بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۱۴

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

Maps

- A map models a searchable collection of key-value entries
- The main operations of a map are for searching, inserting, and deleting items
- Multiple entries with the same key are not allowed (GTM)
- Applications:
  - address book
  - student-record database

Map ADT

- Data:
  - (key, value) *pairs*
- Methods:
  - **insert(key,value)**
  - **find(key)**
  - **delete(key)**

پیاده‌سازی با آرایه و لیست پیوندی

- برای یک map یا دیکشنری با n جفت (key, value) (بدون نیاز به چک کردن تکراری بودن)

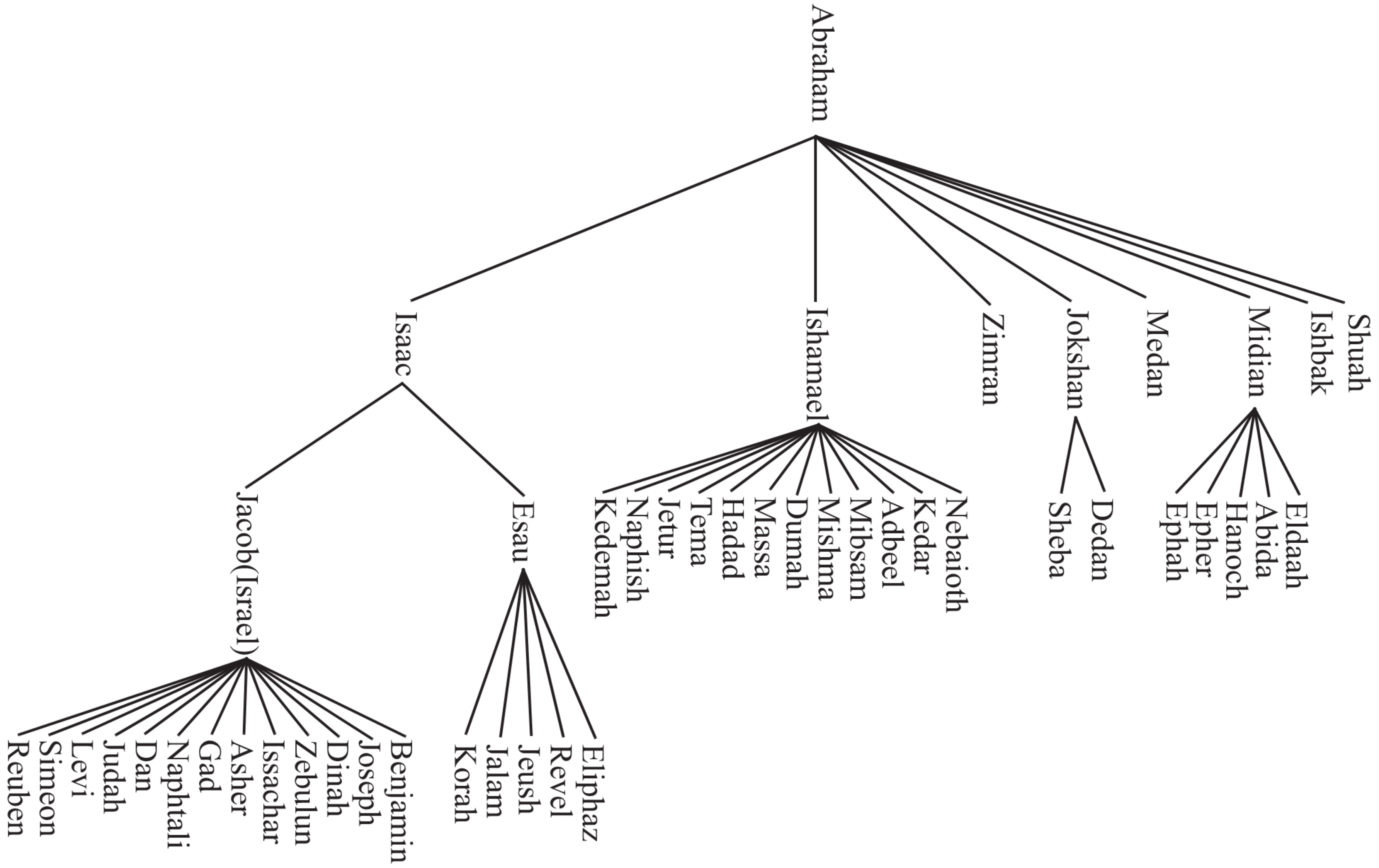
	`insert`	`find`	`delete`
Unsorted linked-list	*O(1)*	*O(n)*	*O(n)*
Unsorted array	*O(1)*	*O(n)*	*O(n)*
Sorted linked list	*O(n)*	*O(n)*	*O(n)*
Sorted array	*O(n)*	*O(logn)*	*O(n)*

