Fall 2018

Data Structures Lecture 6

Fang Yu

Software Security Lab.
Dept. Management Information Systems,
National Chengchi University

Announcement

- **Project proposal** (due extended to Nov. 15) should include the following sections:
- 1. Introduction /Your topic and motivation
- 2. Search tricks /Your score formulation
- 3. System design /Class diagrams [proposal sample]
- 4. Schedule /How and when to accomplish stages
- 5. Challenges /Techniques that you need but may have a hard time to learn on your own

Announcement

- We will have programming lectures and tests on Nov. 8 in 逸仙樓 5F.
- HWs Review
  - BMI
  - Generic Progression
  - Keyword Counting
  - The Ordered List
  - HTML Tag Matching

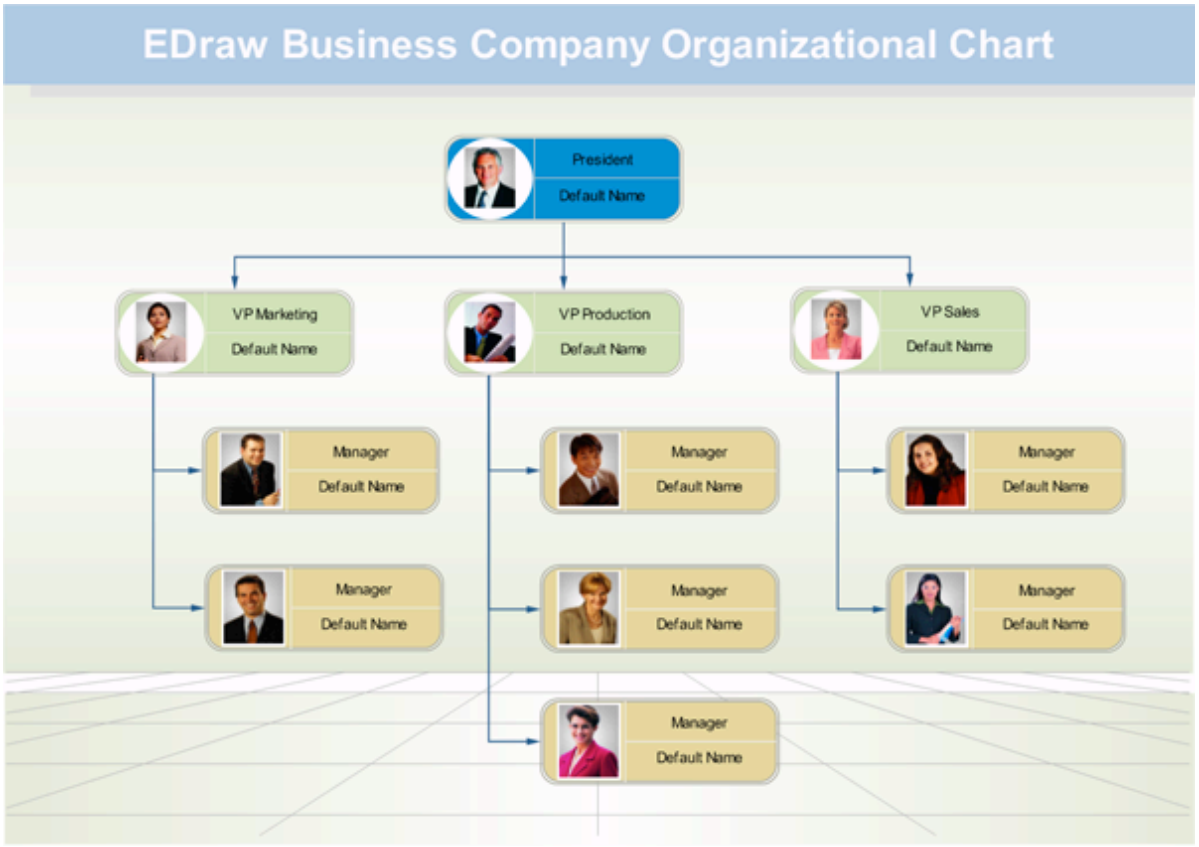
Abstract Non-linear Data Structures

Trees and their variations

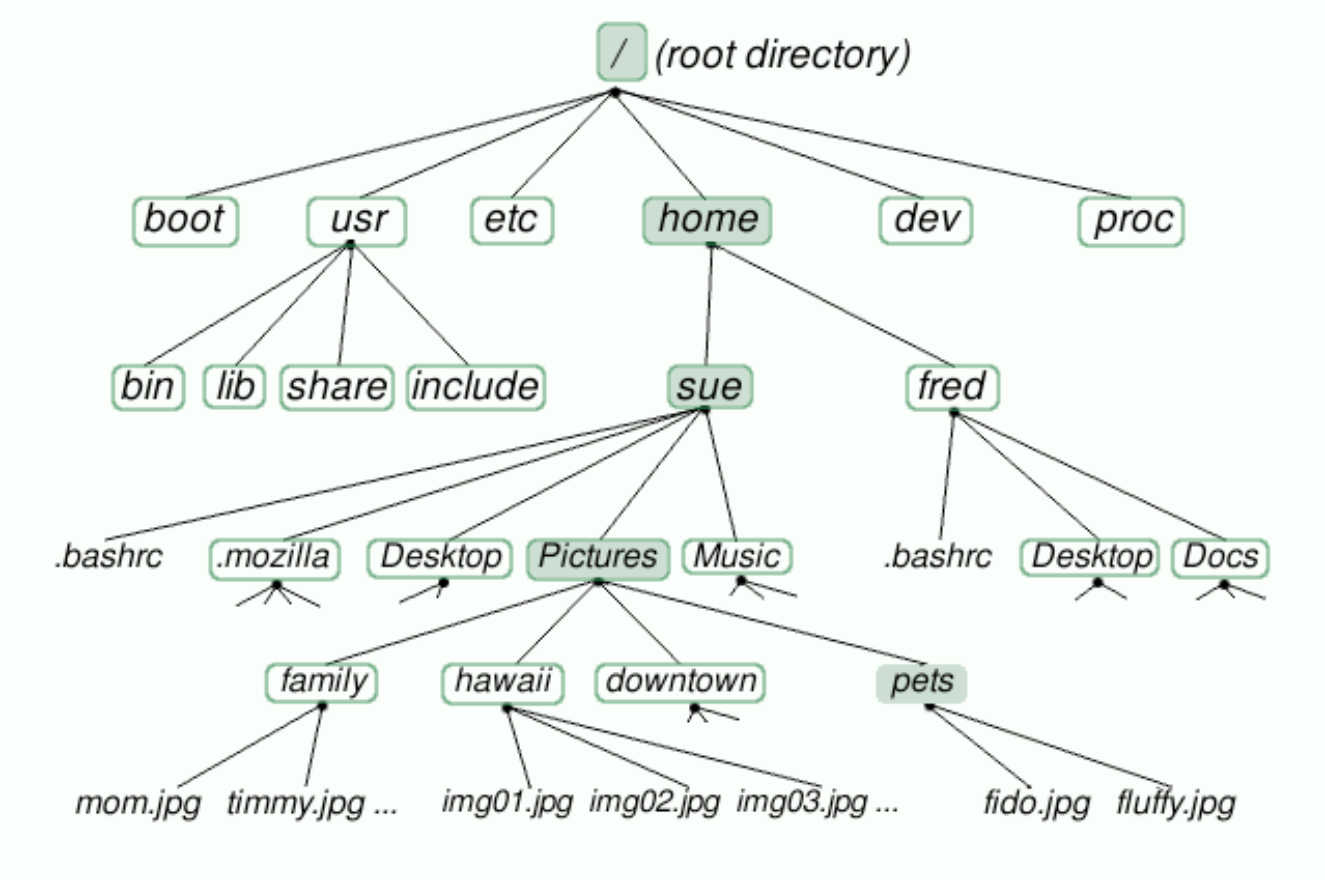
Abstract Data Type (ADT)

- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- We have discussed Array ADT, List ADT, Stack ADT, and Queue ADT
- All of them are linear ADT

