11.2 Applications of Trees

Problem:

- How should items in a list be stored so that an item can be easily located?

  Binary search trees

- What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?

  Decision trees

- How should a set of characters be efficiently coded by bit strings?

  Prefix codes

The Concept of Binary Search Trees

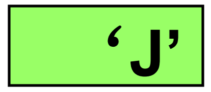
- A binary search tree can be used to store items in its vertices. It enables efficient searches.
- Binary search tree
  - An ordered rooted binary tree
  - Each vertex contains a distinct key value
  - The key values in the tree can be compared using "greater than" and "less than", and
  - The key value of each vertex in the tree is less than every key value in its right subtree, and greater than every key value in its left subtree.

Construct the binary search tree

- The shape of a binary search tree depends on its key values and their order of insertion.

For example, Insert the elements ‘J’ ‘E’ ‘F’ ‘T’ ‘A’ in that order.

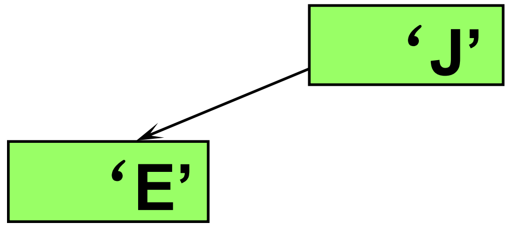
-- The first value to be inserted is put into the root.



-- Inserting ‘E’ into the BST

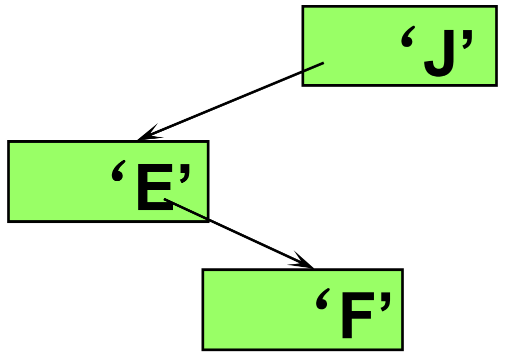
Thereafter, each value to be inserted begins by comparing itself to the value in the root, moving left it is less, or moving right if it is greater.

This continues at each level until it can be inserted as a new leaf.



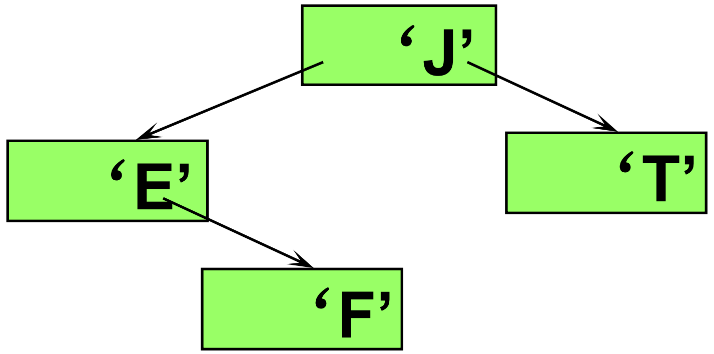
-- Inserting ‘F’ into the BST

Begin by comparing ‘F’ to the value in the root, moving left it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.



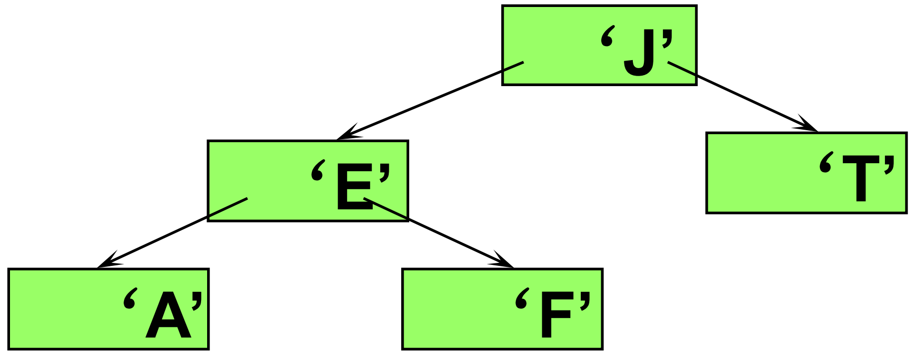
-- Inserting ‘T’ into the BST

Begin by comparing ‘T’ to the value in the root, moving left it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.