درخت

- یک ساختمان داده غیرخطی (مناسب برای جستجو)
- مناسب برای روابط سلسله مراتبی

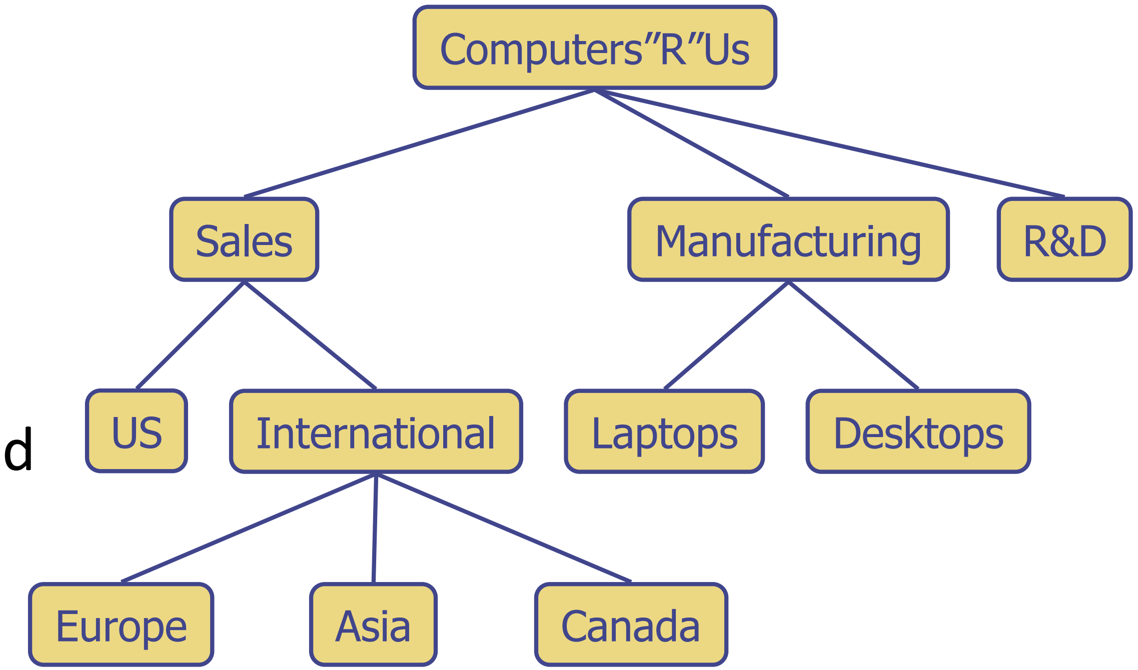
در ادامه...

What is a Tree?



What is a Tree?

- A graph without cycles
- In software systems, a tree is an abstract model of a hierarchical structure
  - Compared with “linear” data structures
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments

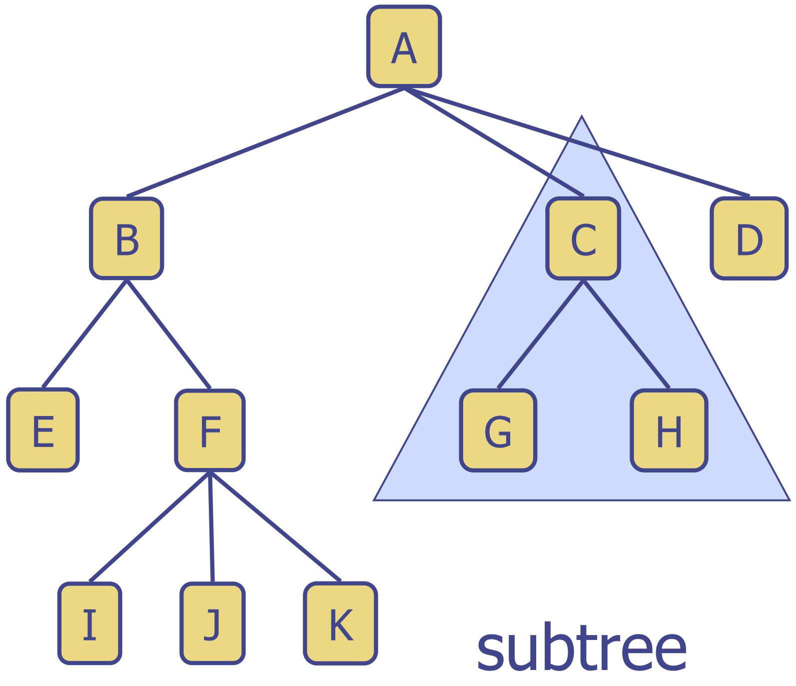


Tree Terminology

IUT-ECE

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.