A Hierarchical Structure

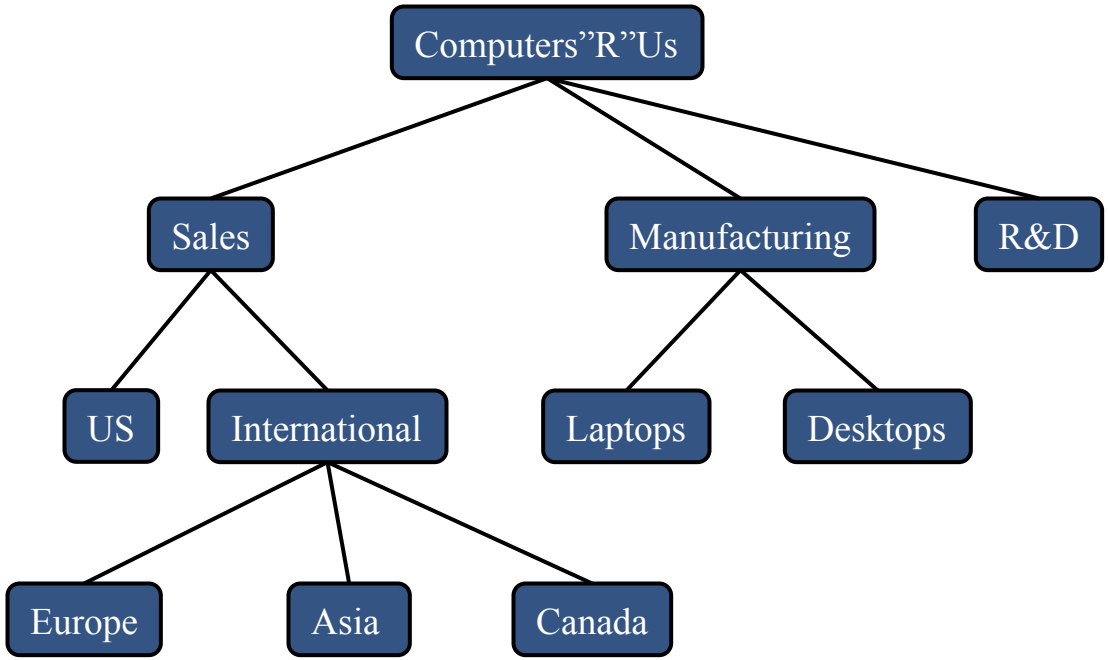


Linux/Unix file systems



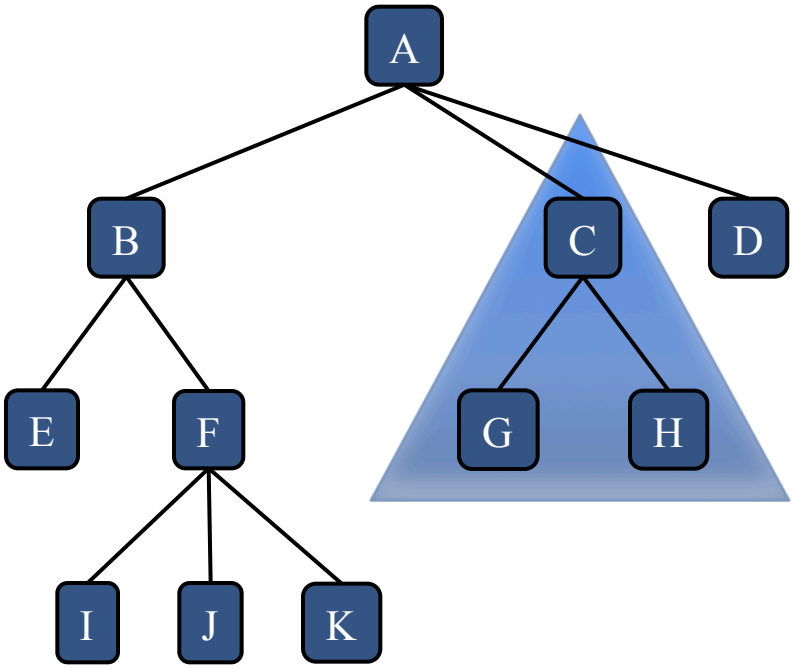
Tree: A Hierarchical ADT

- A tree (upside down) is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Each element (except the top element) has a parent and zero or more children elements