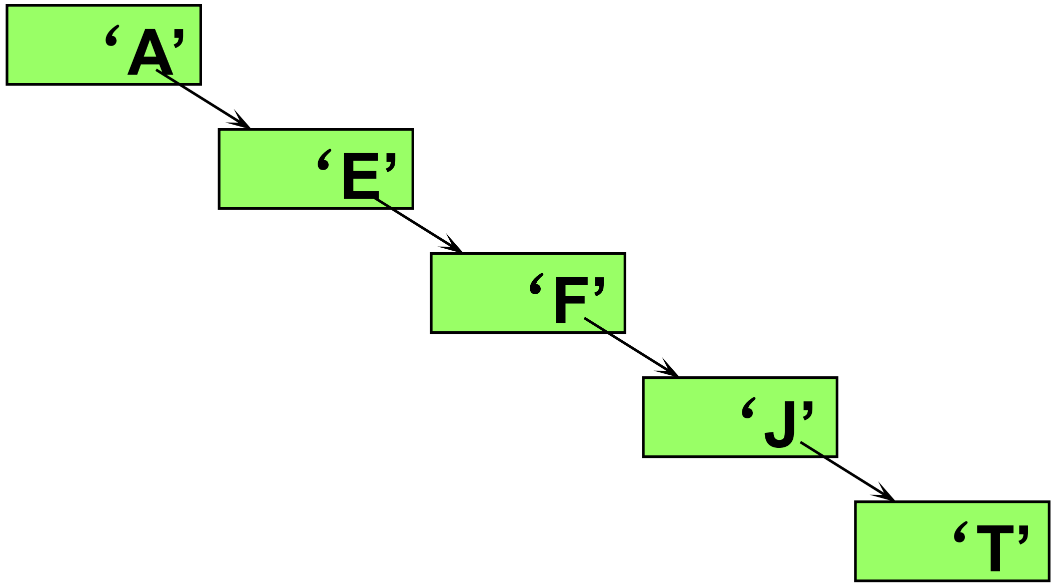
--Inserting ‘A’ into the BST

Begin by comparing ‘A’ to the value in the root, moving left if it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.