- Subtree: tree consisting of a node and its descendants



Tree Terminology

- A tree is ordered if there is a linear ordering defined for the children of each node; that is, we can identify children of a node as being the first, second, third, and so on.

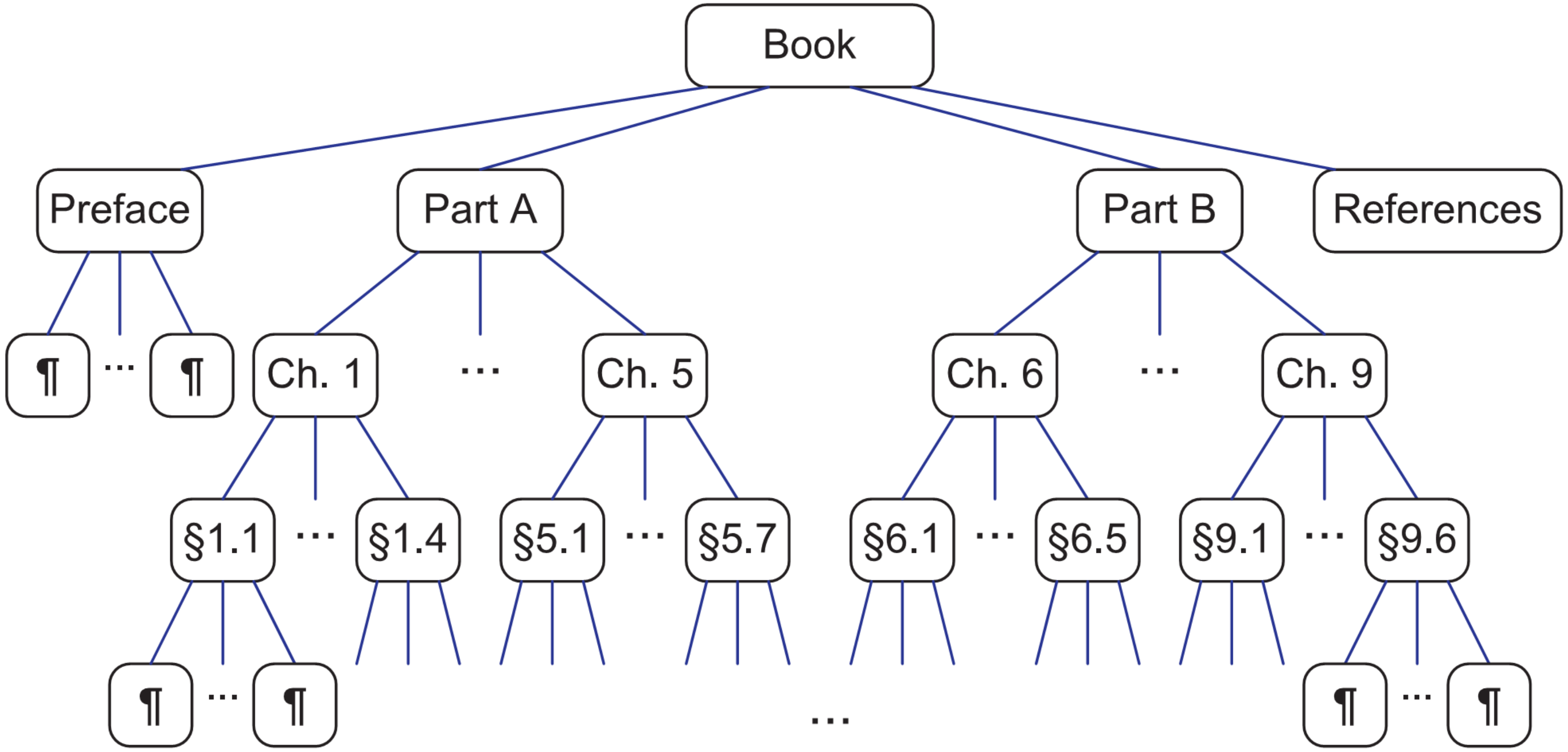
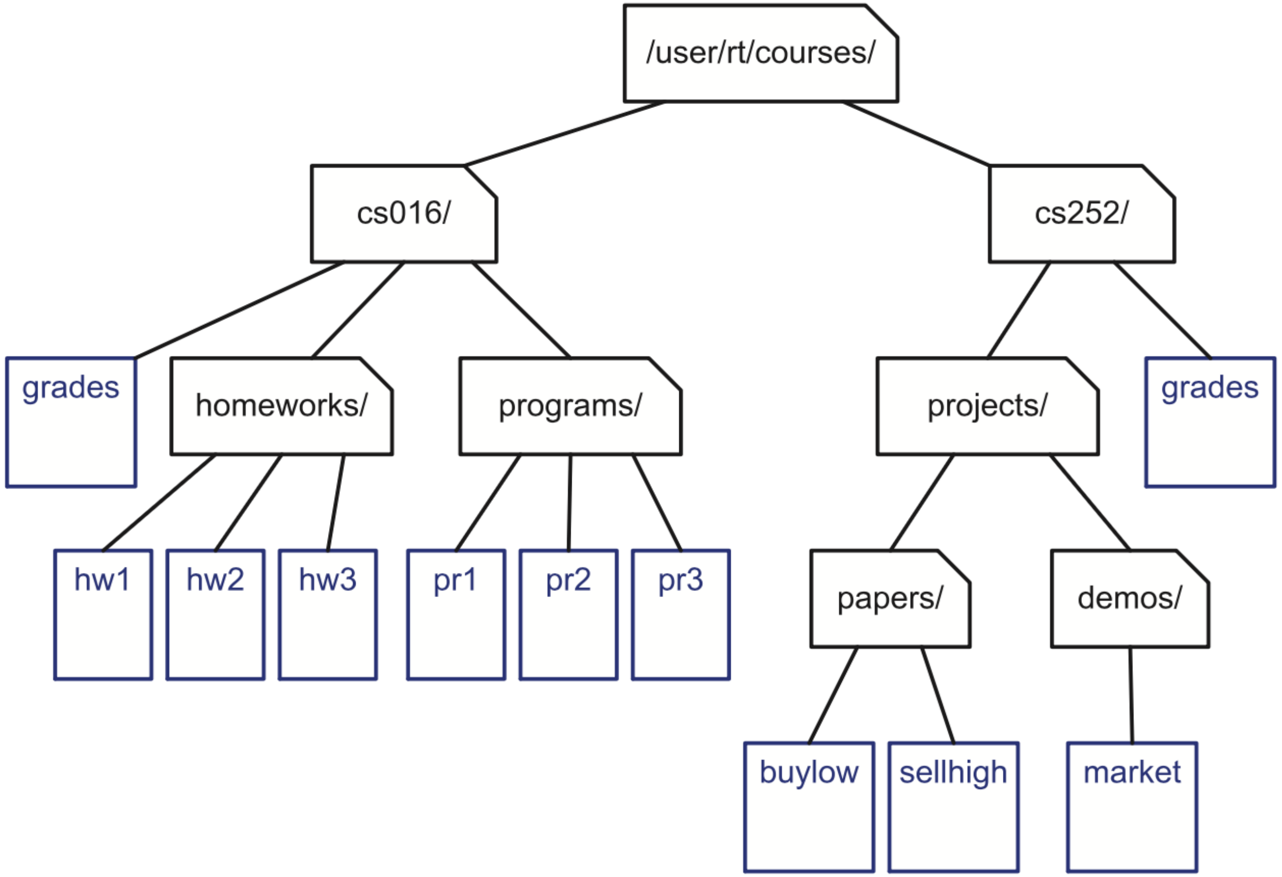