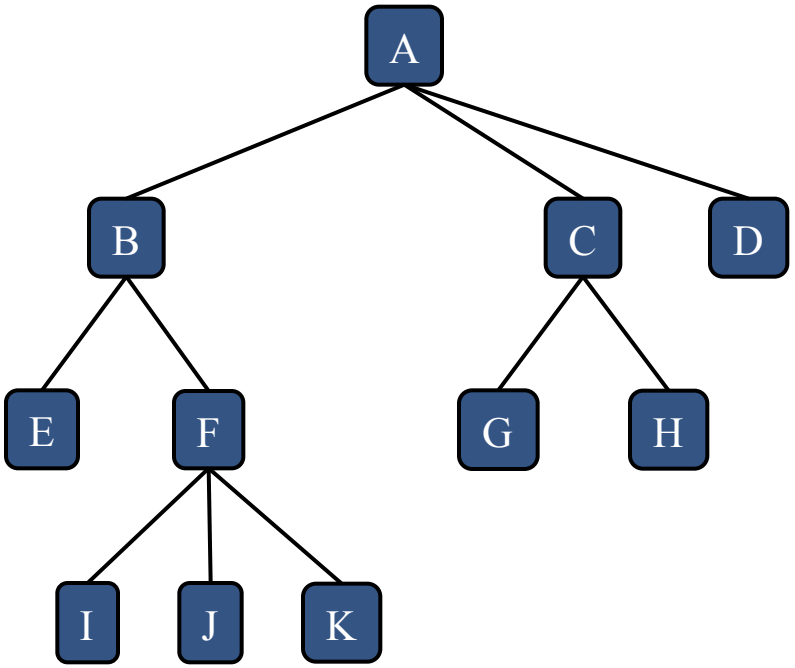
Tree Terminology

- **Root**: a node without any parent (A)
- **Internal node**: a node with at least one child (A, B, C, F)
- **External node** (a.k.a. leaf): a node without children (E, I, J, K, G, H, D)
- **Subtree**: tree consisting of a node and its descendants



Tree Terminology

- **Ancestor**s 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.



Tree ADT

- We use positions to define the tree ADT
- The positions in a tree are its nodes and neighboring positions satisfy the parent-child relationships

method	description
root()	Return the tree's root; error if tree is empty
parent(v)	Return v's parent; error if v is a root
children(v)	Return v's children (an iterable collection of nodes)
isRoot(v)	Test whether v is a root
isExternal(v)	Test whether v is an external node
isInternal(v)	Test whether v is an internal node

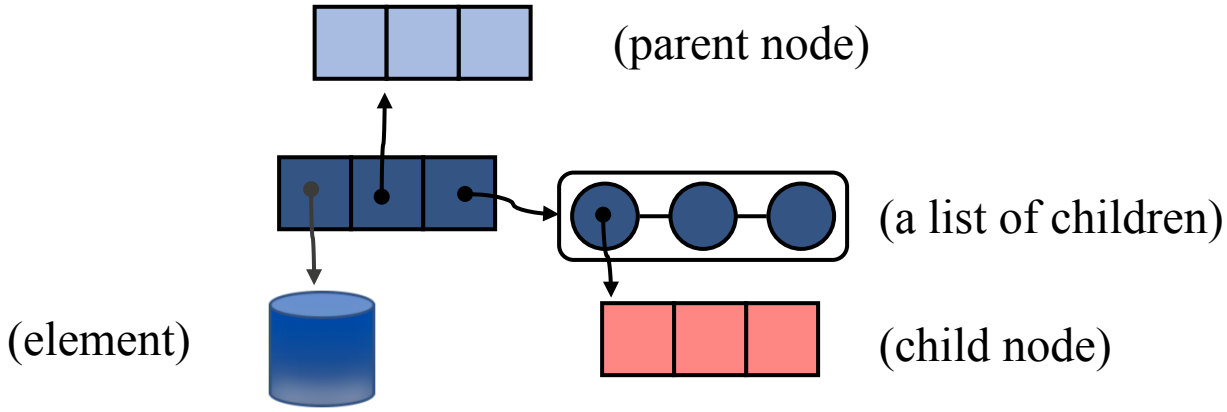
Tree ADT

- Generic methods (not necessarily related to a tree structure):

method	description
isEmpty()	Test whether the tree has any node or not
size()	Return the number of nodes in the tree
iterator()	Return an iterator of all the elements stored in the tree
positions()	Return an iterable collection of all the nodes of the tree
replace(v,e)	Replace with e and return the element stored at node v

