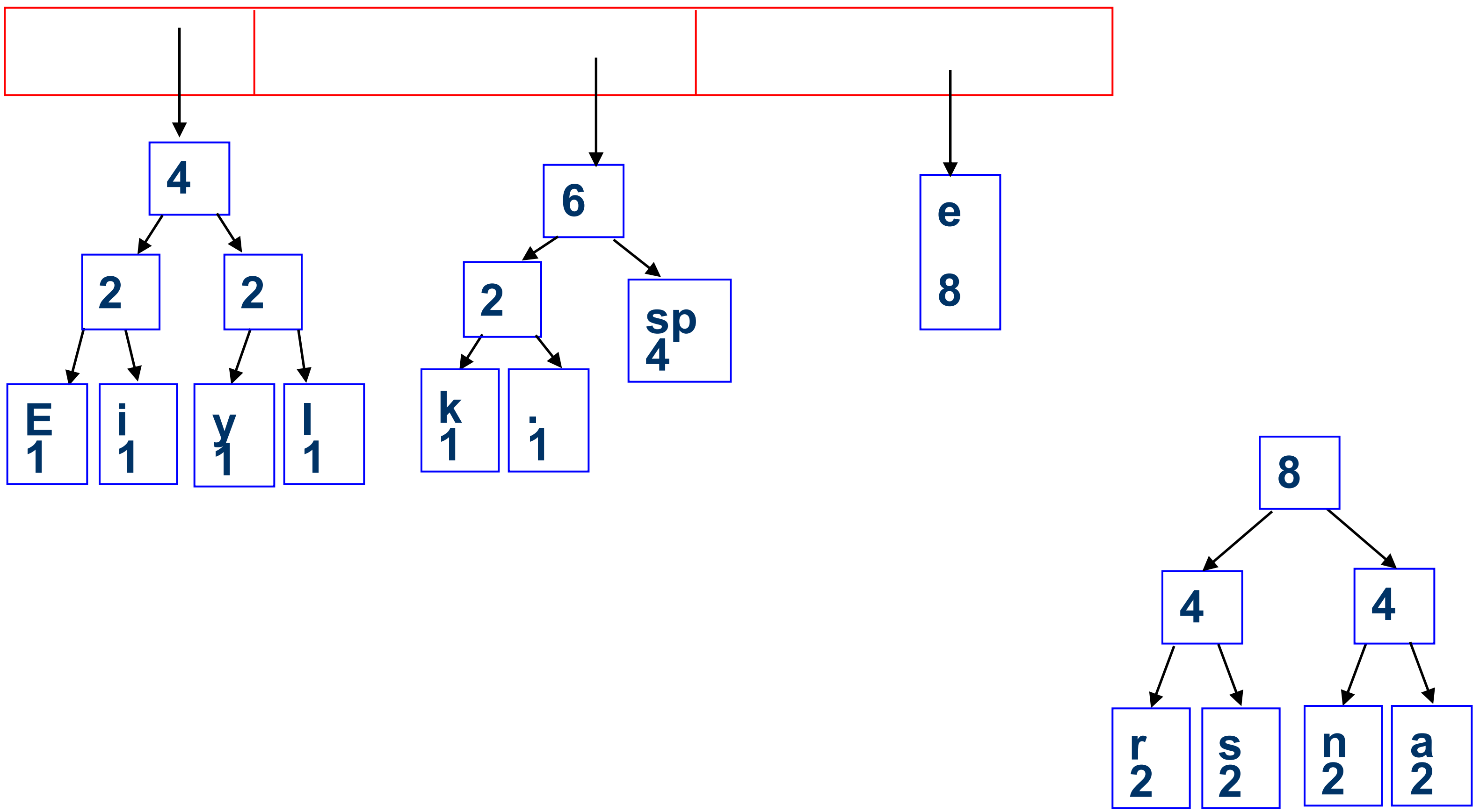


Building a Tree

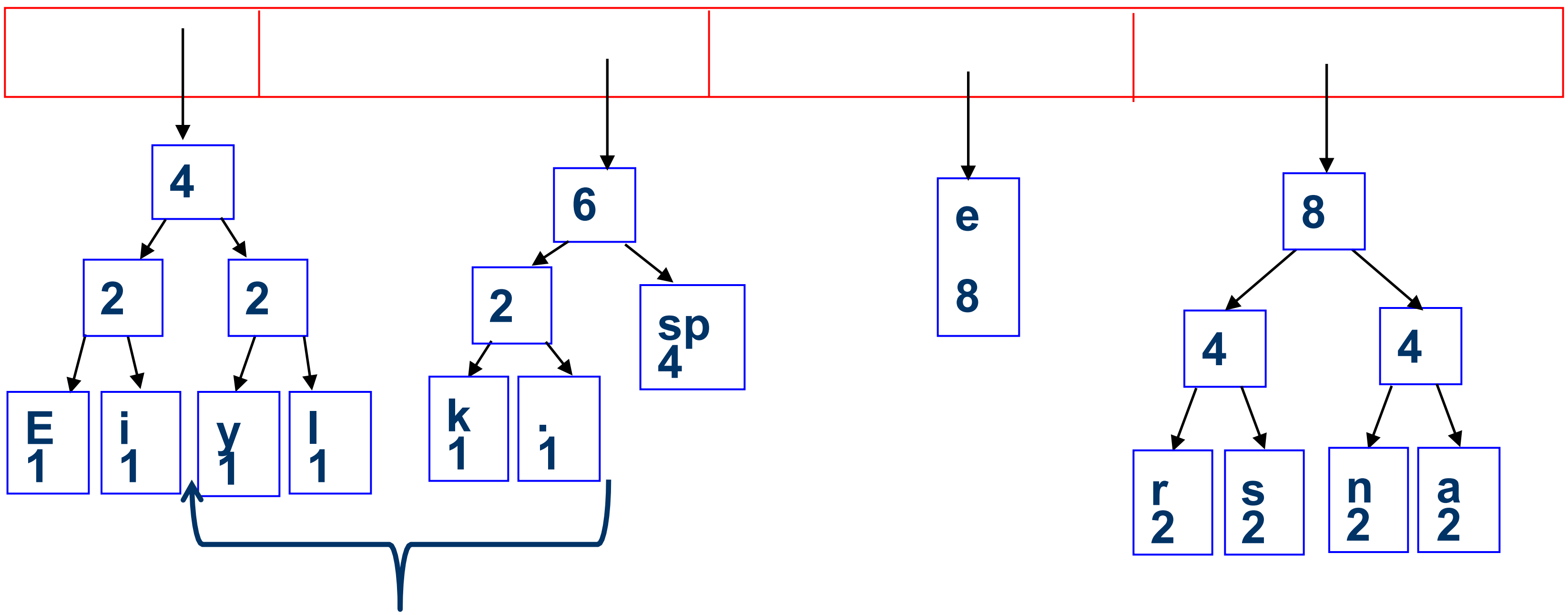


What is happening to the characters with a low number of occurrences?