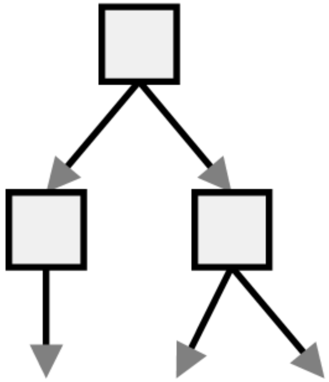
Figure 7.4: An ordered tree associated with a book.

Example (unordered tree): File System



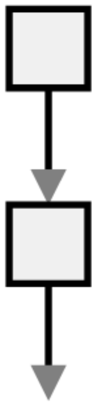
مثال

- درخت؟



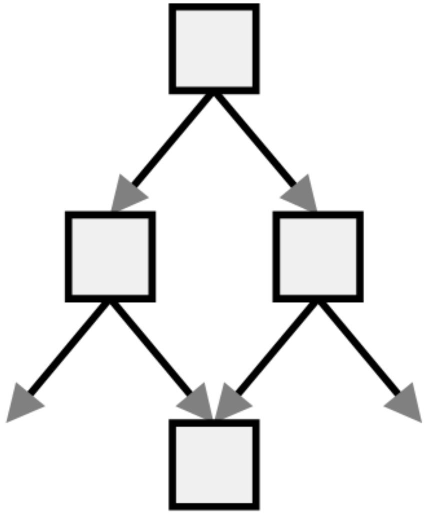
مثال

- درخت؟



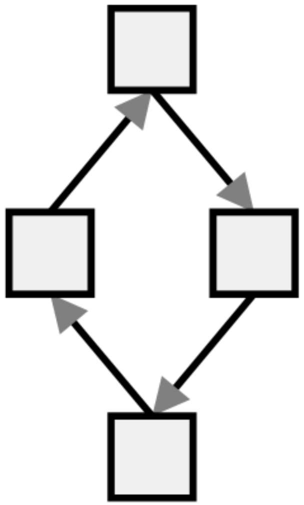
مثال

- درخت؟

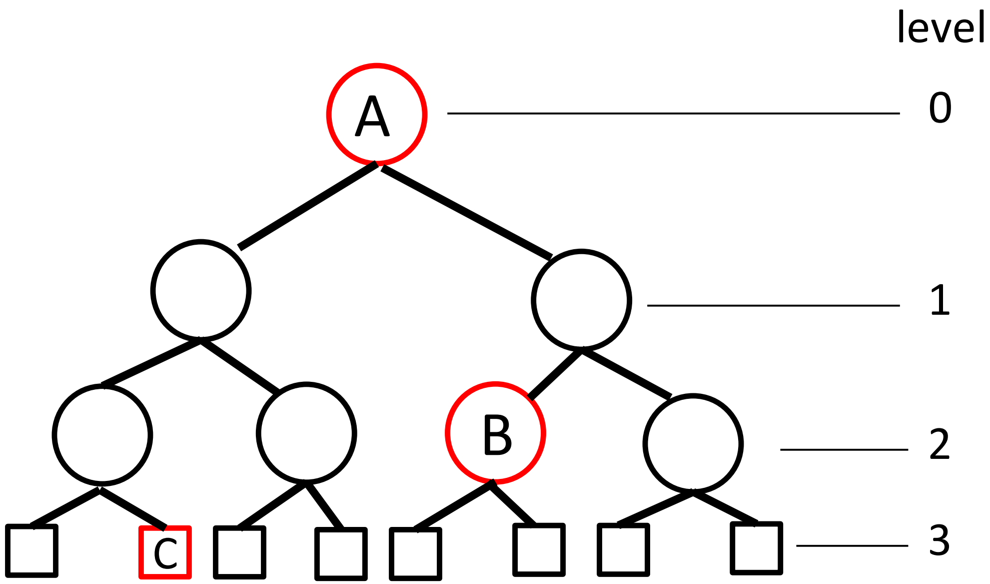


مثال

- درخت؟



مثال



Node	level	depth	height
A	0	0	3
B	2	2	1
C	3	3	0

Tree ADT

- We can use positions to abstract nodes

- Generic methods:
  - integer size()
  - boolean empty()

- Accessor methods:
  - position root()
  - list<position> positions()