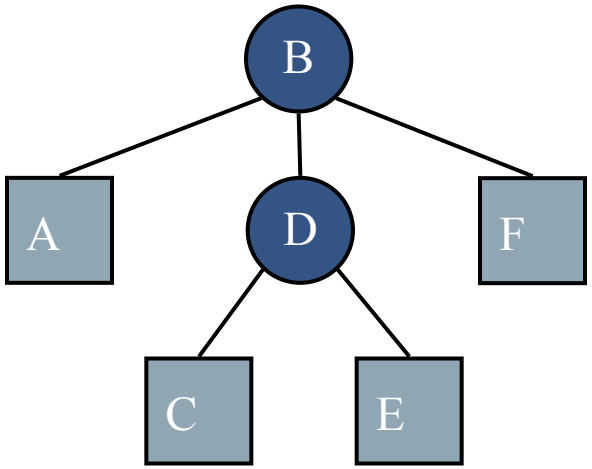
A Linked Structure for Tree

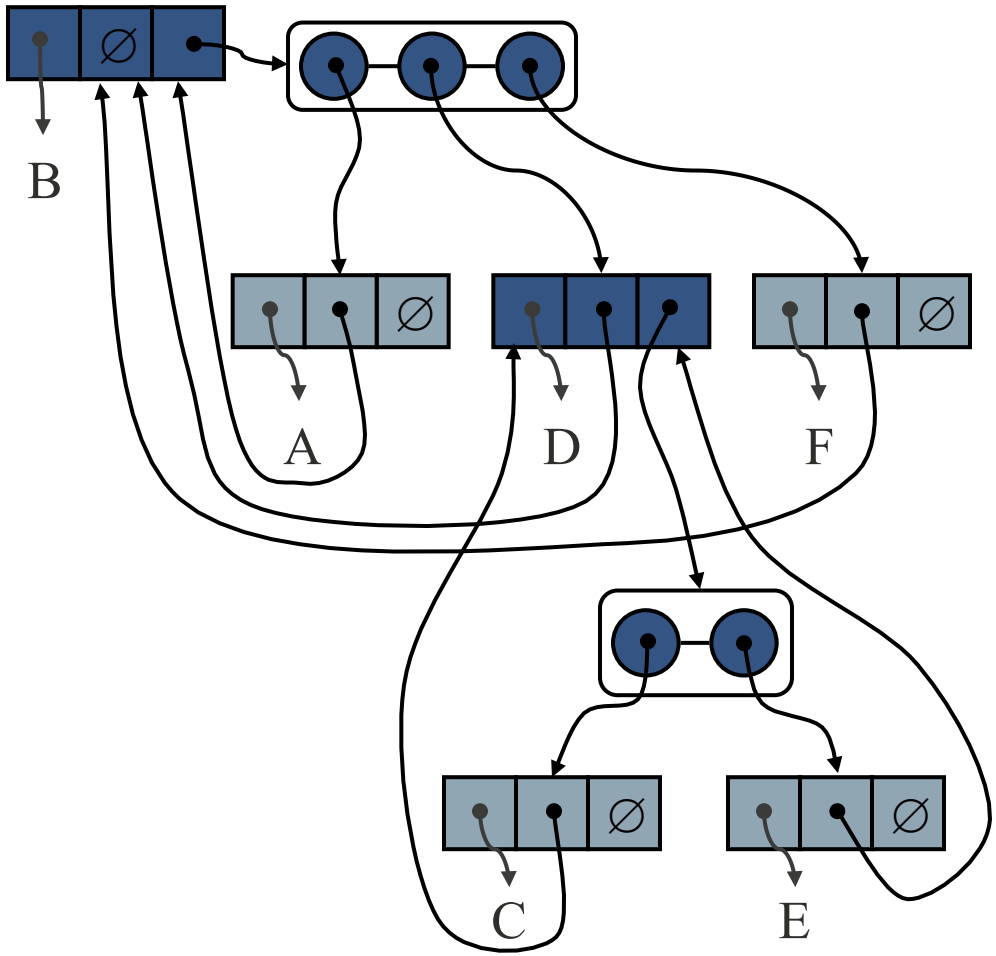
- A node is represented by an object storing
  - Element
  - A parent node
  - A sequence of children nodes