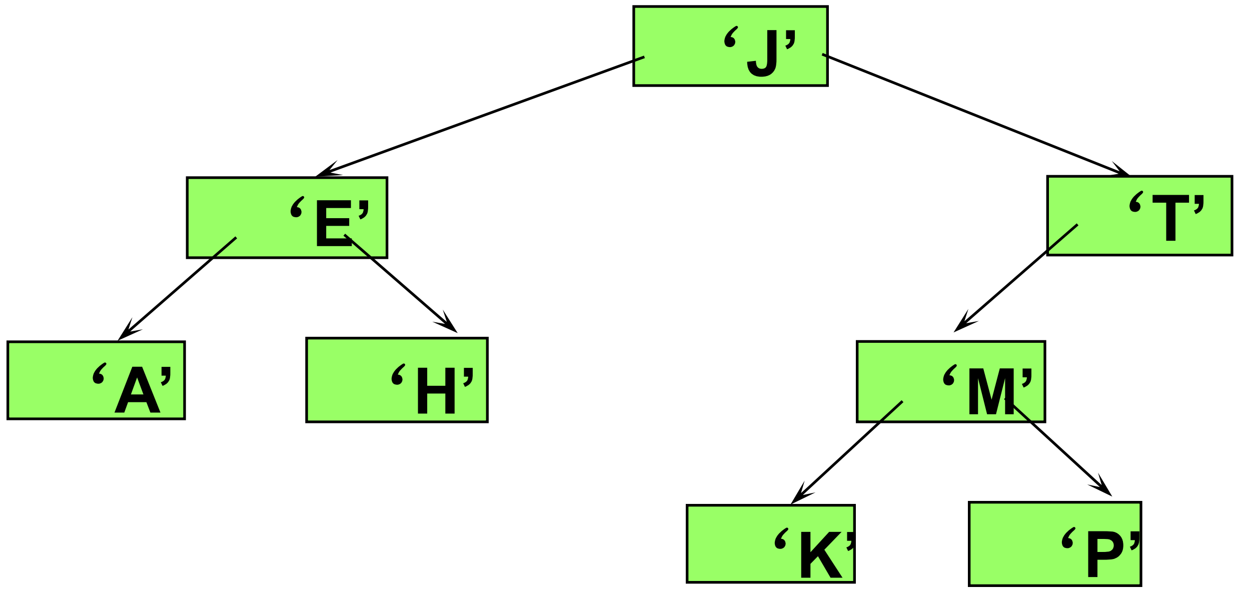


What binary search tree . . .

is obtained by inserting the elements ‘A’ ‘E’ ‘F’ ‘J’ ‘T’ in that order.