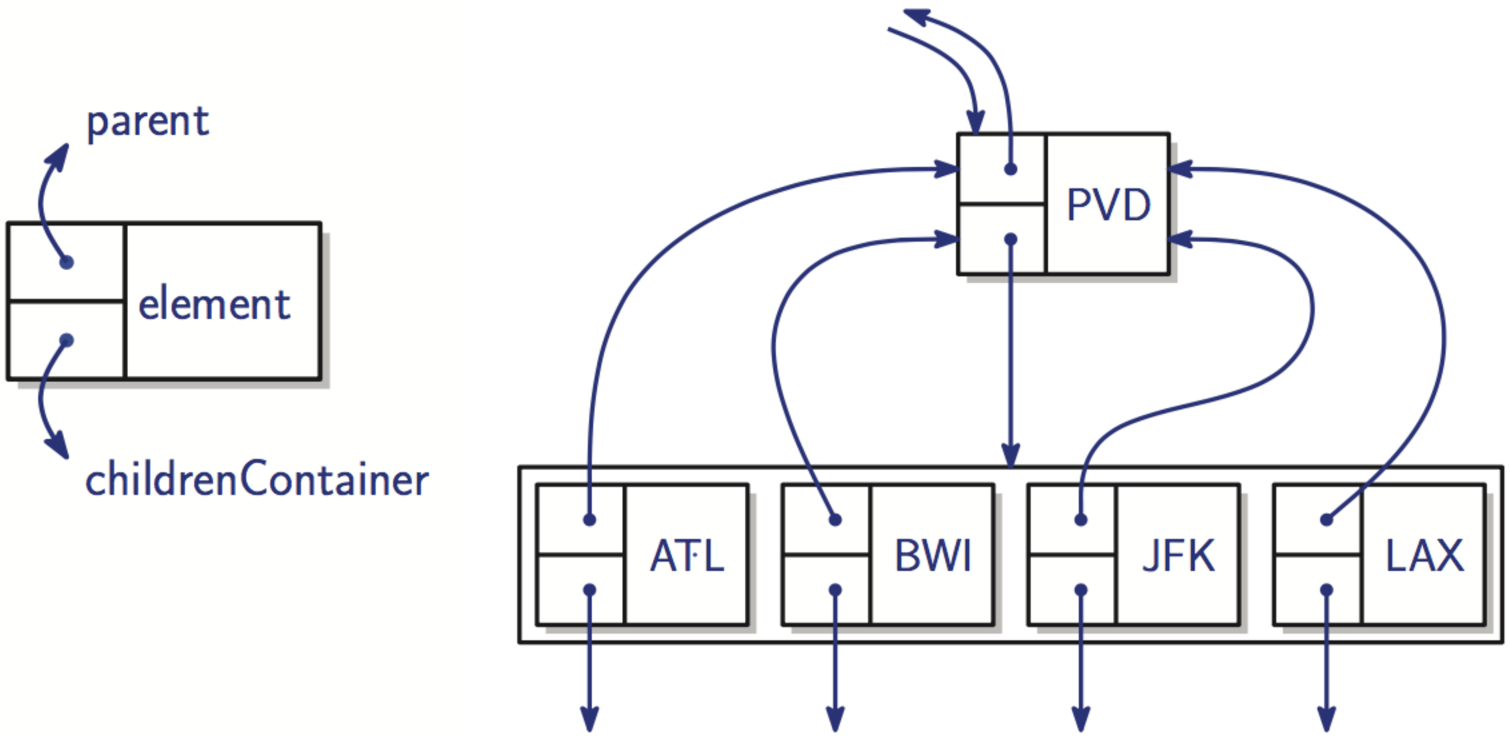
- Position-based methods:
  - position p.parent()
  - list<position> p.children()

- Query methods:
  - boolean p.isRoot()
  - boolean p.isExternal()

- Additional "update" methods may be defined by data structures implementing the Tree ADT
  - Remove the node at some position
  - Swap a parent and its specific child
  - Etc ...

A linked structure for General Trees

- One way of implementing a general tree



(a) (b)

Tree Traversal Algorithms

Traversal Computations

1. Depth?

2. Height?

3. Visit every nodes
   - Preorder
   - Postorder
   - Inorder

❖ These are the basic things to do for a given tree

1. Depth of a node

The depth of p's node can also be recursively defined as follows:

- If p is the root, then the depth of p is 0
- Otherwise, the depth of p is one plus the depth of the parent of p

1. Depth of a node

Algorithm depth(T, p):
    if p.isRoot() **then**
        return 0
    else
        return $1 +$ depth$(T, p$.parent$())$

Complexity? $O(d_p)$, worst-case $O(n)$

2. Height of a tree T: height1

Proposition 7.4: *The height of a tree is equal to the maximum depth of its external nodes.*