A Linked Structure for Tree

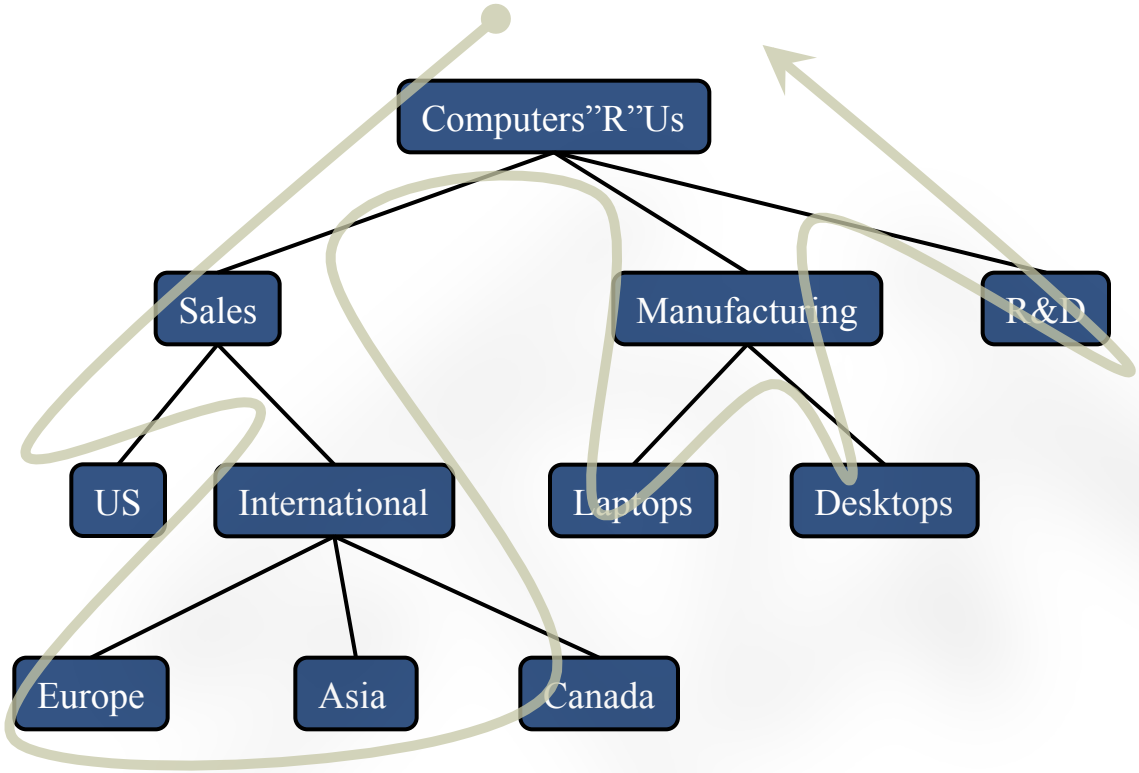
- Node objects implement the Position ADT