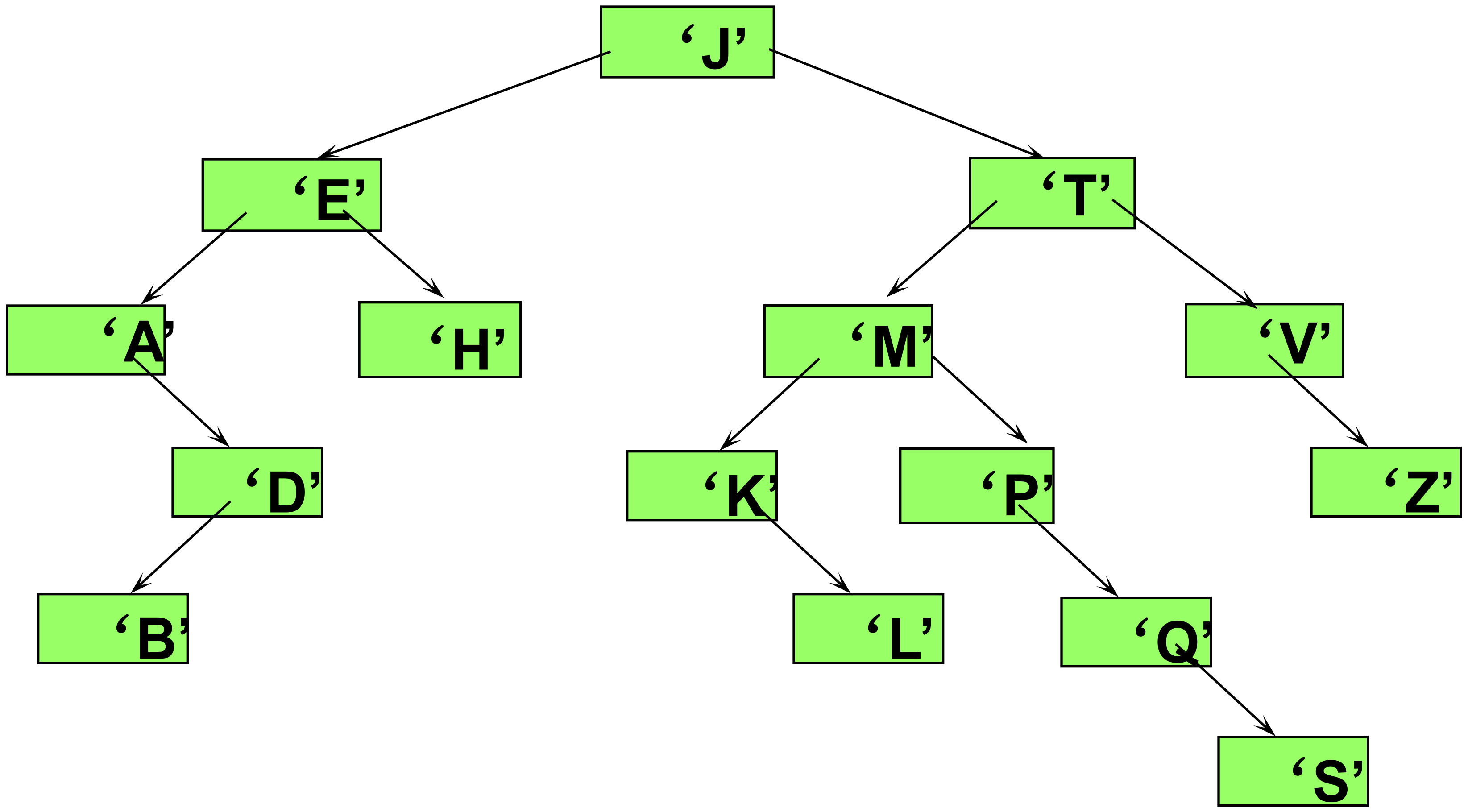
Another binary search tree



Add nodes containing these values in this order:

‘D’ ‘B’ ‘L’ ‘Q’ ‘S’ ‘V’ ‘Z’

Is ‘F’ in the binary search tree?



Binary Search Tree Algorithm

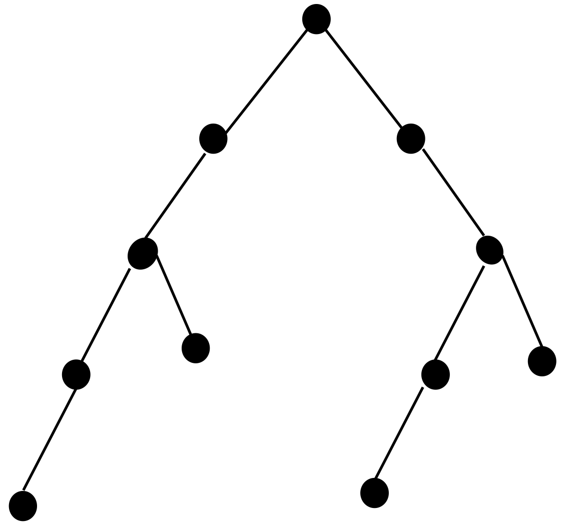
Algorithm 1 Binary Search Tree Algorithm

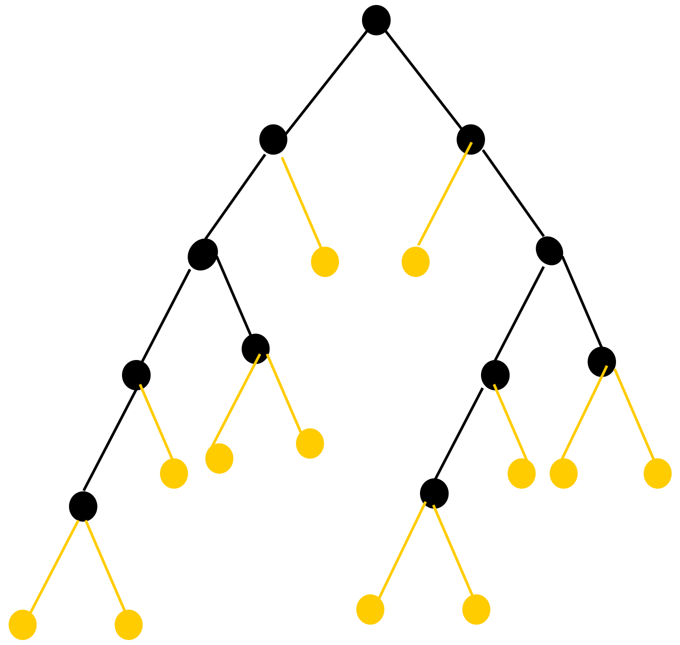
```
Procedure insertion ($T$: binary search tree, $x$: item)
$v$:=root of $T$
While $v \neq$null and label($v$) $\neq x$
 Begin
    if $x$<label($v$) then
        if left child of $v \neq$null then $v$:=left child of $v$
        else add new vertex as a left child of $v$ and set $v$:=null
    else
        if right child of $v \neq$null then $v$:=right child of $v$
        else add new vertex as a right child of $v$ and set $v$:=null
 end
 if root of $T$ =null then add a vertex $r$ to the tree and label it with $x$
  else if $v$ is null or label($v$) $\neq x$ then label new vertex with $x$ and let $v$ be this new
      vertex
 { $v$ = location of $x$}
```