2. Height of a tree T: height1

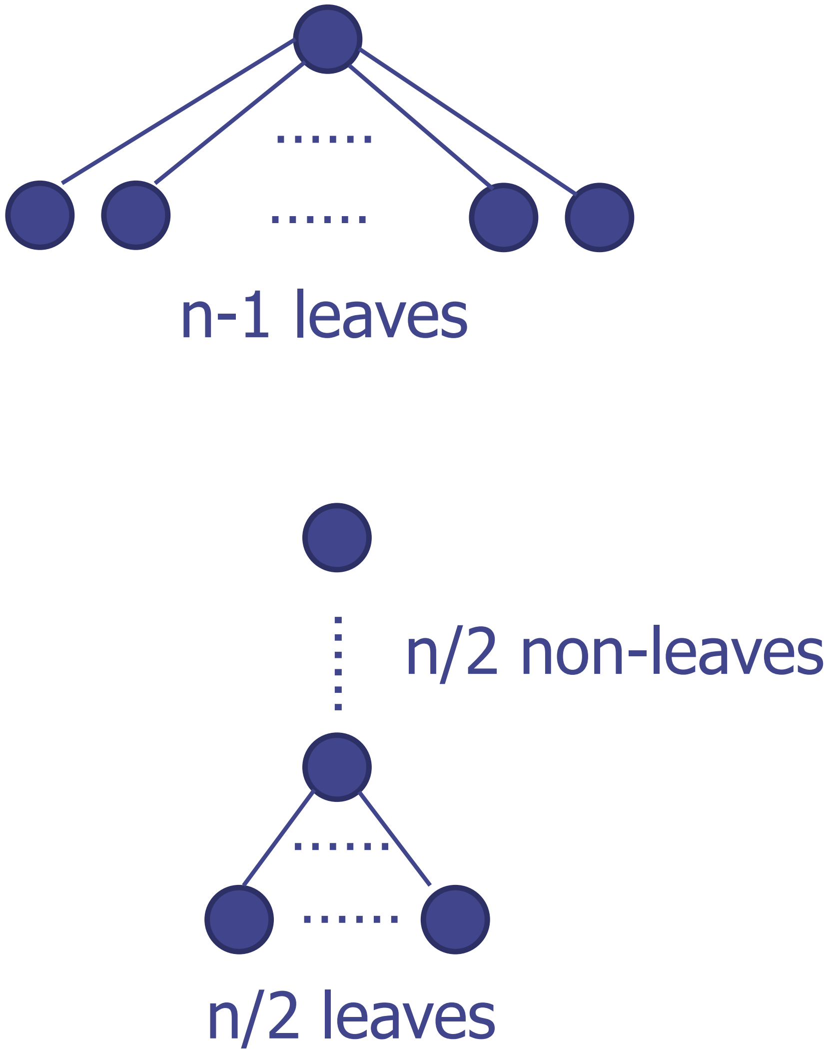
- Equal to the maximum depth of its leaves
- OK. Then, what about this algorithm?

Algorithm height1(T):
  $h = 0$
  for each $p \in T$.positions() **do**
    if p.isExternal() **then**
      $h = \max(h, \text{depth}(T, p))$
  return h

- Complexity?

Worst-case: $O(n^2)$

Two Trees



2. Height of a tree T: height2

- Why is height1 inefficient?

The ***height*** of a node p in a tree T is also defined recursively.

- If p is external, then the height of p is 0
- Otherwise, the height of p is one plus the maximum height of a child of p

2. Height of a tree T: height2

- Why is height1 inefficient?

Algorithm height2(T, p):
    if p.isExternal() **then**
        return 0
    else
        $h = 0$
        for each $q \in p$.children() **do**
            $h = \max(h, \text{height2}(T, q))$
        return $1 + h$

Proposition 7.5: *Let T be a tree with n nodes, and let c_p denote the number of children of a node p of T. Then $\sum_p c_p = n - 1$.*

2. Height of a tree T: height2

IUT-ECE

- Why is height1 inefficient?

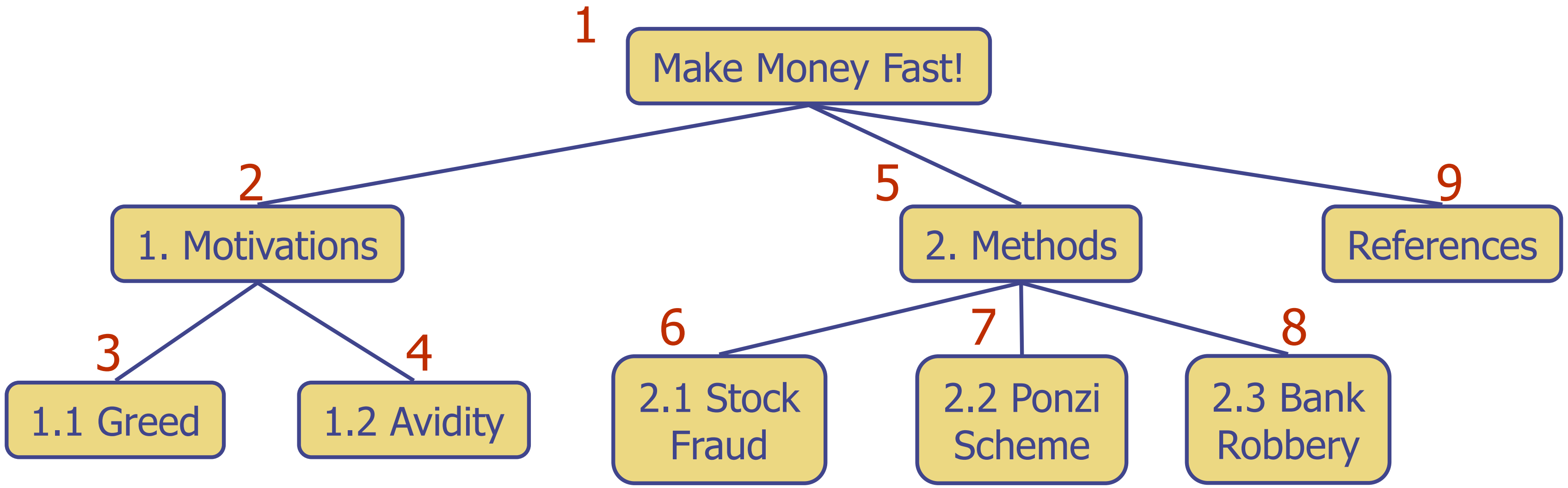
Algorithm height2(T, p):
  if p.isExternal() **then**
    return 0
  else
    $h = 0$
    for each $q \in p$.children() **do**
      $h = \max(h, \text{height2}(T, q))$
    return $1 + h$