Tree Traversal

- Visit all nodes in a tree
- Do some operations during the visit

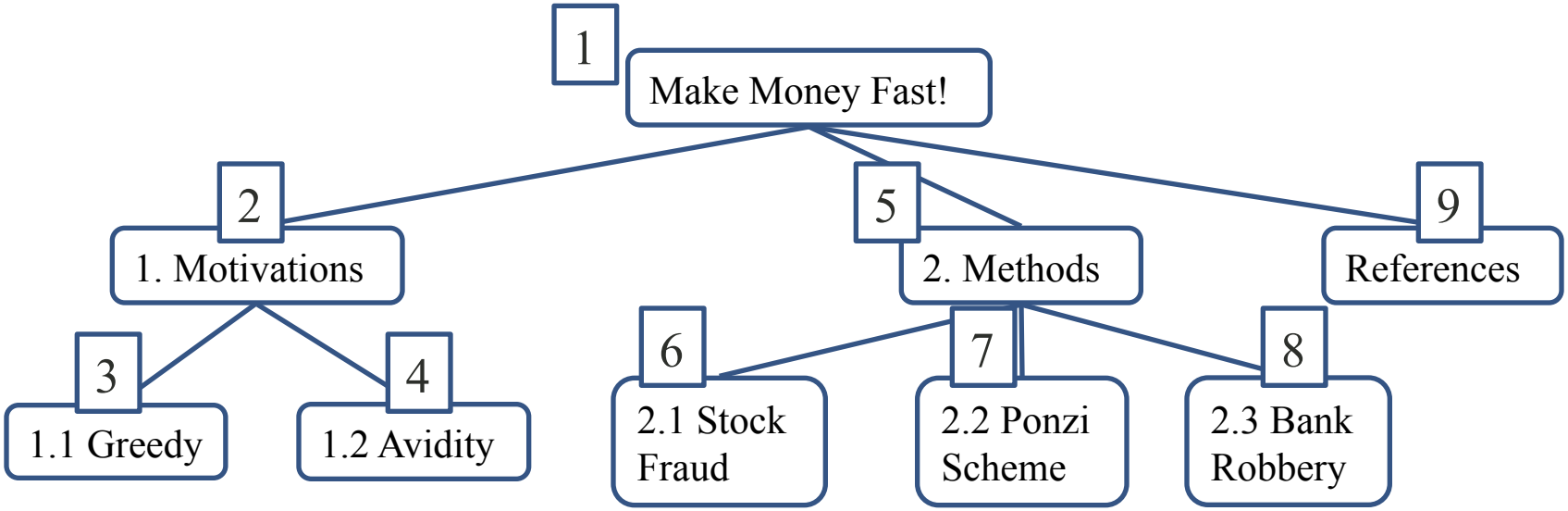


Preorder Traversal

- A node is visited (so is the operation) before its descendants
- Application:
  - Print a structured document

```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preOrder (w)
```

Preorder Traversal



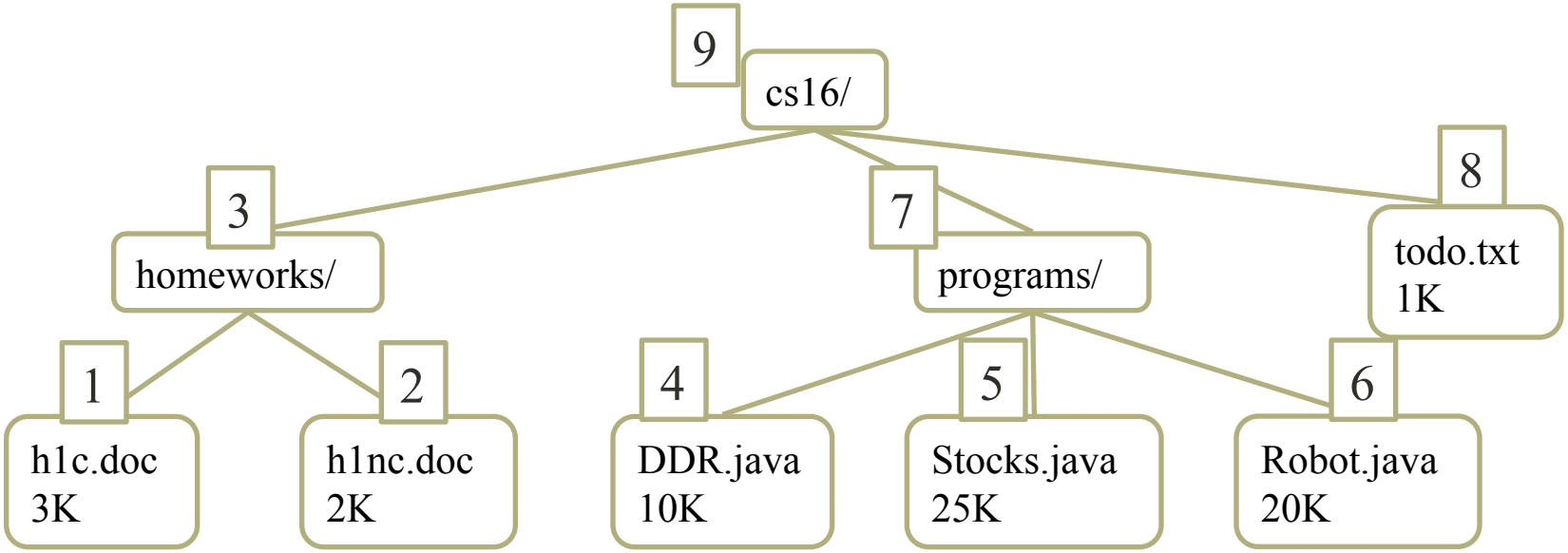
For your project, you can print a structured web site with its sub links using preorder traversal