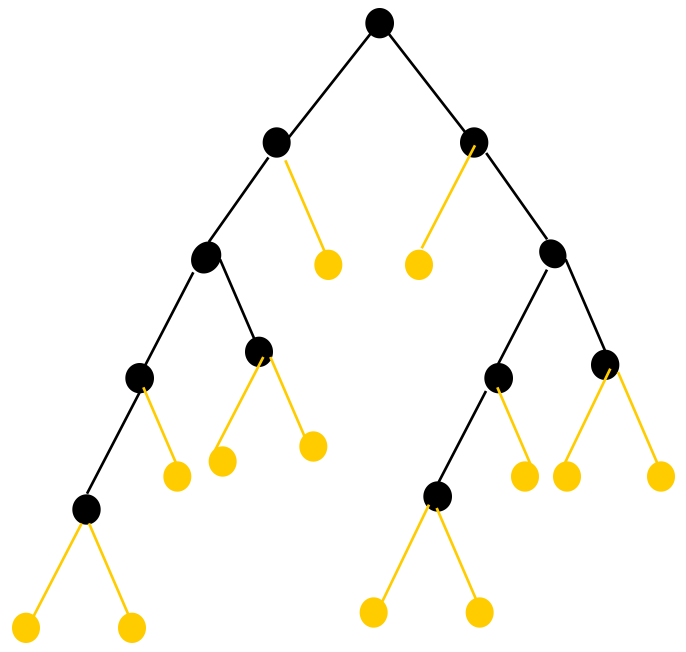
The computational complexity

Suppose we have a binary search tree *T* for a list of *n* items.

We can form a full binary tree *U* from *T* by adding unlabeled vertices whenever necessary so that every vertex with a key has two children.