$O(\sum_p (1 + c_p))$ Worst-case: *O(n)*

3. Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm ***preOrder***(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)

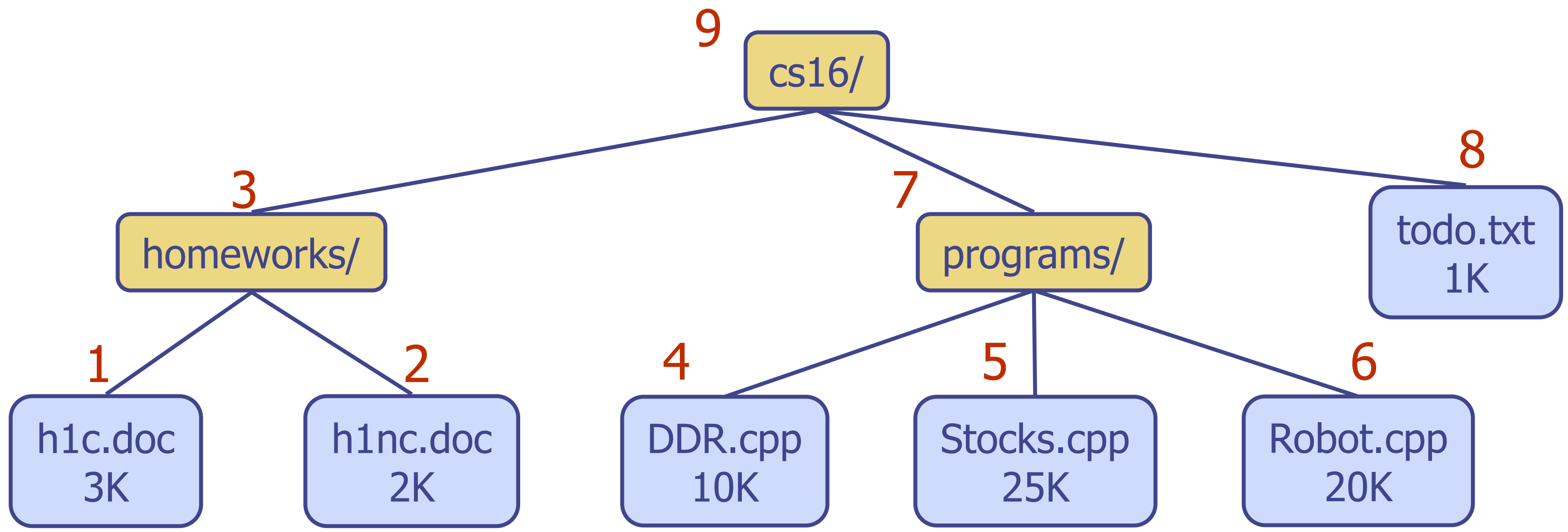


3. Postorder Traversal

IUT-ECE

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm *postOrder*(*v*)
    for each child *w* of *v*
        postOrder (*w*)
    visit(*v*)



3. Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree

Algorithm *inOrder*(*v*)
    if $\neg$ *v.isExternal*()
        inOrder(*v.left*())
    visit(*v*)
    if $\neg$ *v.isExternal*()
        inOrder(*v.right*())