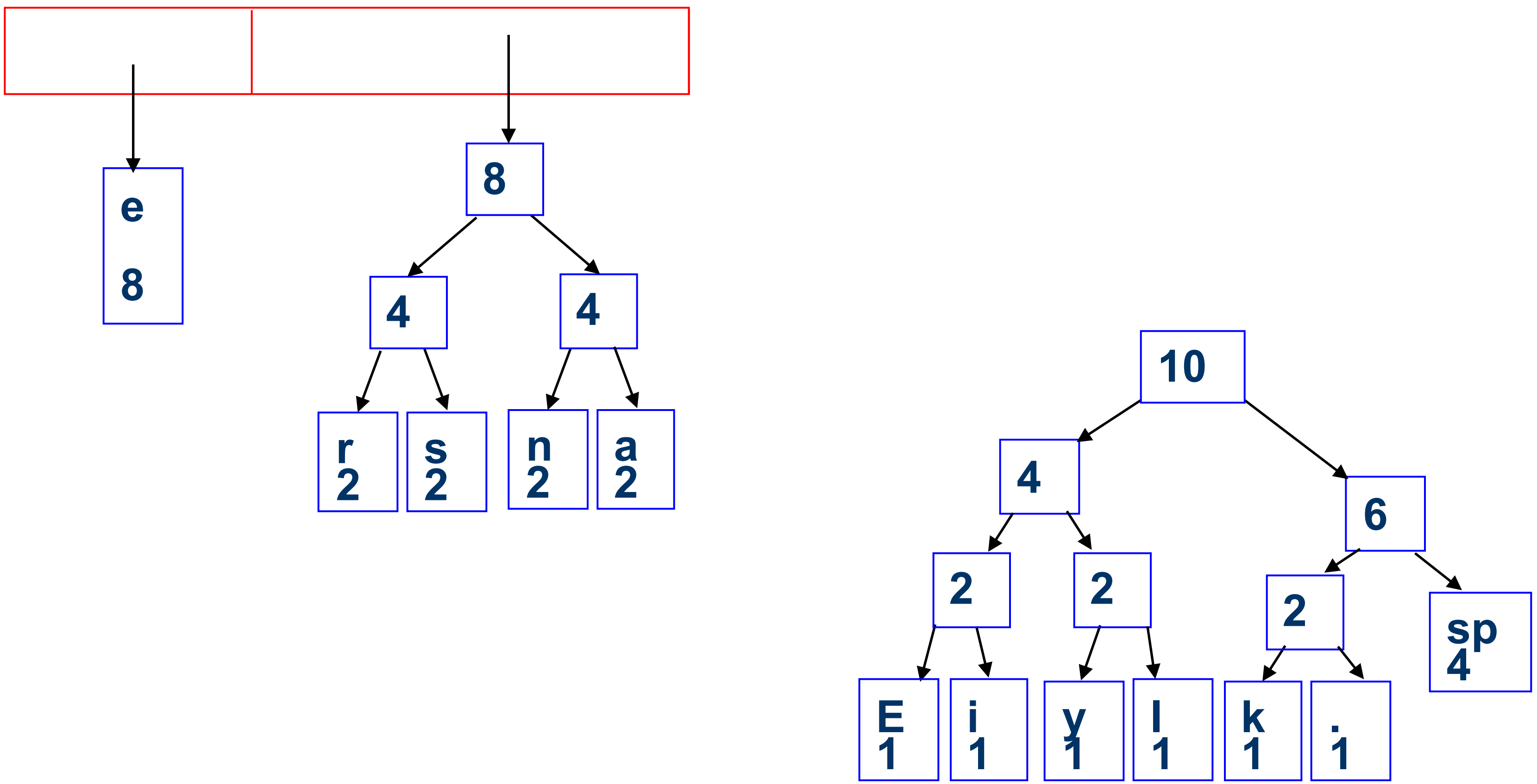
Building a Tree



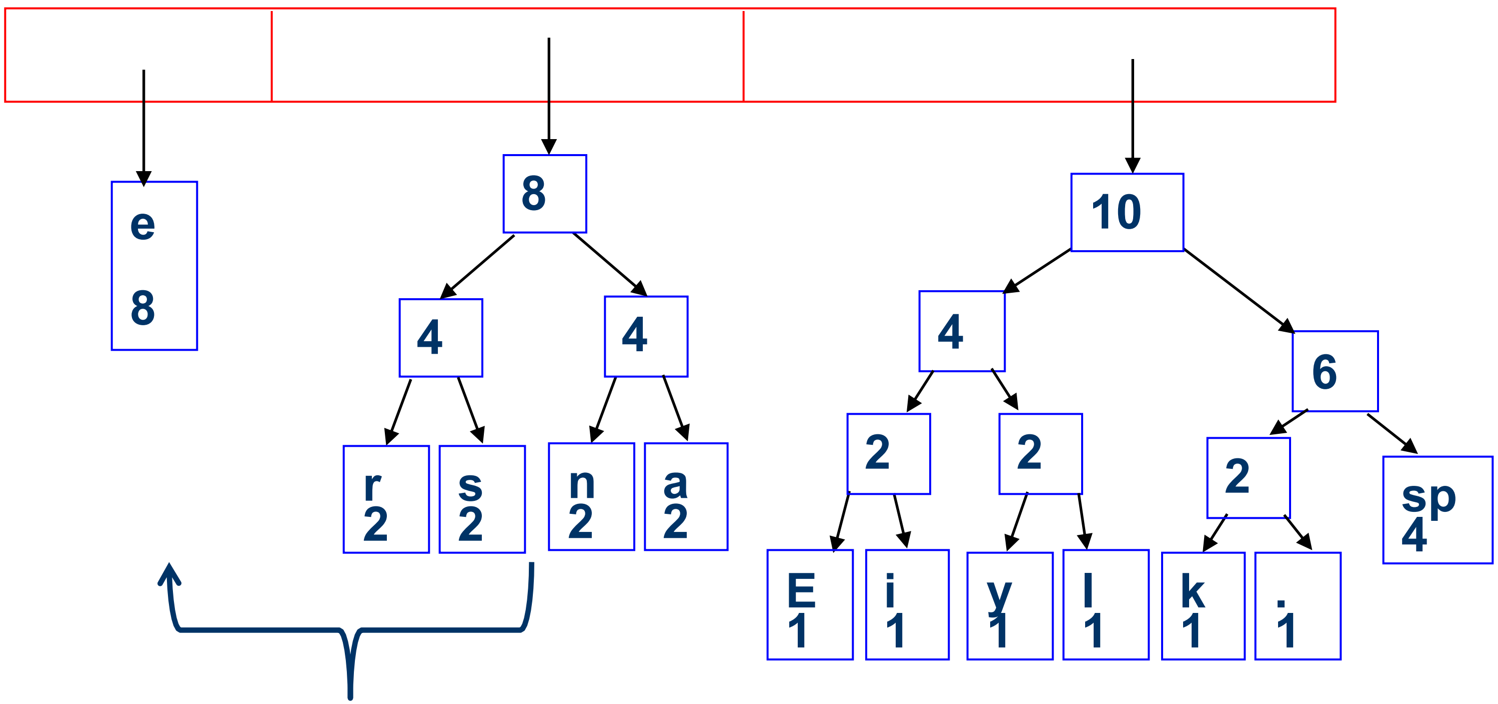
Building a Tree



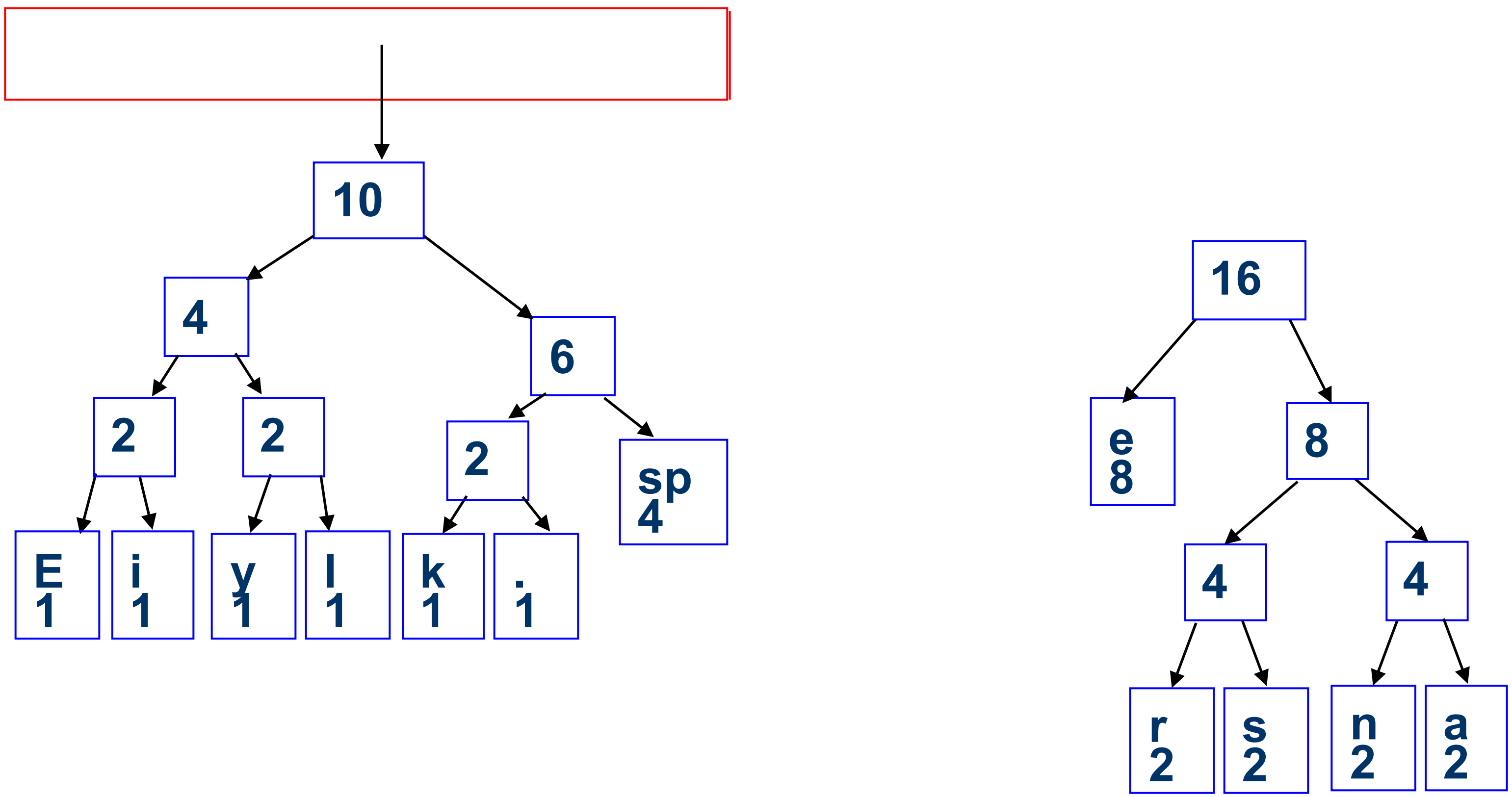
Building a Tree



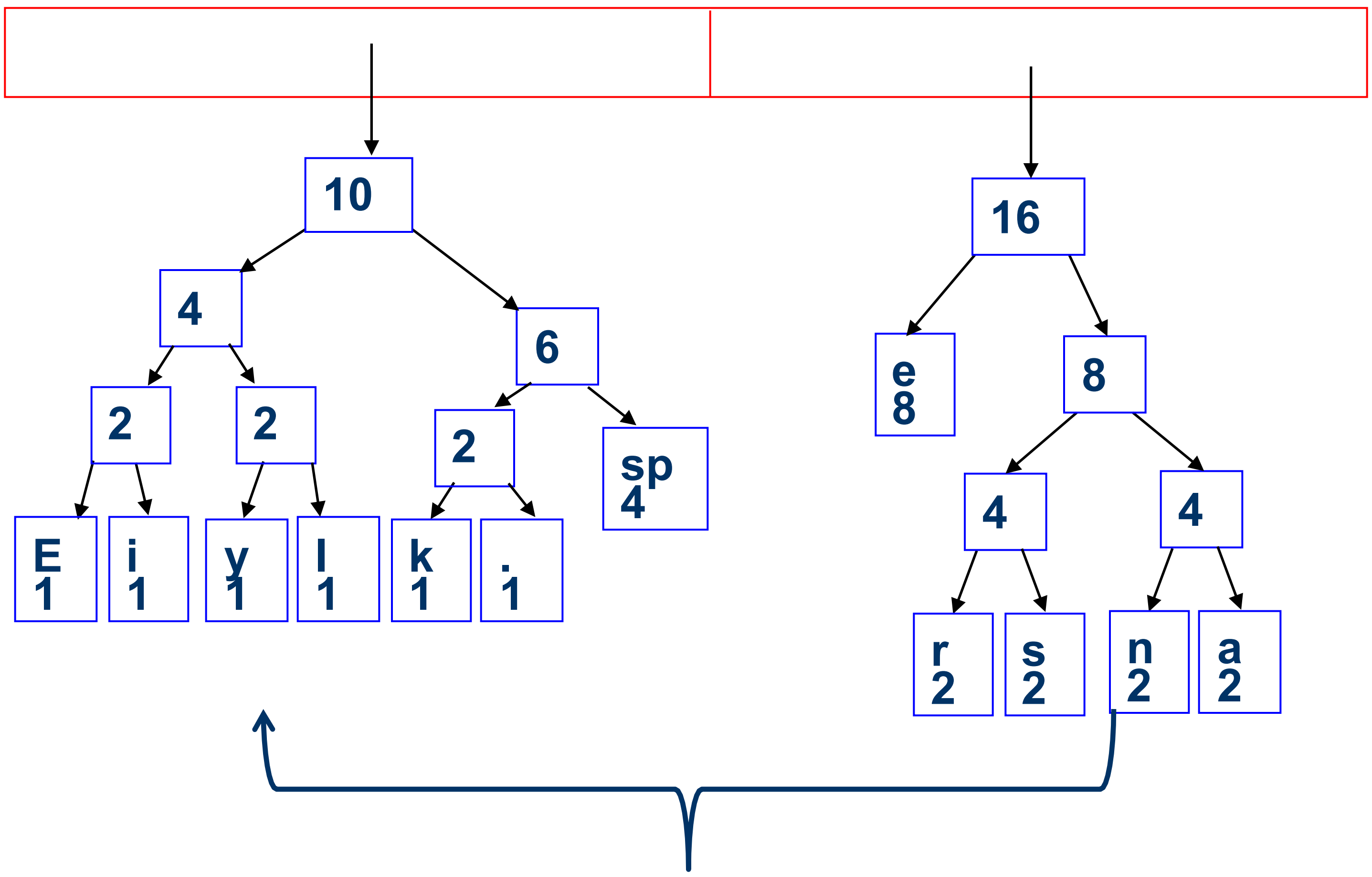
Building a Tree



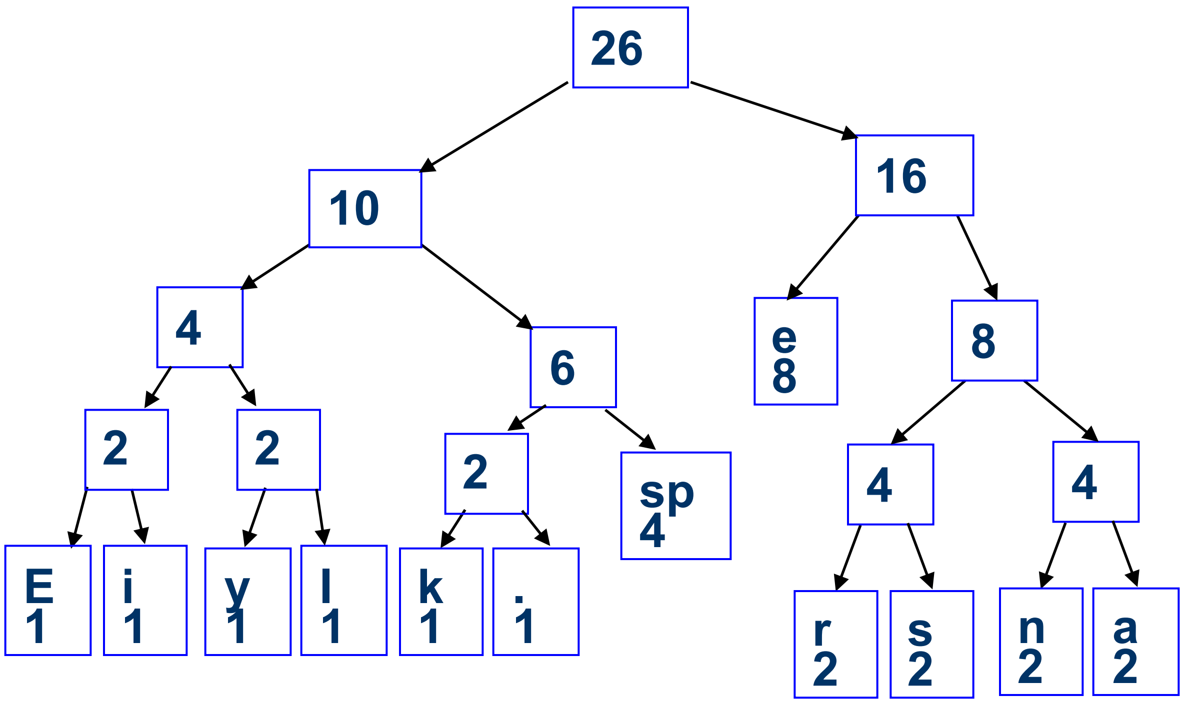
Building a Tree



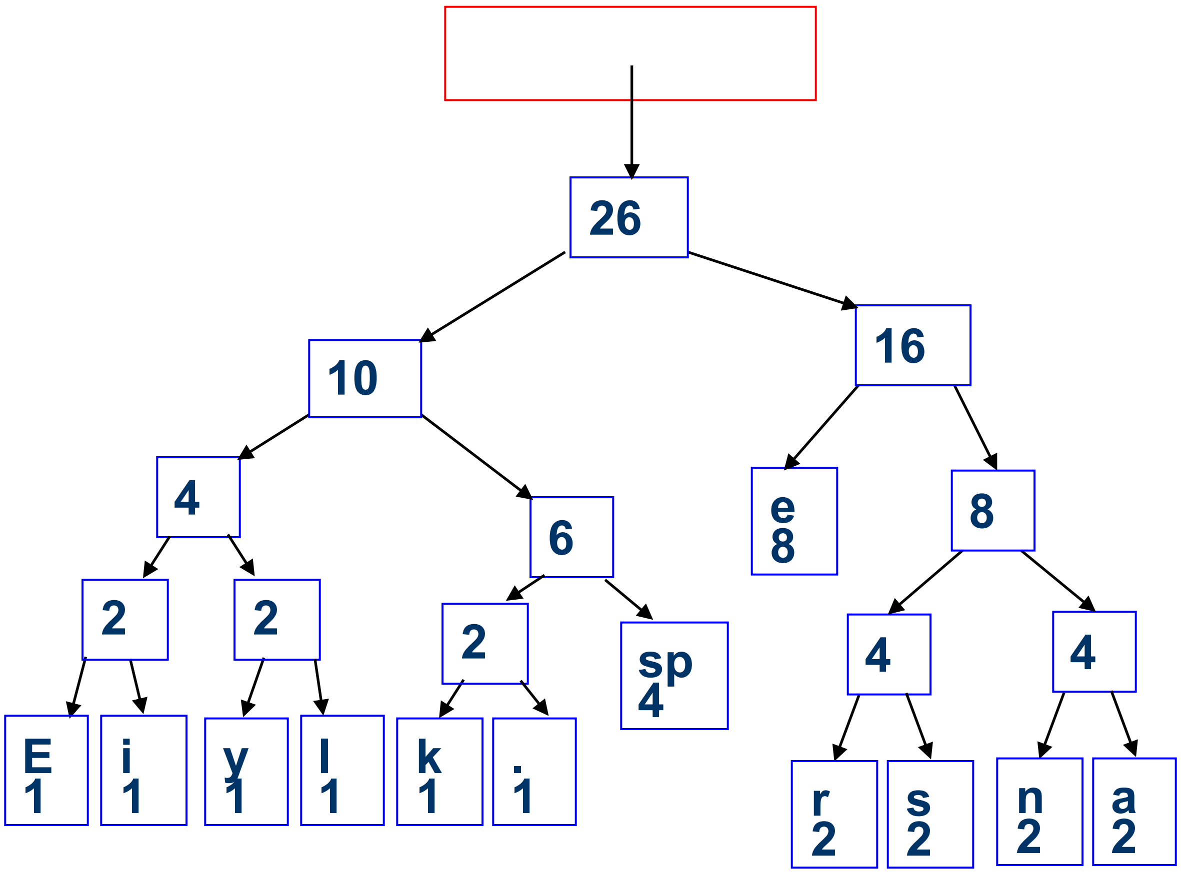
Building a Tree



Building a Tree



Building a Tree



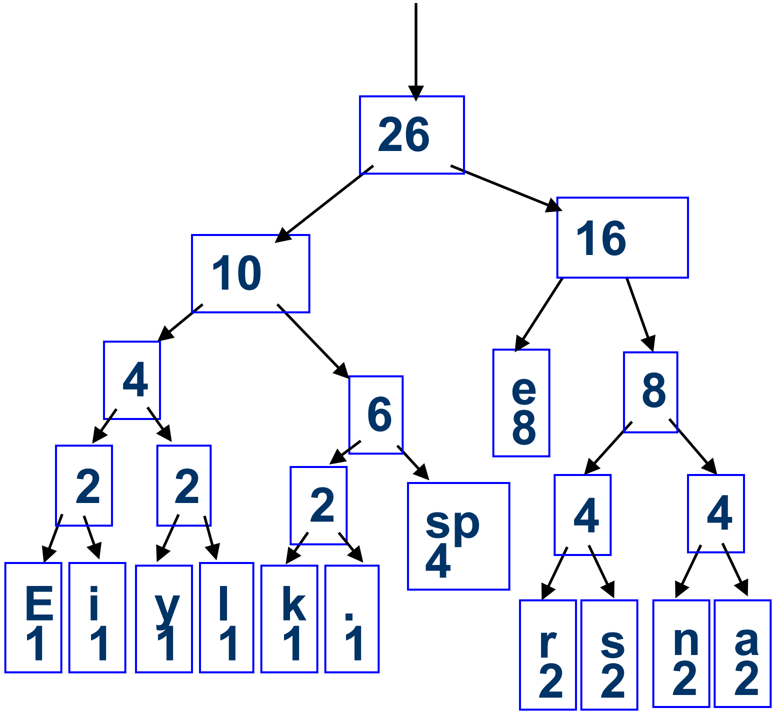
After enqueueing this node there is only one node left in priority queue.

Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

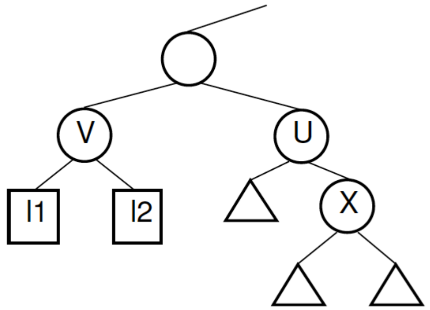
Frequency of root node should equal number of characters in text.



Eerie eyes seen near lake. □ 26 characters

Building a Huffman Tree

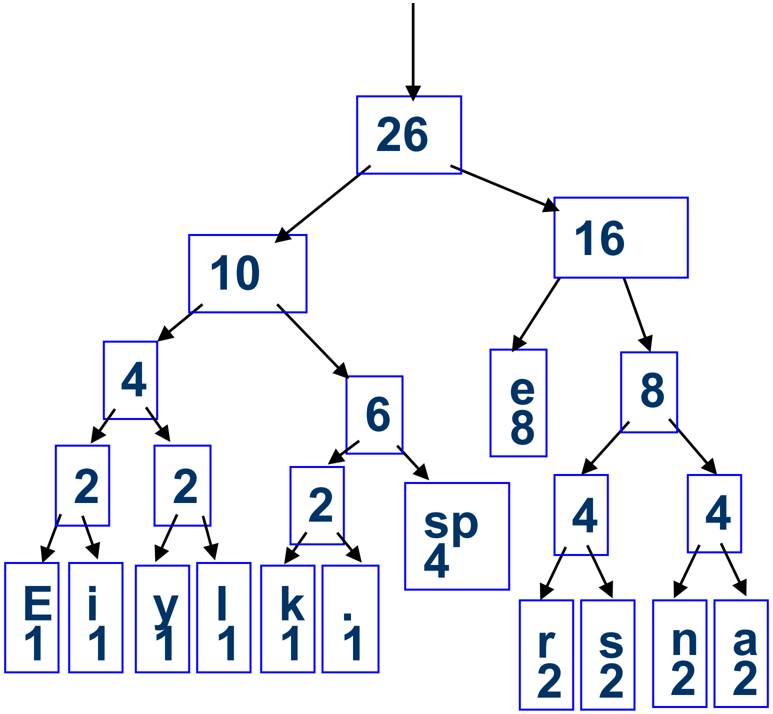
- ✓ Analysis:
  - ✓ Each node will have storage for two data items:
    - ✓ the weight of the node and
    - ✓ the symbol associated with the node
  - ✓ All symbols will be stored in leaf nodes
  - ✓ For nodes that are not leaf nodes, the symbol part has no meaning



An impossible Huffman tree, showing the situation where the two nodes with least weight, I1 and I2, are not the deepest nodes in the tree. Triangles represent subtrees.

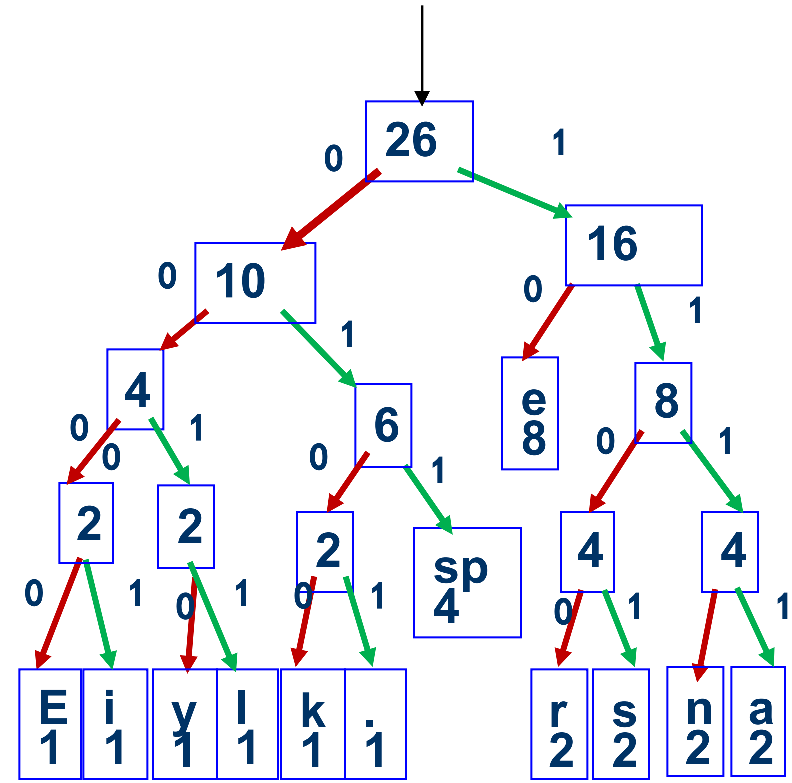
Encoding the File
Traverse Tree for Codes

- 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

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.

```
0000101100000110011100
0011011010111101101011
1001111101011111100011
001111110100100101
```

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

- Why is there no need for a separator character?

.

Encoding the File Results

- Have we made things any better?
- 84 bits to encode the text
- ASCII would take

  8 * 26 = 208 bits

```
000010110000011001110
000110110101111011010 1
11001111101011111100 01
1001111110100100101
```

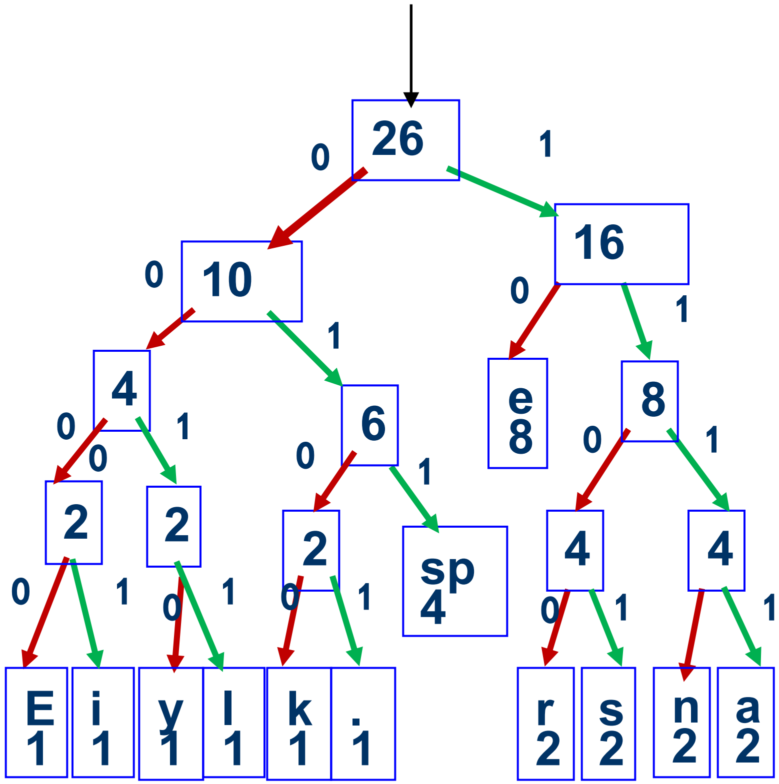
Decoding the File

- 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

10100011011110111101111110000110101



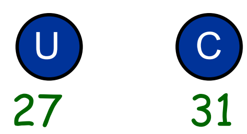
Huffman Code Construction

- Character count in text.

Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

Huffman Code Construction

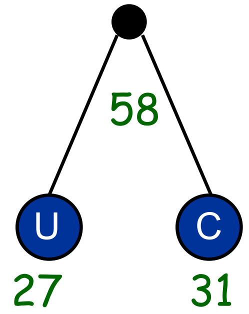
Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27



Huffman Code Construction

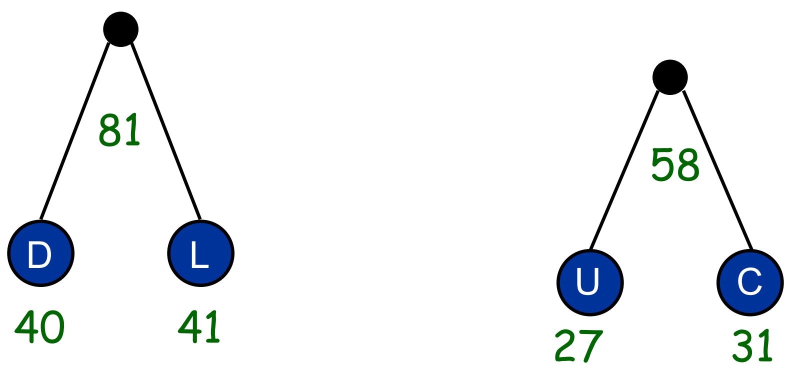
Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
	58
H	55
L	41
D	40

C	31
U	27



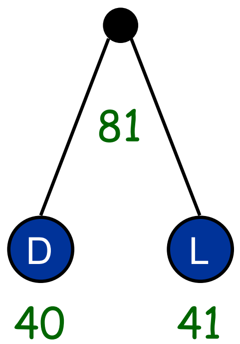
Huffman Code Construction

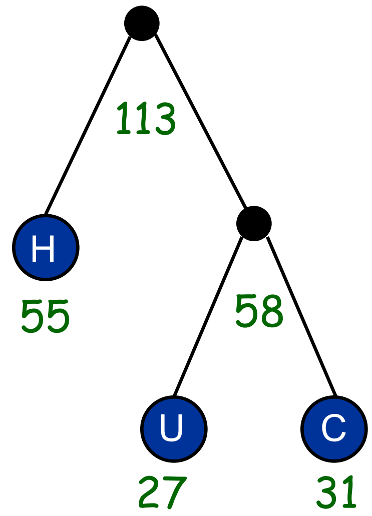
Char	Freq
E	125
T	93
	81
A	80
O	76
I	73
N	71
S	65
R	61
	58
H	55



L	41
D	40

Huffman Code Construction

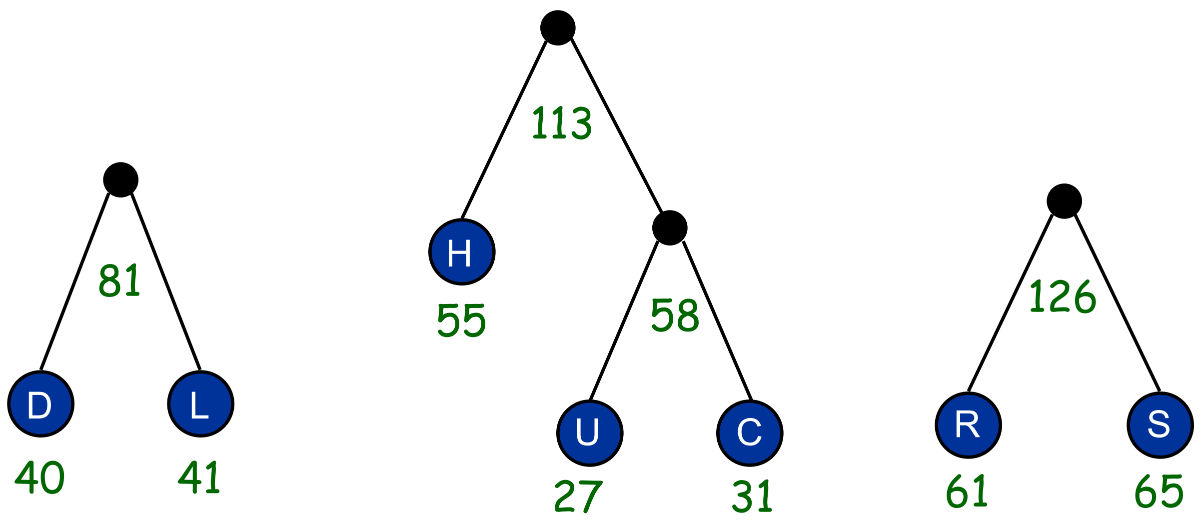




Char	Freq
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71
S	65
R	61

	58
H	55

Huffman Code Construction



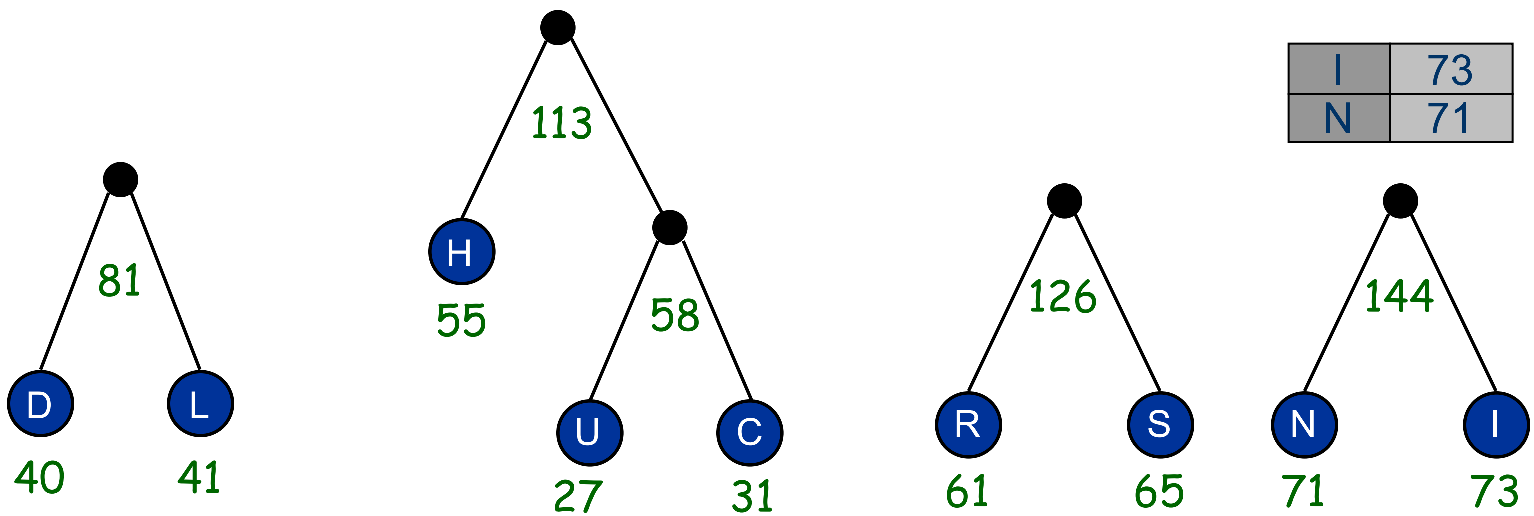
Char	Freq
	126
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71