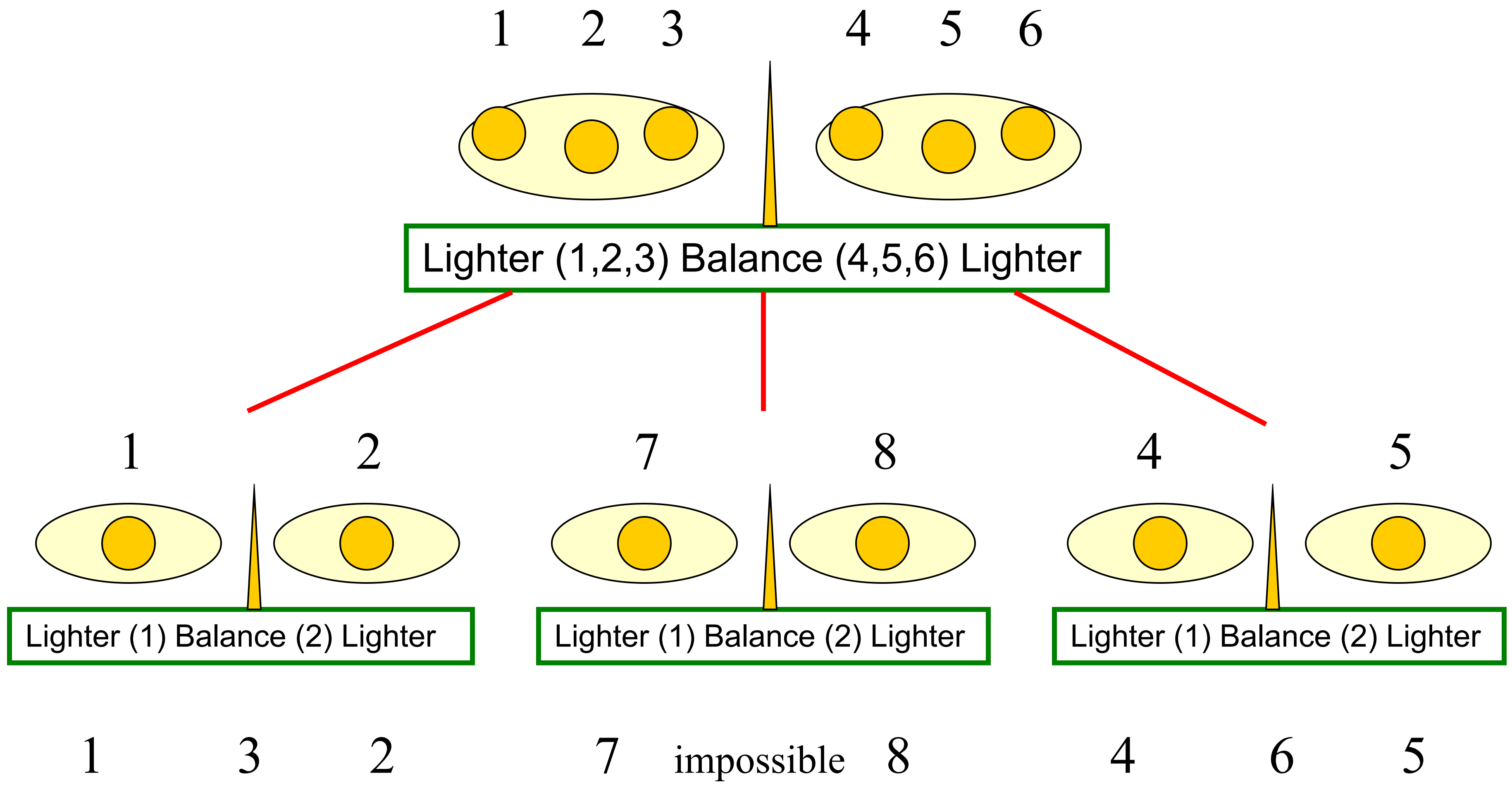
- **The most comparisons needed to add a new item is the length of the longest path in *U* from the root to a leaf.**
- **If a binary search tree is balanced, locating or adding an item requires no more than $\lceil \log(n+1) \rceil$ comparisons.**

Decision Trees

- Rooted trees can be used to model problems in which a series of decisions leads to a solution.

- A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a decision tree.

☸ An example: Counterfeit coin detection


1 2 3
4 5 6
Lighter (1,2,3) Balance (4,5,6) Lighter
1
2
7
8
4
5
Lighter (1) Balance (2) Lighter
Lighter (1) Balance (2) Lighter
Lighter (1) Balance (2) Lighter
1 3 2
7 impossible 8
4 6 5

Prefix Codes

- The problem of using bit strings to encode the letters of the English alphabet.

  How to improve coding efficiency?

- Using bit strings of different lengths to encode letters can improve coding efficiency.

  How to ensure the code having the definite meaning?

  For example, e: 0 a: 1 t: 01

  0101: eat, tea, eaea, tt?

- When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end.

the Concept of Prefix Codes

- To ensure that no bit string corresponds to more than one sequence of letters, the bit string for a letter must never occur as the first part of the bit string for another letter. Codes with this property are called prefix codes.