S	65
R	61

Huffman Code Construction

Char	Freq
	144
	126
E	125
	113
T	93
	81
A	80
O	76

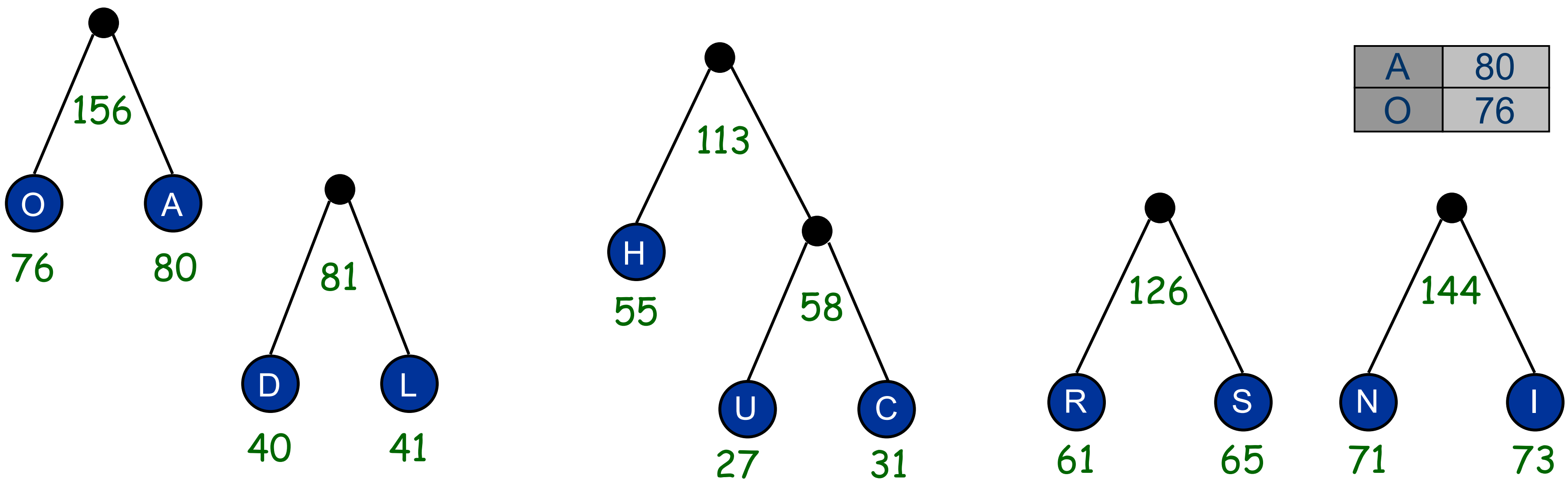
I	73
N	71



Huffman Code Construction

Char	Freq
	156
	144
	126
E	125
	113
T	93
	81

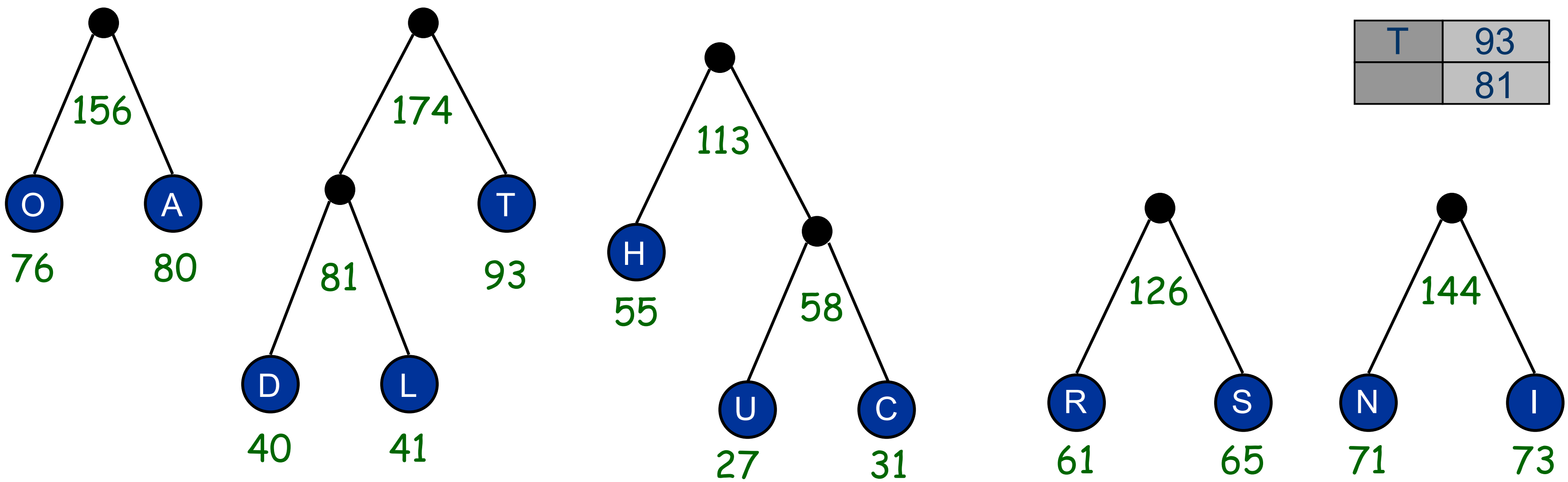
A	80
O	76



Huffman Code Construction

Char	Freq
	174
	156
	144
	126
E	125
	113

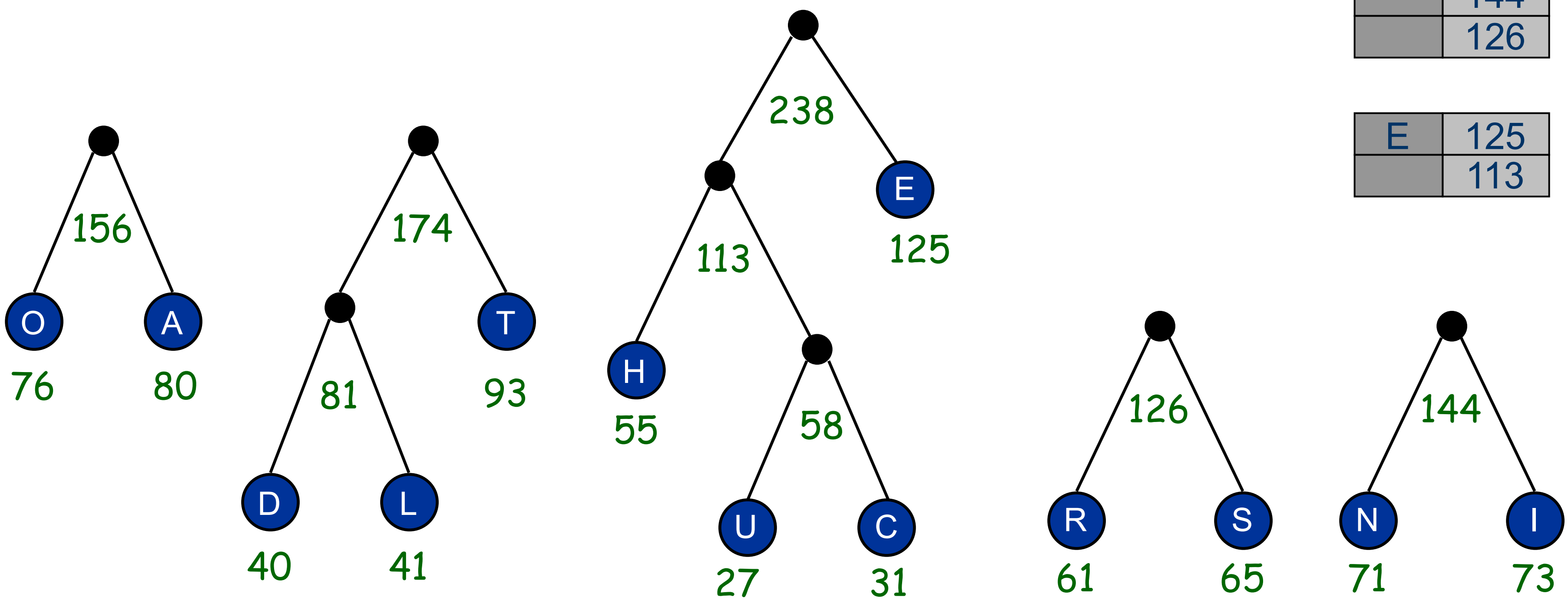
T	93
	81



Huffman Code Construction

Char	Freq
	238
	174
	156
	144
	126

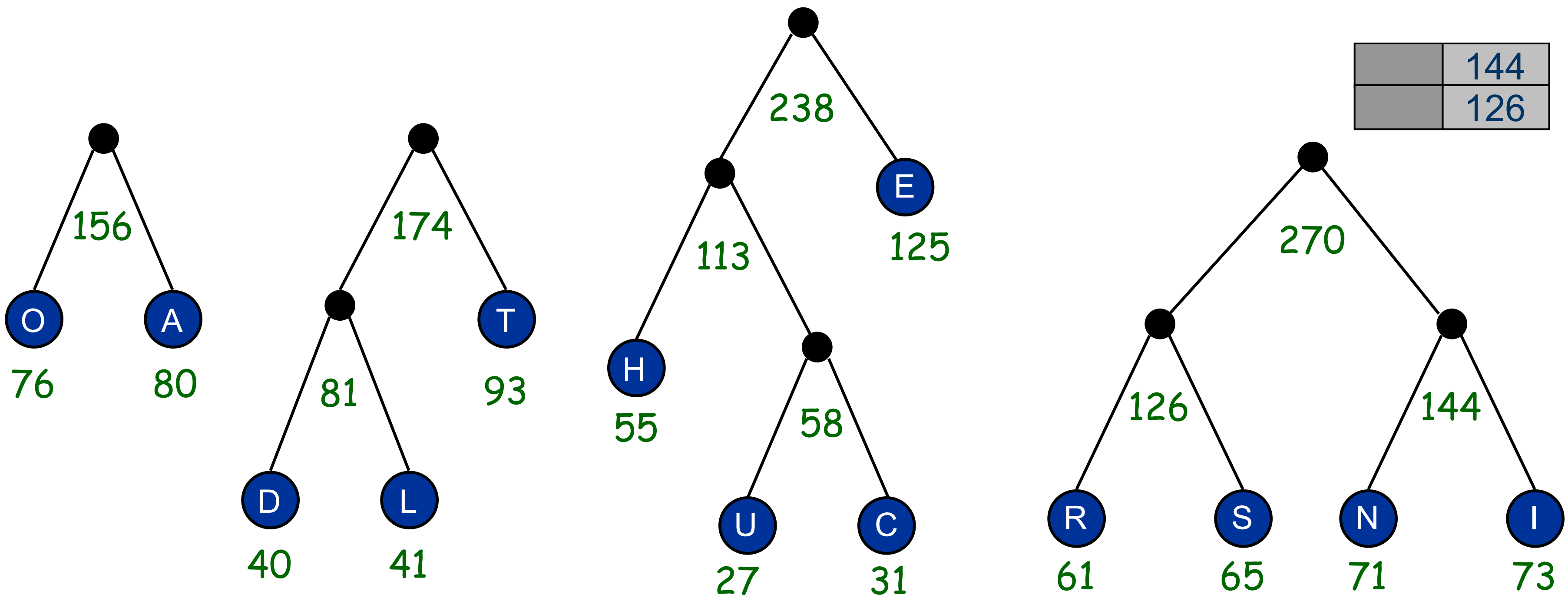
E	125
	113



Huffman Code Construction

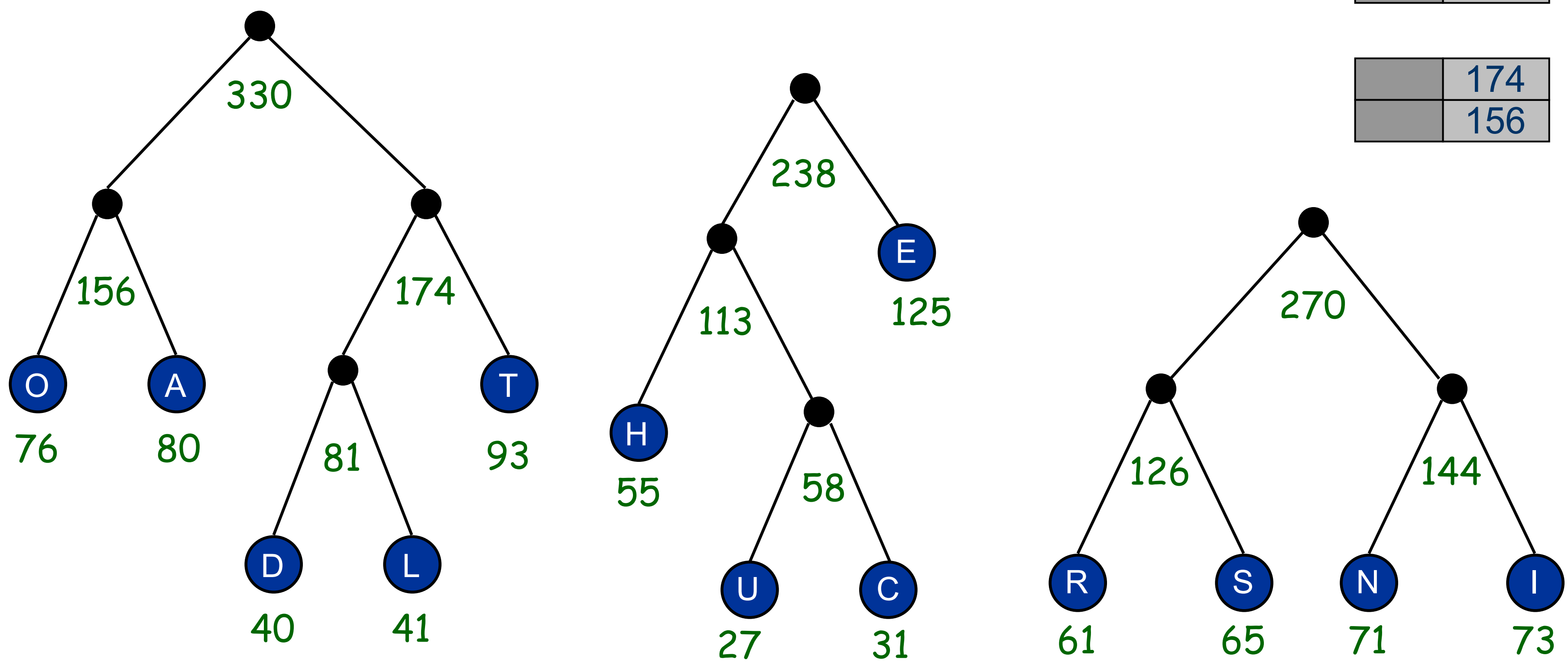
Char	Freq
	270
	238
	174
	156

	144