For example,

e: 0	**a: 10**	**t: 110**

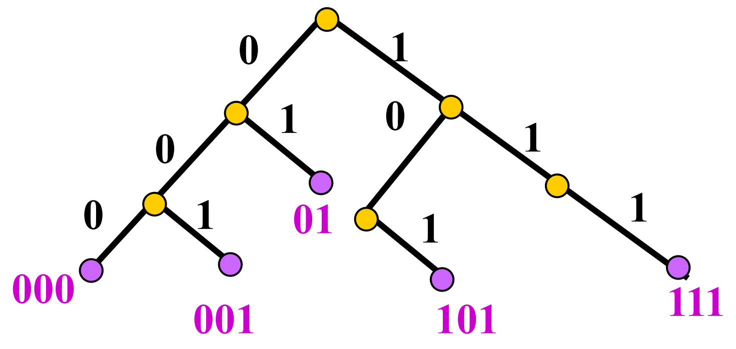
How to Construct Prefix Codes

- Using a binary tree.

  -- the left edge at each internal vertex is labeled by 0.

  -- the right edge at each internal vertex is labeled by 1.

  -- the leaves are labeled by characters which are encoded with the bit string constructed using the labels of the edges in the unique path from the root to the leaves.



Problem: How to produce efficient codes based on the frequencies of occurrences of characters?

For example,

Character	*a*	*b*	*c*	*d*	*e*	……
Frequences (w_i)	82	14	28	38	131	……

$$obj.\ \ \min\left(\sum_{i=1}^{26} l_i w_i\right)$$

where l_i is the length of prefix codes for characters i.

General problem: Tree T has t leaves, w_1, w_2,…, w_t are weights, l_i=$l(w_i)$. Let the weight of tree T be

$$w(T) = \sum_{i=1}^{t} l_i w_i$$

$$obj.\ \ \min(w(T))$$

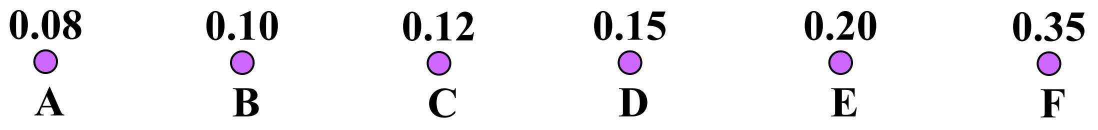
Huffman Coding

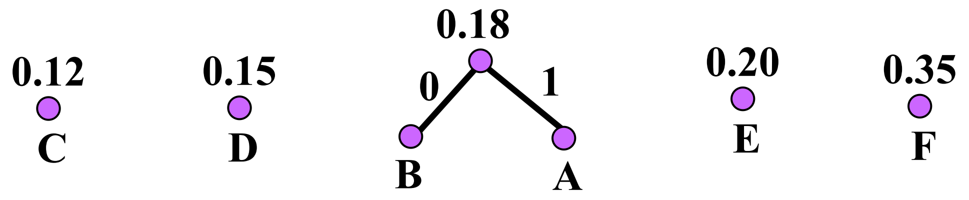
Algorithm 2 Huffman Coding.

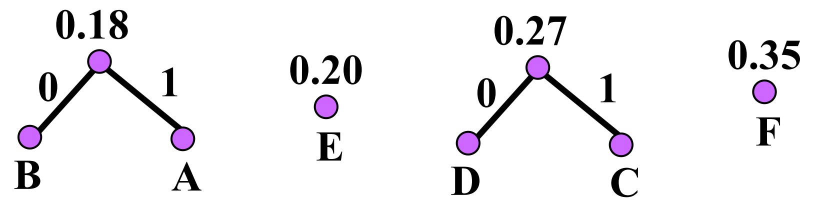
```
Procedure Huffman ($C$: symbols $a_i$ with frequencies $w_i$, $i$=1, …, $n$)
$F$:=forest of $n$ rooted trees, each consisting of the single vertex $a_i$
    and assigned weight $w_i$
While $F$ is not a tree
 begin
    Replace the rooted trees $T$ and $T'$ of least weights from $F$ with
    $w(T) \geq w(T')$ with a tree having a new root that has $T$ as its left
    subtree and $T'$ as its right subtree. Label the new edge to $T$
    with 0 and the new edge to $T'$ with 1.
    Assign $w(T) + w(T')$ as the weight of the new tree.
end
 {The Huffman coding for the symbol $a_i$ is the concatenation of
    the labels of the edges in the unique path from the root to the
    vertex $a_i$}
```