	126



Huffman Code Construction

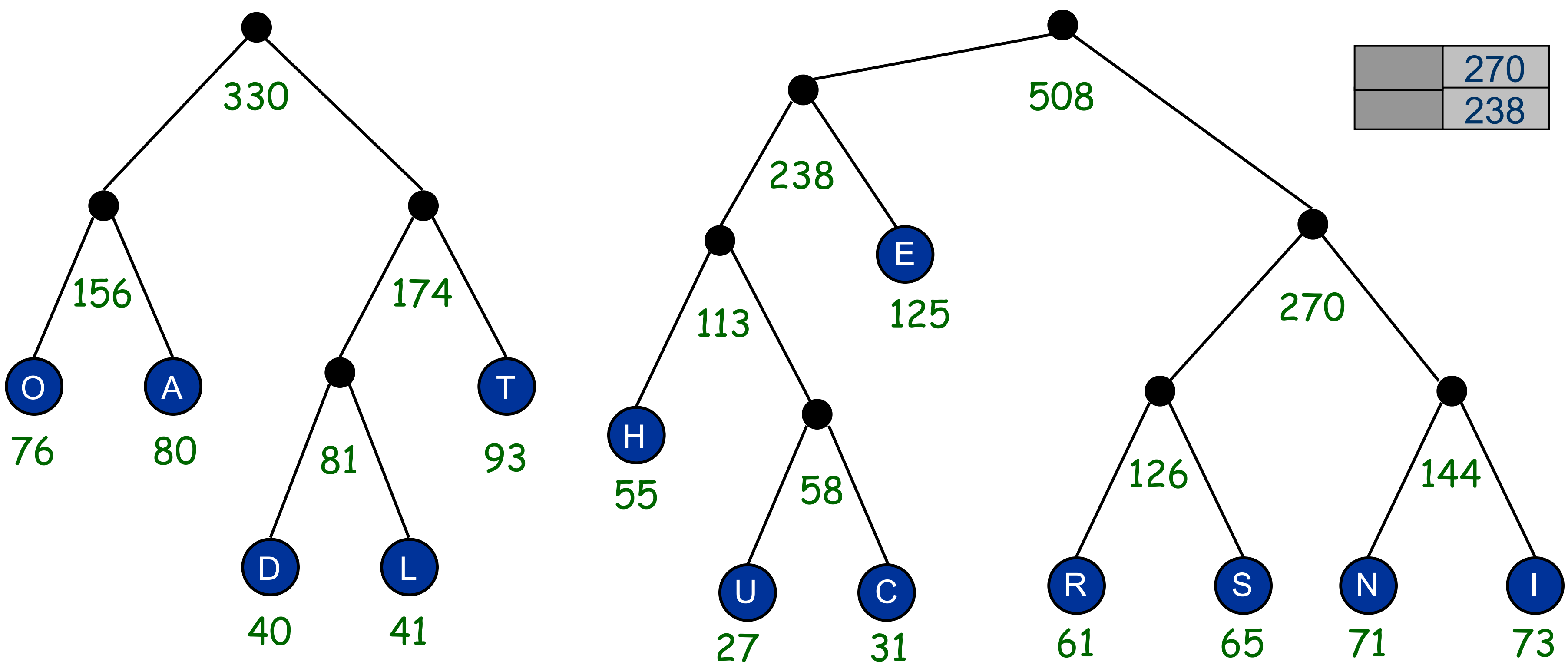
Char	Freq
	330
	270
	238
	174
	156



Huffman Code Construction

Char	Freq
	508
	330

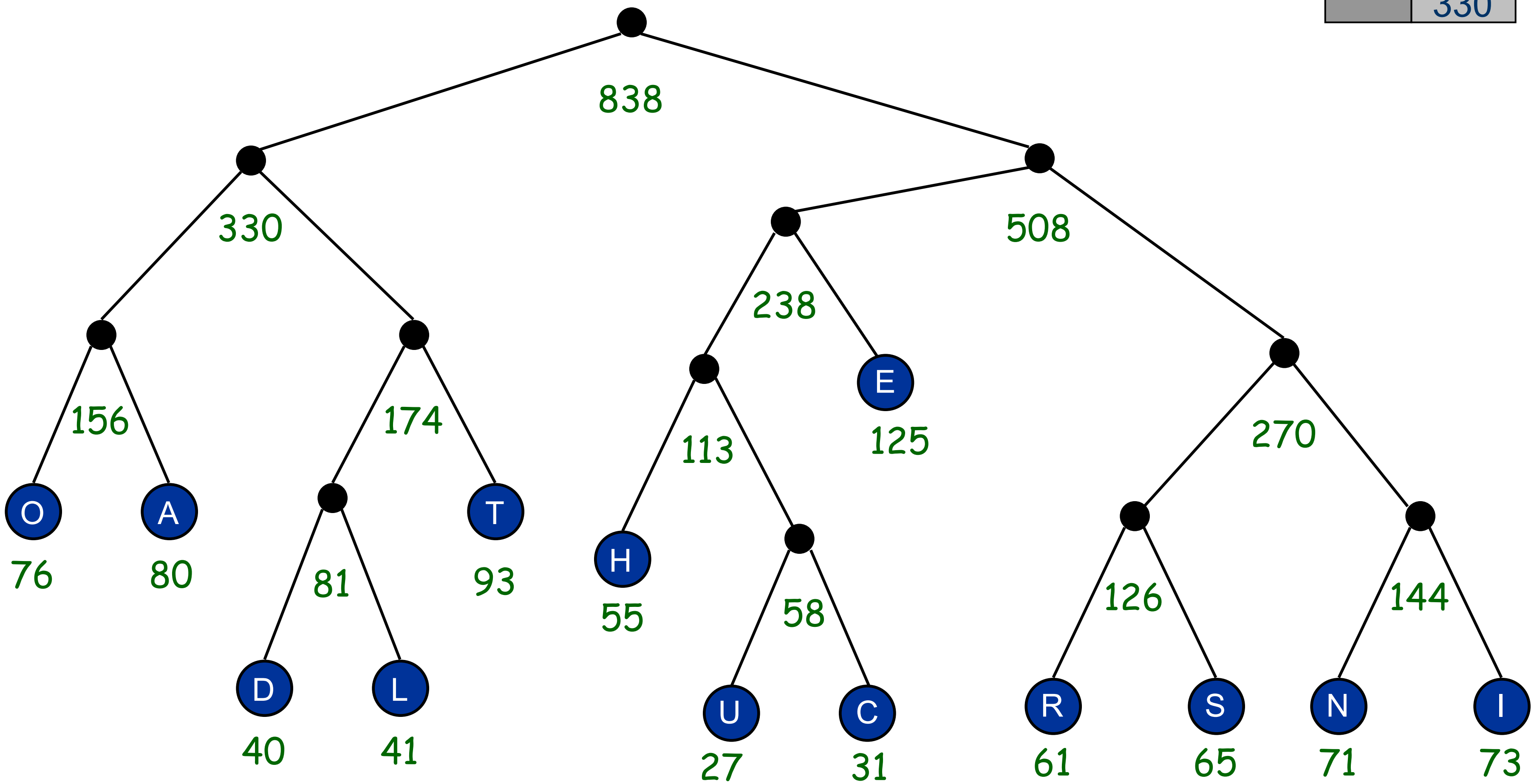
	270
	238



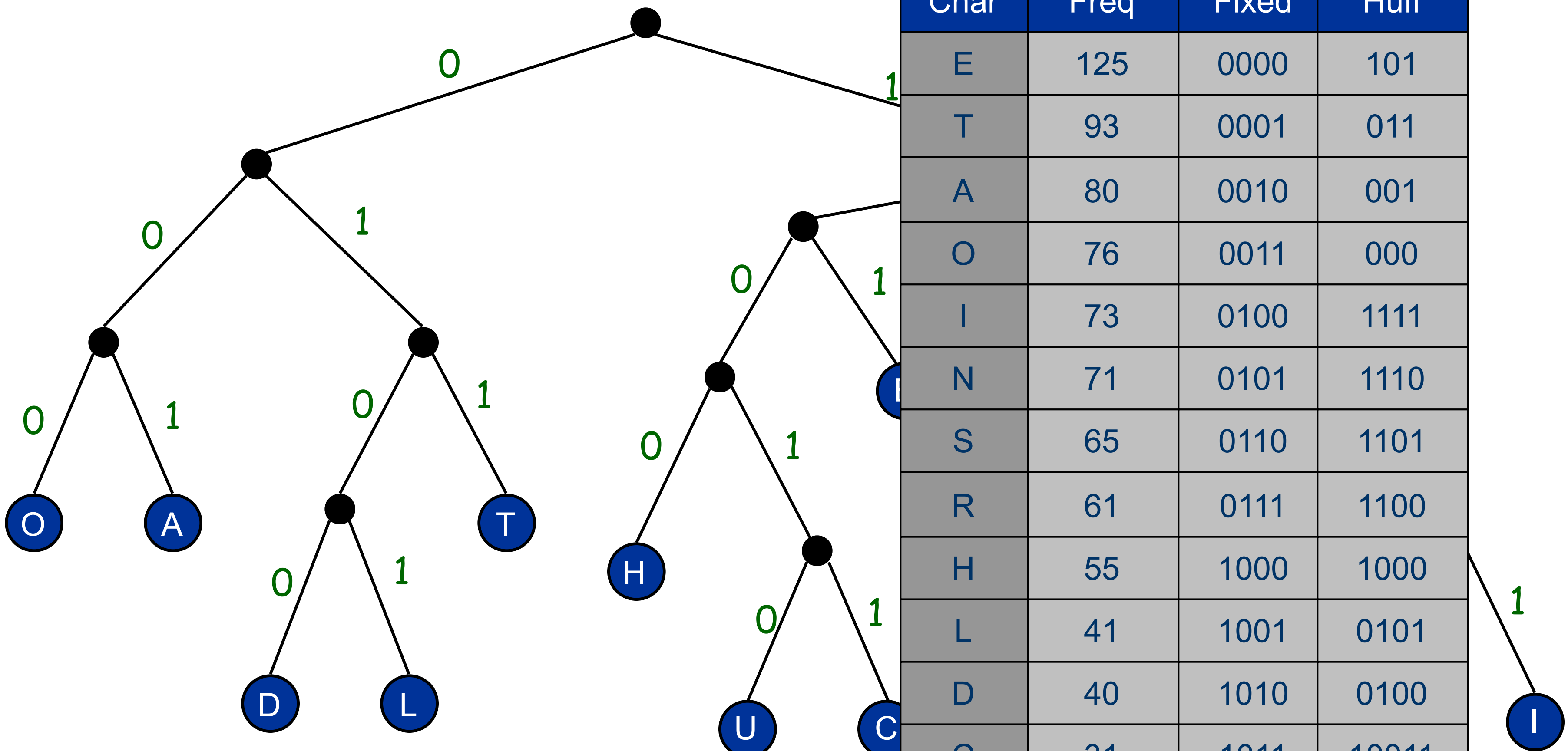
Huffman Code Construction

Char	Freq
	838

	508
	330



Huffman Code Construction



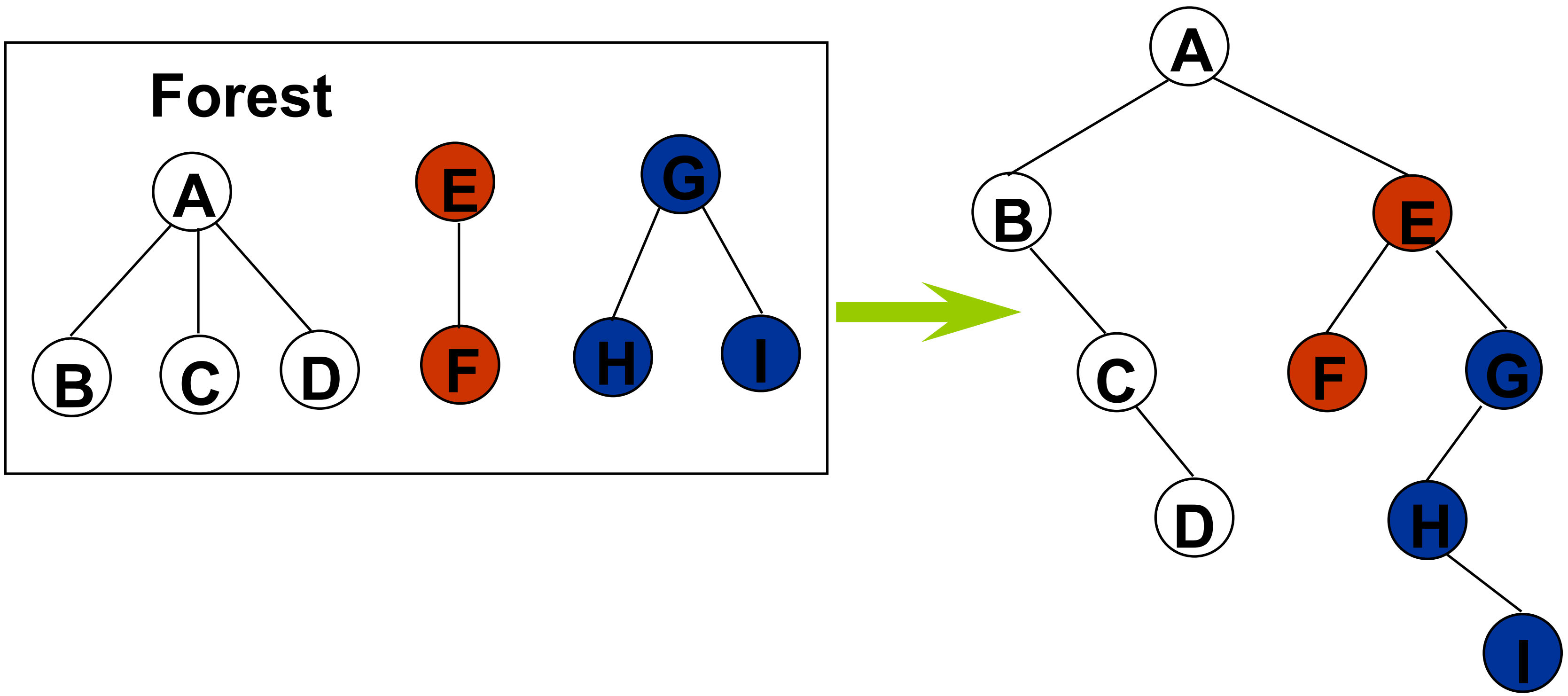
Char	Freq	Fixed	Huff
E	125	0000	101
T	93	0001	011
A	80	0010	001
O	76	0011	000
I	73	0100	1111
N	71	0101	1110
S	65	0110	1101
R	61	0111	1100
H	55	1000	1000
L	41	1001	0101
D	40	1010	0100
C	31	1011	10011
U	27	1100	10010
Total	838	4.00	3.62