〖Example 1〗 **Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?**

Solution:

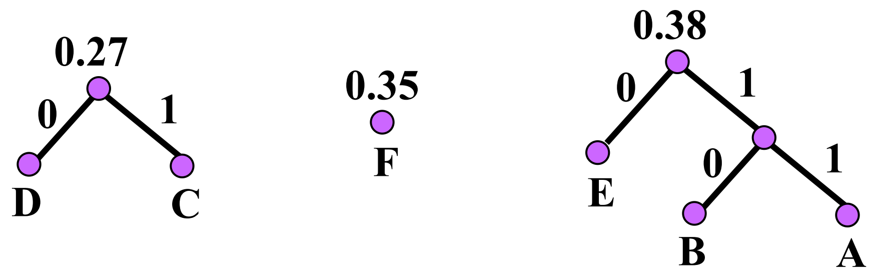




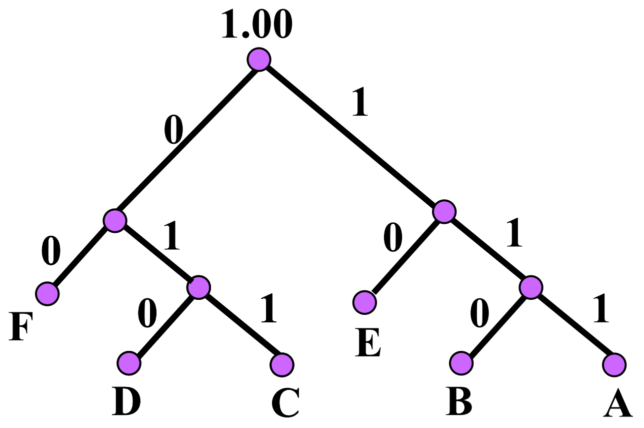


〖Example 1〗 **Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?**

Solution:

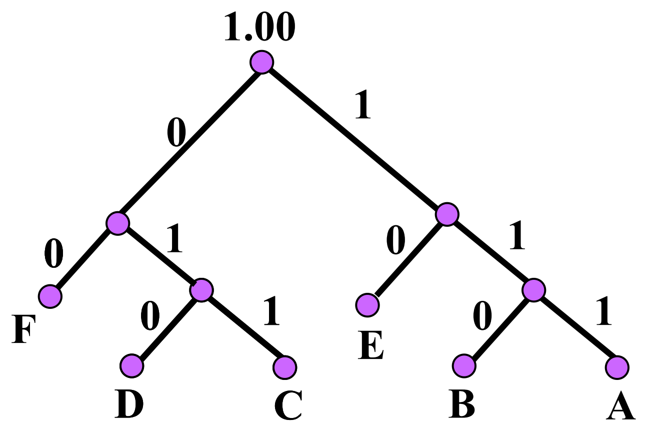


……



〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: *A*:0.08, *B*:0.10, *C*:0.12, *D*:0.15, *E*:0.20, *F*:0.35. What is the average number of bits used to encode a character?

Solution:



The encoding produced encodes *A* by 111, *B* by 110, *C* by 011, *D* by 010, *E* by 10, and *F* by 00.

The average number of bits used to encode a symbol using this encoding is

$3 \cdot 0.08 + 3 \cdot 0.10 + 3 \cdot 0.12 + 3 \cdot 0.15 + 2 \cdot 0.20 + 2 \cdot 0.35 = 2.45.$

Homework:

SE: P. 769 1, 20, 23

EE: P. 805 1, 20, 23