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

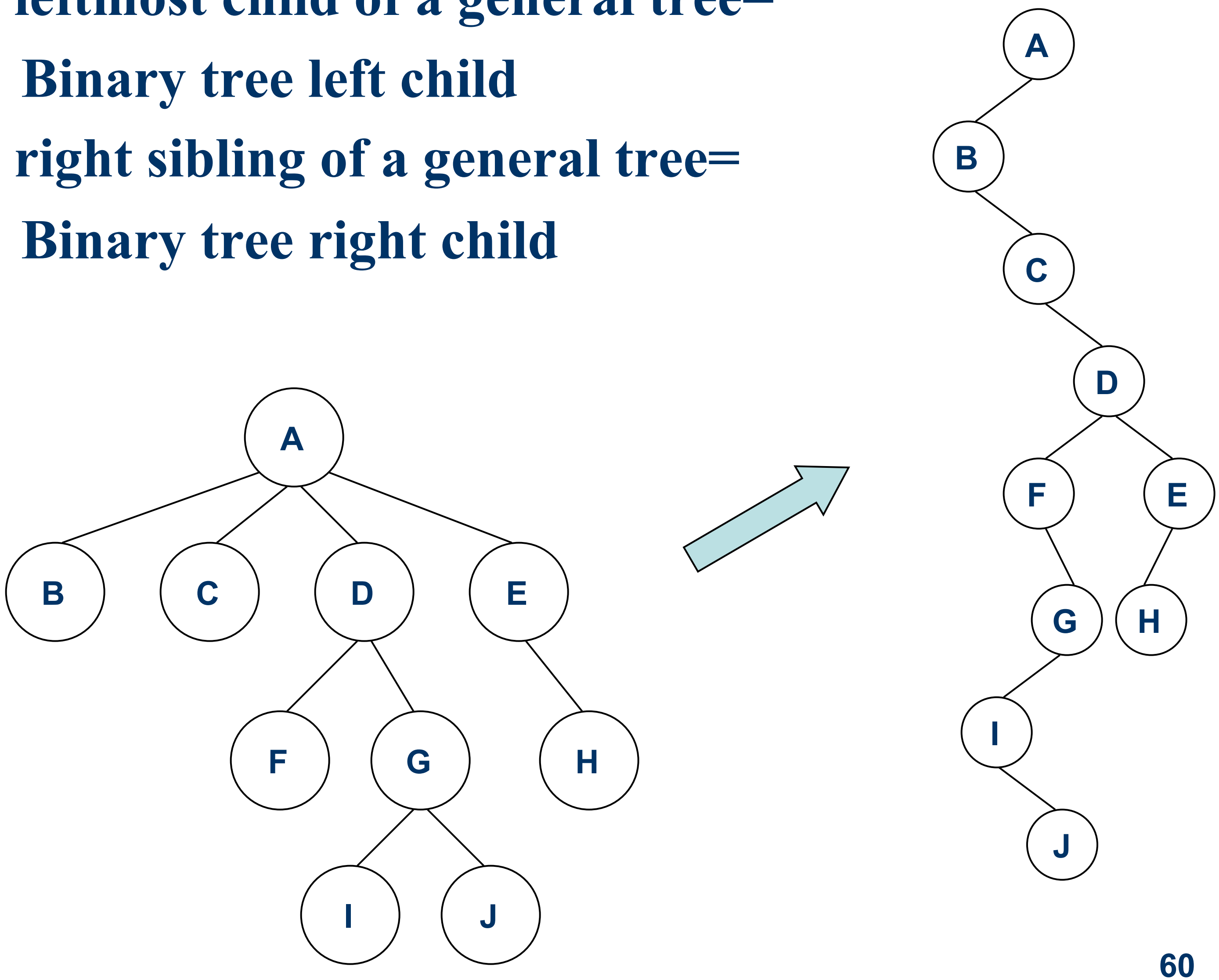
Forest

- **A forest is a set of n >= 0 disjoint trees**

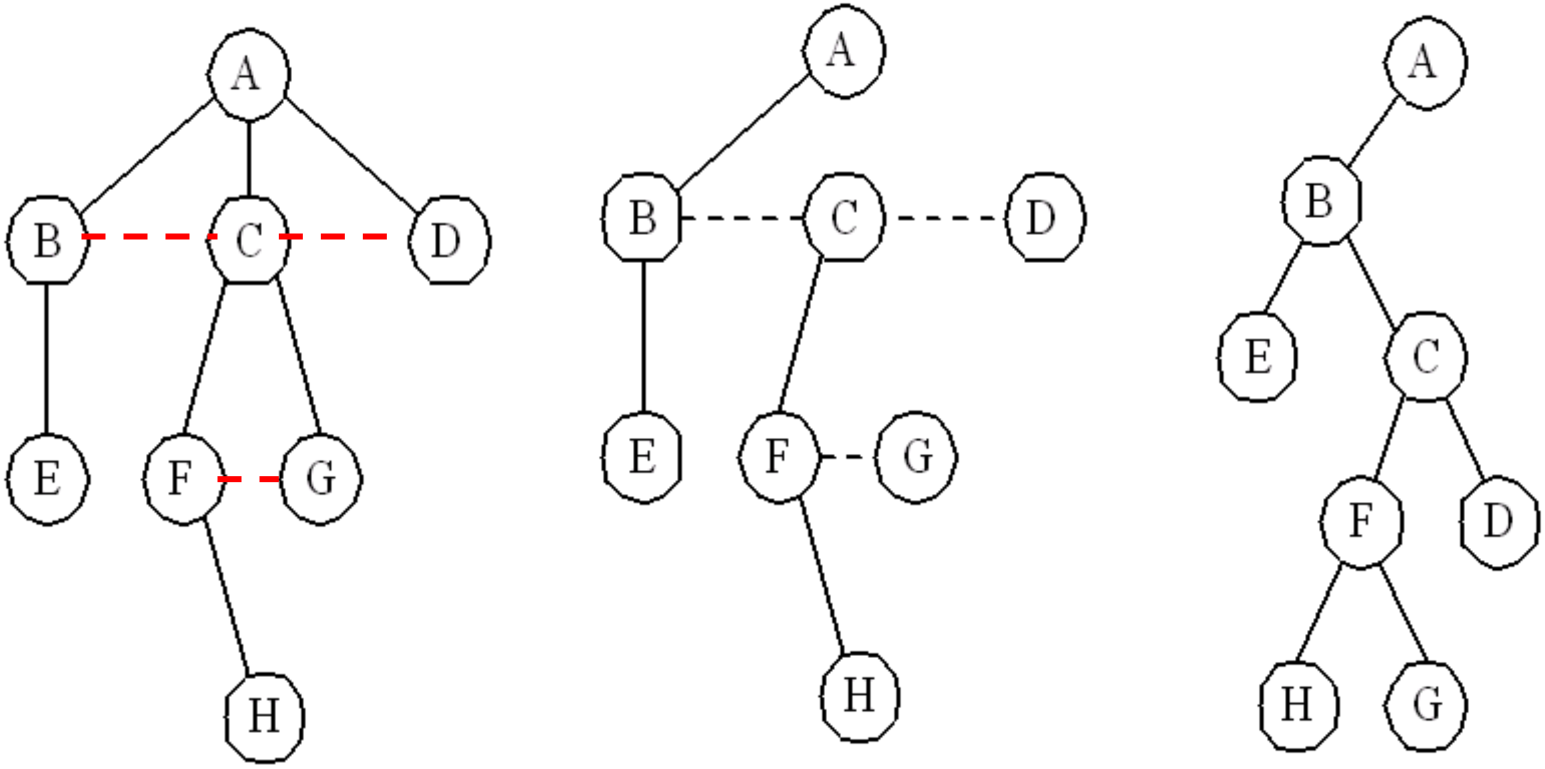


Converting a General Tree to a Binary Tree

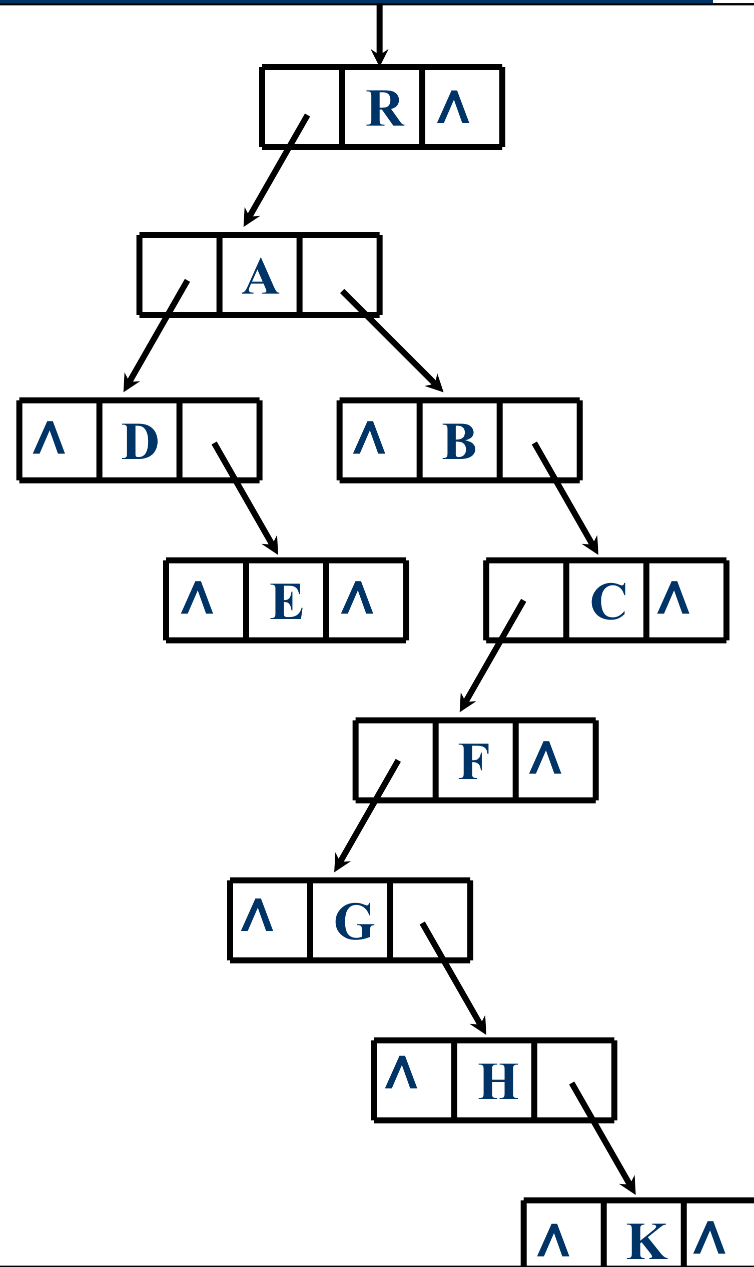
- leftmost child of a general tree=
  Binary tree left child
- right sibling of a general tree=
  Binary tree right child



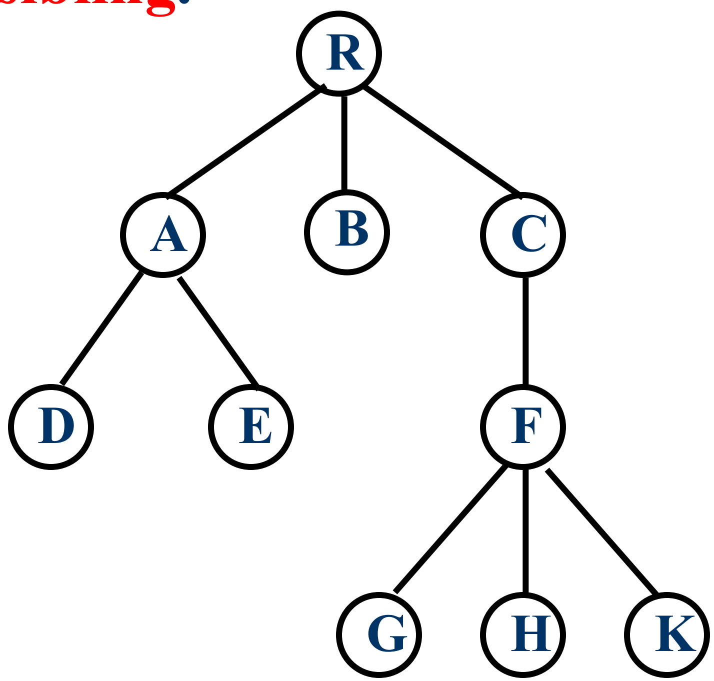
Converting a General Tree to a Binary Tree



Converting to a Binary Tree



Two pointers. One points to the first child, the other points to the right sibling.

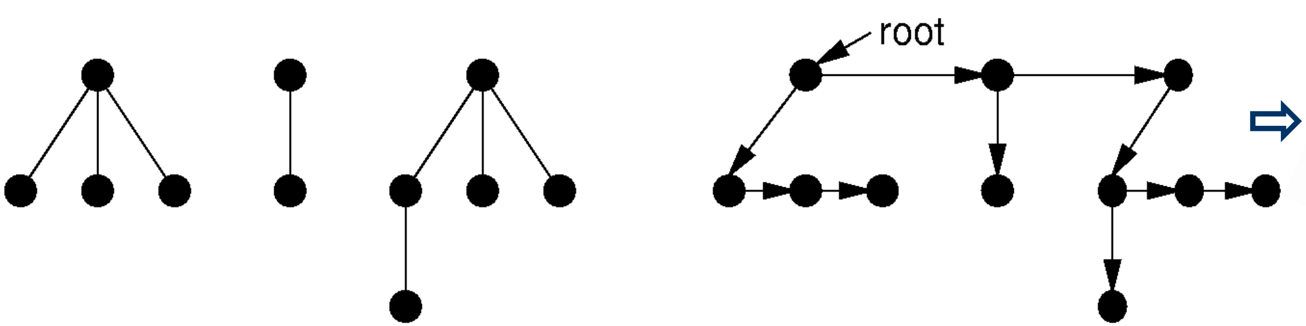


Converting Forest to a Binary Tree

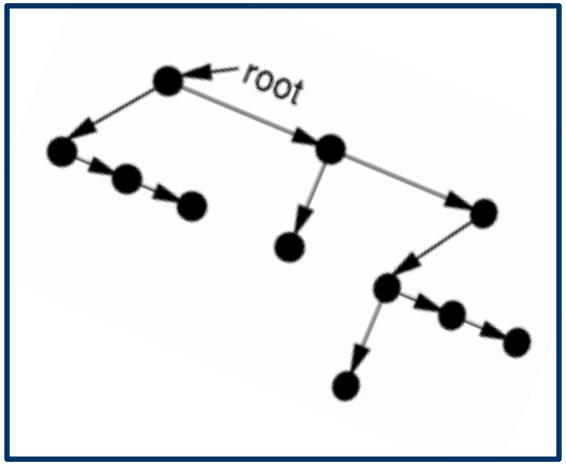
Left child/right sibling representation essentially stores a binary tree.

Use this process to convert any general tree to a binary tree.

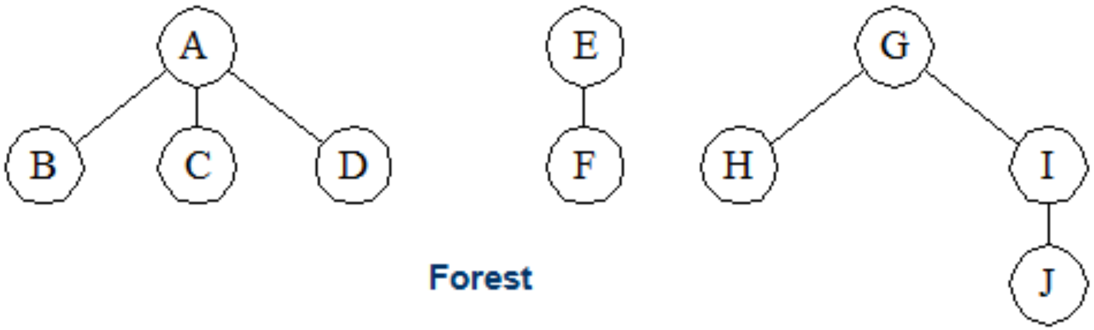
A forest is a collection of one or more general trees.



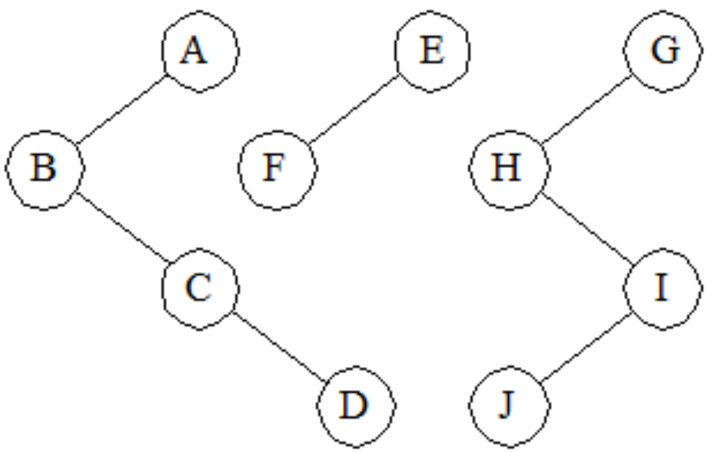
(a) (b)



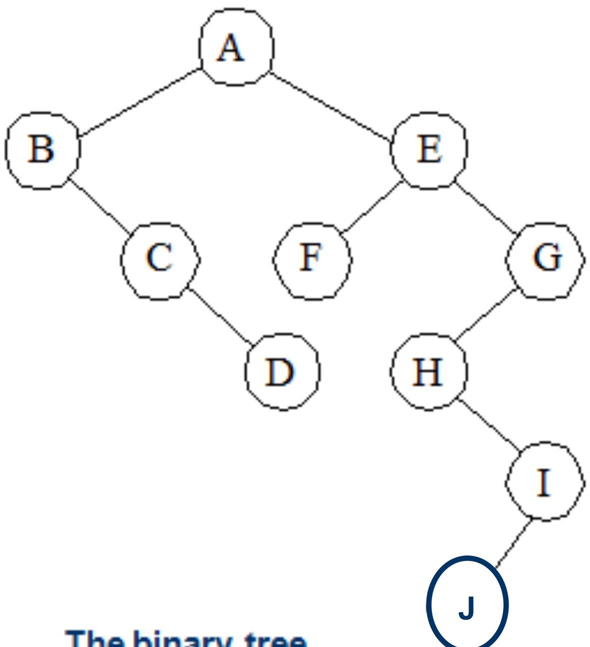
Converting Forest to a Binary Tree



Forest



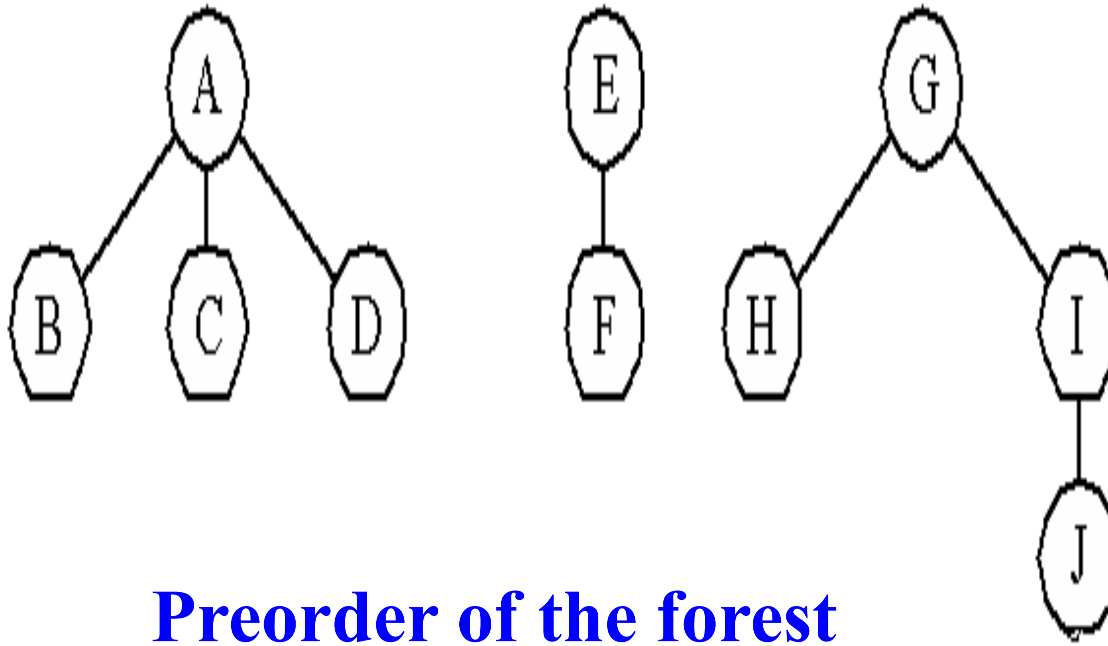
The corresponding binary tree



The binary tree

The traversal of forest

1） Preorder
If it is non-empty forest, the preorder of the forest is ：
①visit the root node of the first tree in the forest.
②visit its first tree in preoder.
③visit other trees in preorder.

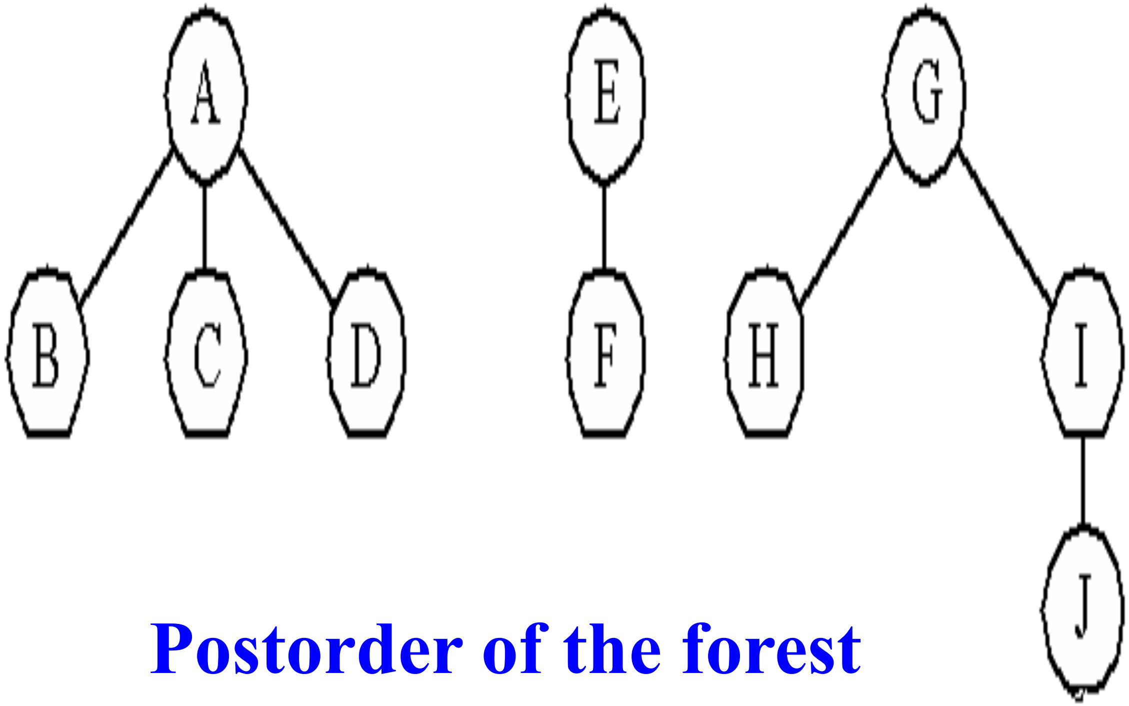


Preorder of the forest
ABCDEFGHIJ

Equivalent to the preorder traversal of the corresponding binary tree

The Traversal of Forest

2）Postorder
If it is non-empty forest, the postorder of the forest is：
① visit its first tree in postoder.
② visit the root node of the first tree in the forest.
③ visit other trees in postorder.



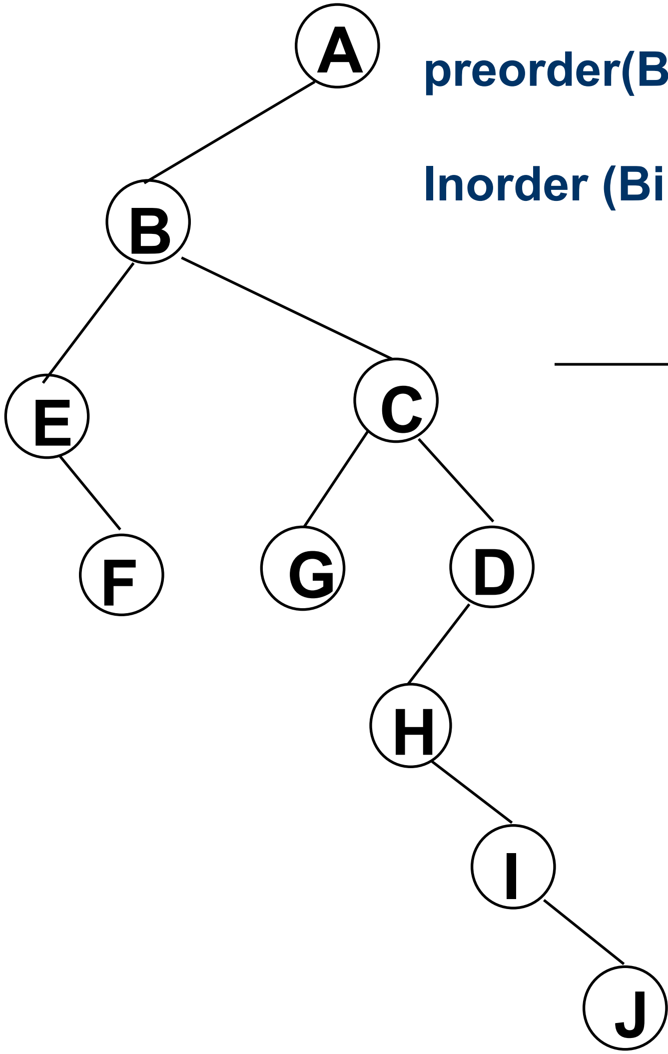
Postorder of the forest
BCDAFEHJIG

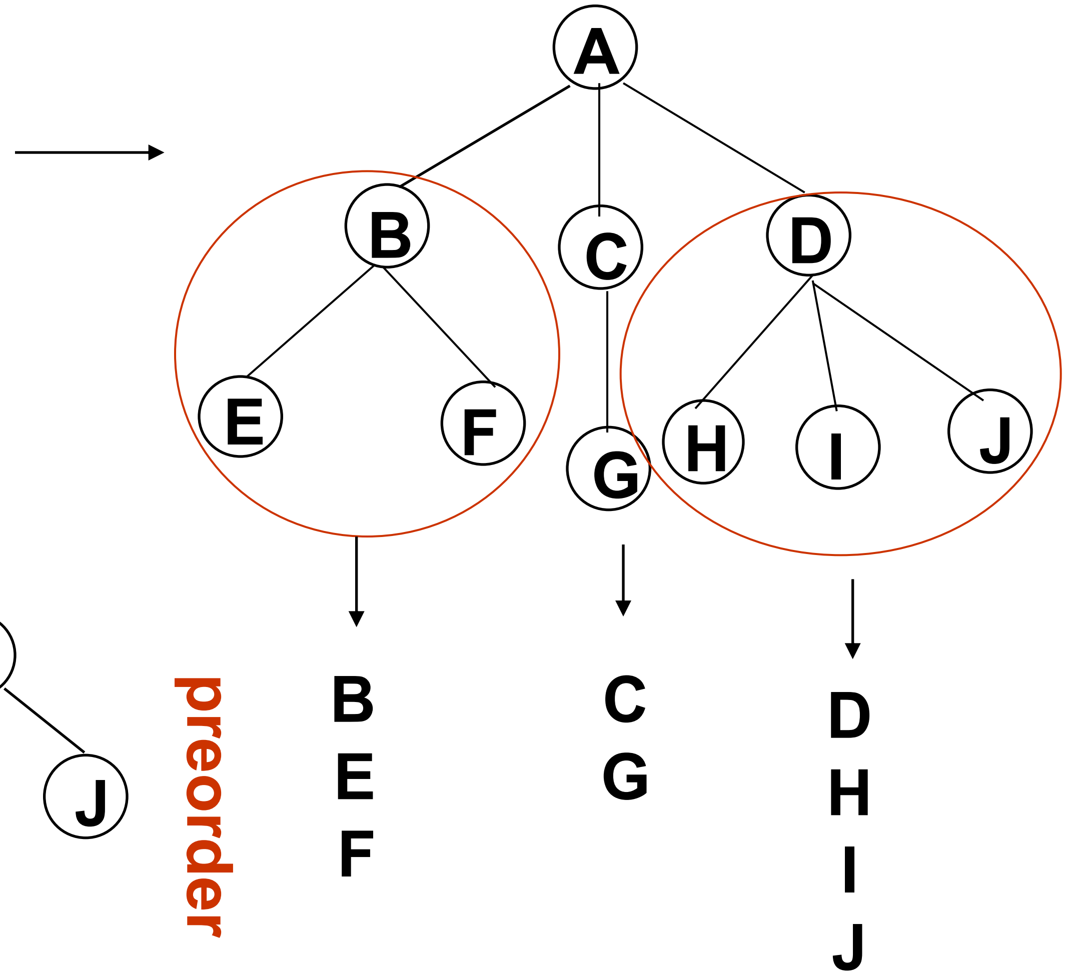
Equivalent to the inorder traversal of the corresponding binary tree

Preorder and Postorder Traversals of General Trees

preorder(Binary tree): ABEFCGDHIJ   Preorder(Gernaral Tree)

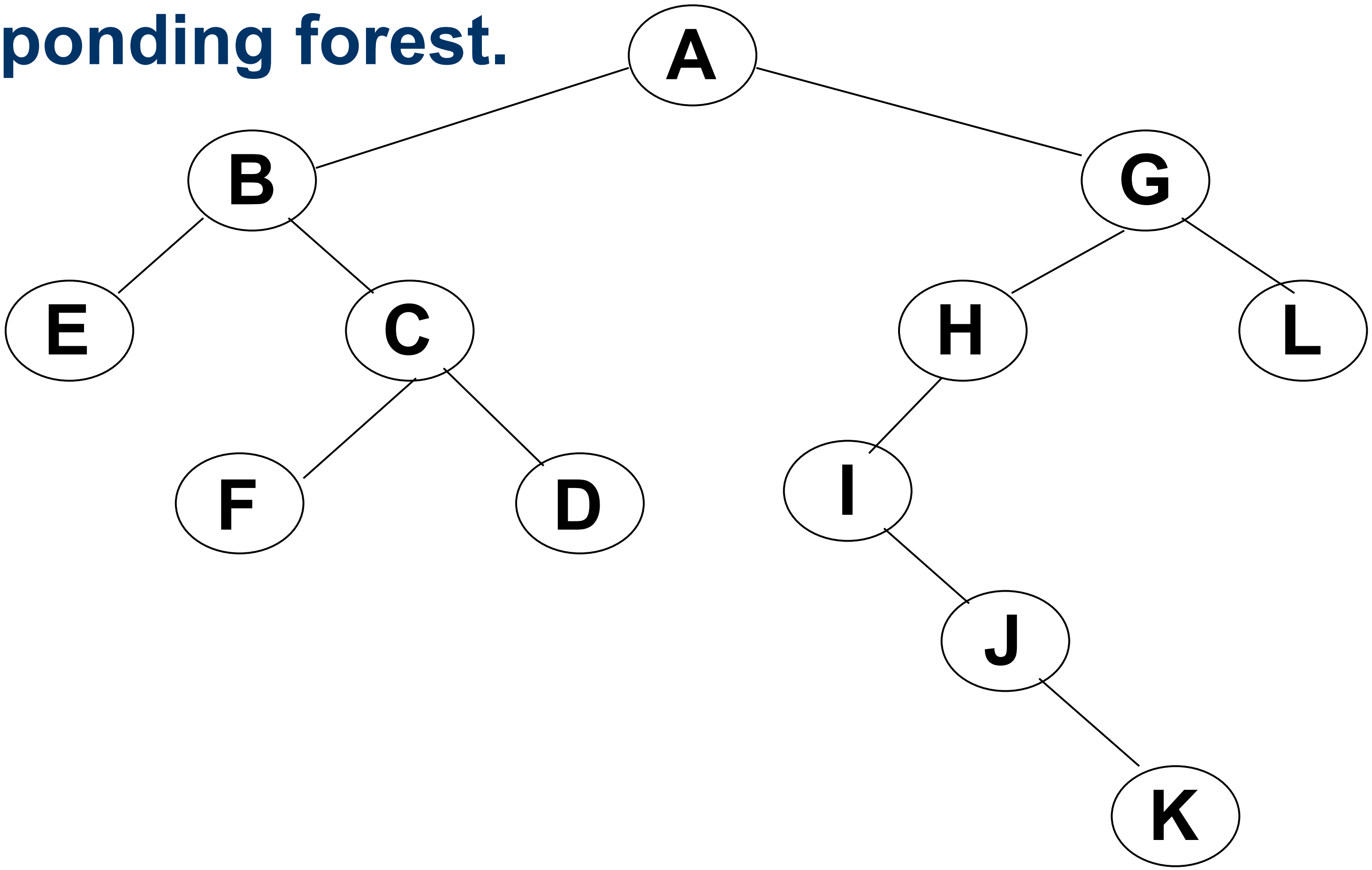
Inorder (Binary tree): EFBGCHIJDA Postorder(Gernaral Tree)





Exercise

First convert this binary tree to a forest, then give the preorder and postorder of the corresponding forest.



The Storage of The Tree

- Child representation
- Child-sibling representation
- Parent representation

The Storage of The Tree

- **(1) Child representation**
  - **Multi-linked list**
    - **Length fixed multi-linked list**

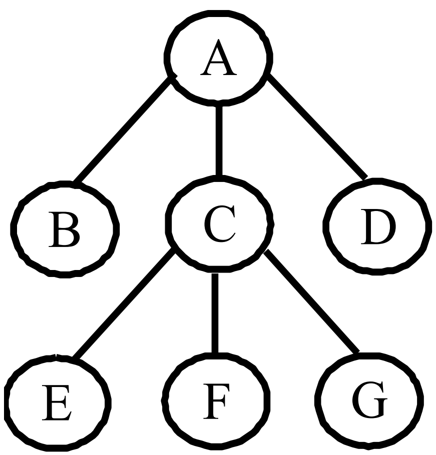
data	$child_1$	…	$child_n$

- **Length non-fixed multi-linked list**

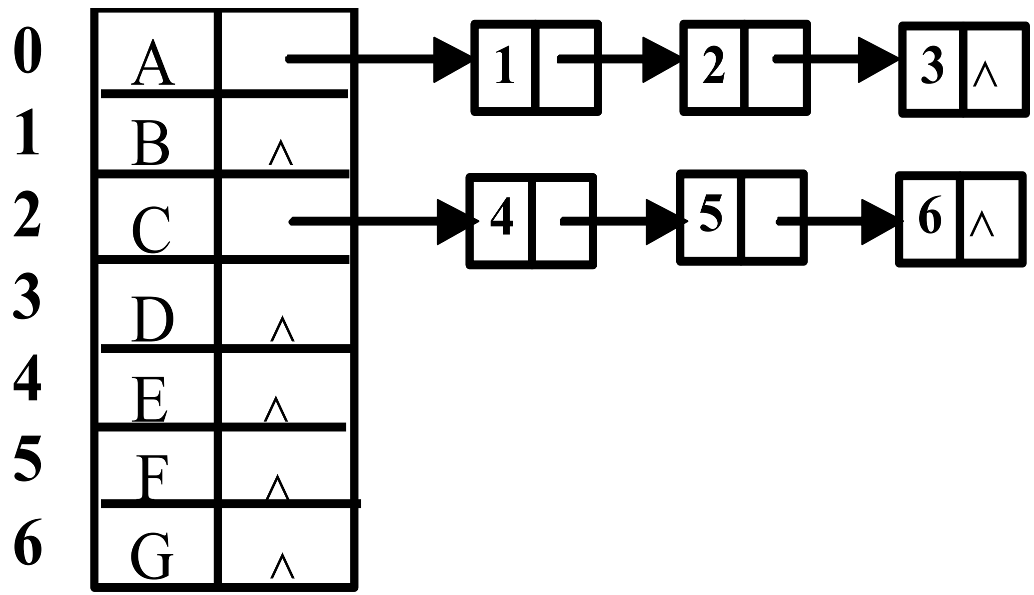
data	degree	$child_1$	…

The Storage of The Tree

(1) Children linked list



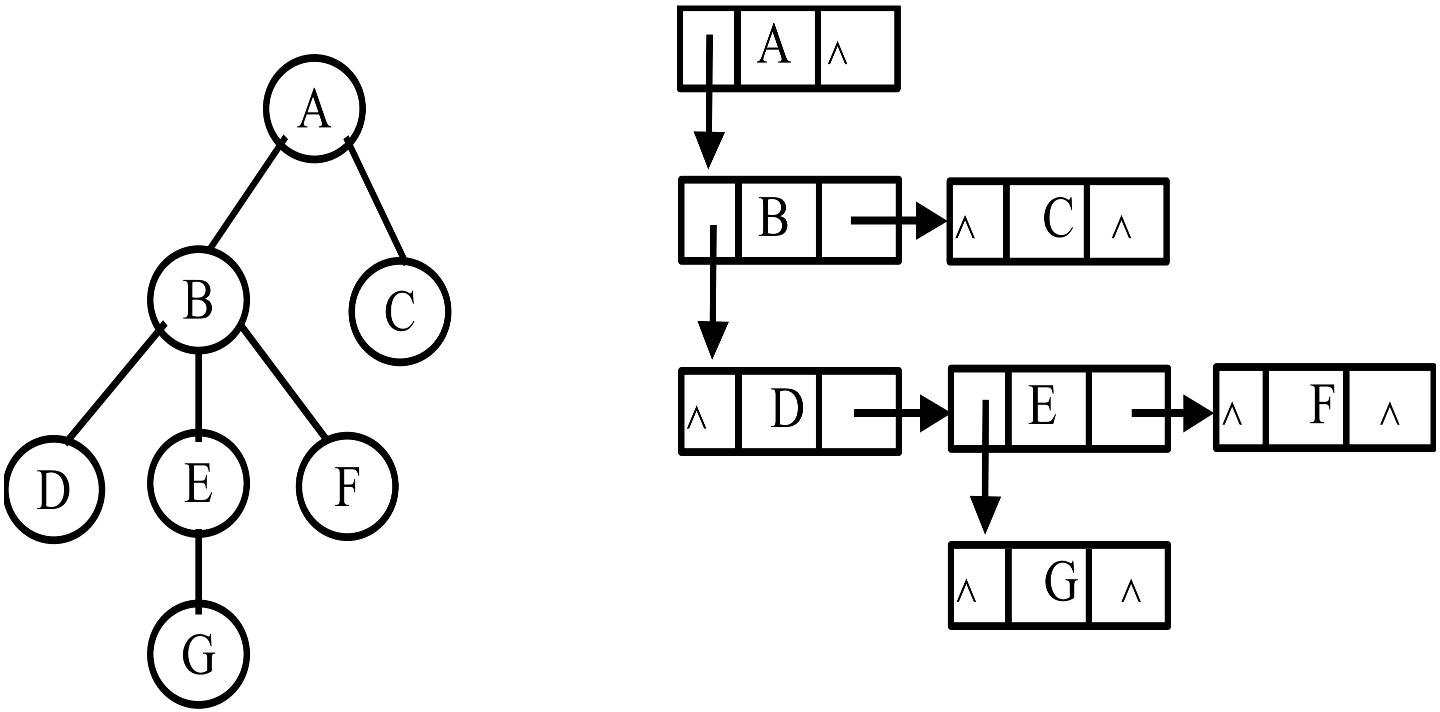
(a)



(b)

The Storage of The Tree

(2) child-sibling representation



(a) (b)

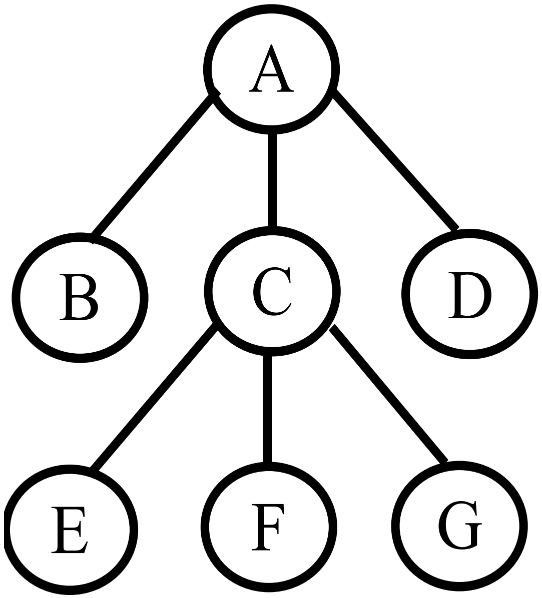
The Storage of The Tree

(3) Parent representation：

Utilize an Array-based structure for nodes in the tree, meanwhile add a pointer to show the position of its parent in the array.

Data	Parent

The Storage of The Tree



(a)

0	A	-1
1	B	0
2	C	0
3	D	0
4	E	2
5	F	2
6	G	2

(b)

Parent representation

Final Exam

- **TIME：2022.05.25 18:00-19:40**
- **LANGUAE : ENGLISH**
- **OFFLINE**

Final Exam

I. Multichoice (30 points)

15 questions (2 points / question)

II. Short Answers (60 points)

6-8questions（5-10 points/ question）

II. Programming (10 points)

1 question (pseudo code; detailed code)

Final Exam

Grade Distrubution：

- ✓ **Online Test: 5% (You are allowed to take each online test three times. The highest scores will be picked through 3 attempts)**
- ✓ **Online Homework: 5%**
- ✓ **Online Lab Assignment: 5%**
- ✓ **Offline Lab Assignment: 15%**
- ✓ **Final Exam   (paper based exam): 70%**

Final Exam

- **Multichoice questions**

Cover all topics talked in the class

- **Short answers**

Cover all topics talked in the class

- **Programming**

Cover all topics talked in the class

except AVL Tree implementation

Final Exam

- **Chapter 1**

  Algorithm Analysis

- **Chapter 2**

  List

  Stacks

  Queues

  String Matching

- **Chapter 3**

  Definitions Properties and Implementation of Binary Trees

  Binary Tree Traversals

  Binary Search Trees

  AVL Trees

  Heaps and Priority Queues

  Huffman Coding Trees

  General Trees and Forests

Homework

- Please refer to Icourse，Huawei Cloud.

- Due date for quiz: 23:30 2022/5/17
- Due date for homework: 23:30 2022/5/22
- Due data for online lab assignment: 2022/5/22 23：30