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)
```

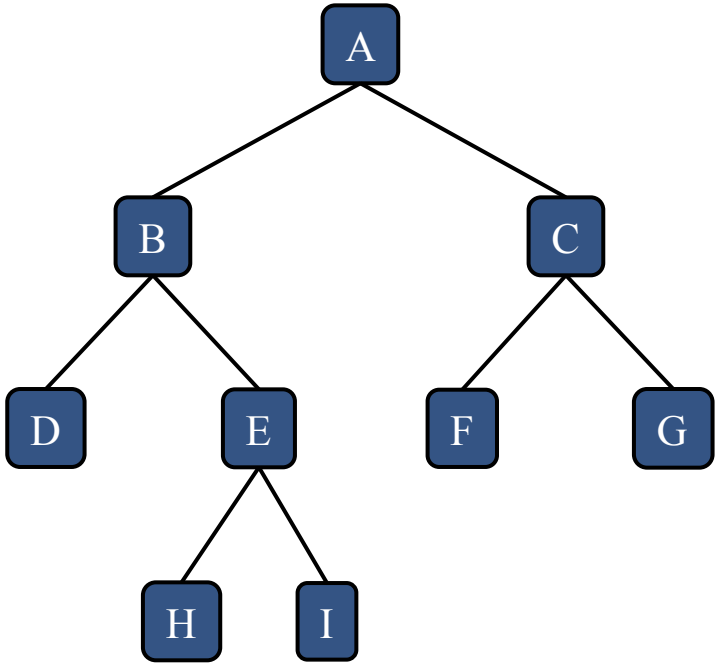
Postorder Traversal



For your project, you can compute the score of a web site and its sub links
Using postorder traversal

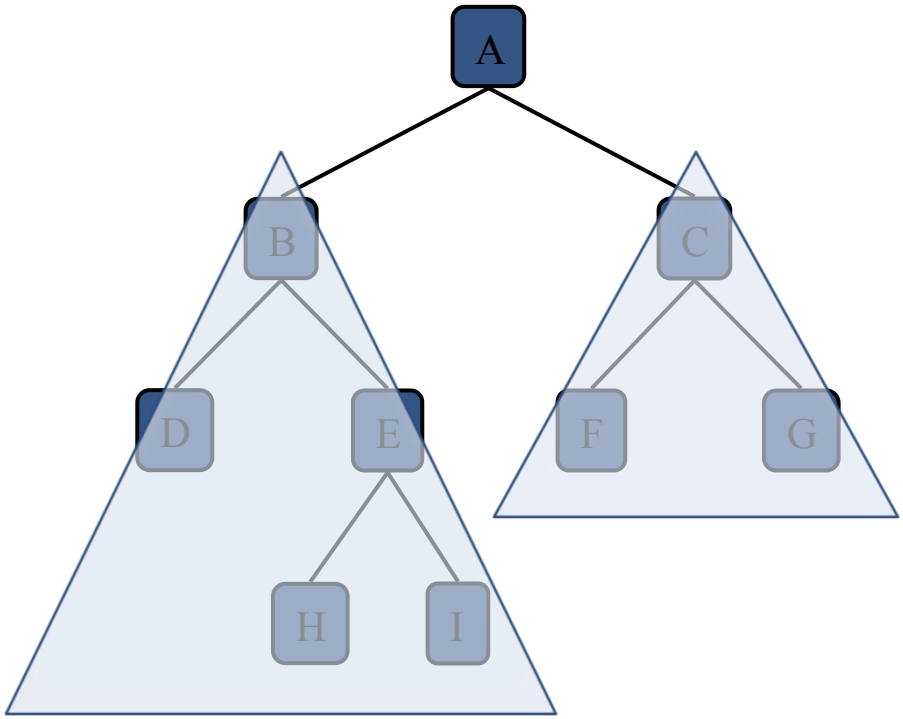
Binary Tree

- A binary tree is a tree with the following properties:
  - Each internal node has at most two children
  - The children of a node are an ordered pair (left and right)
- We call the children of an internal node left child and right child



Binary Tree

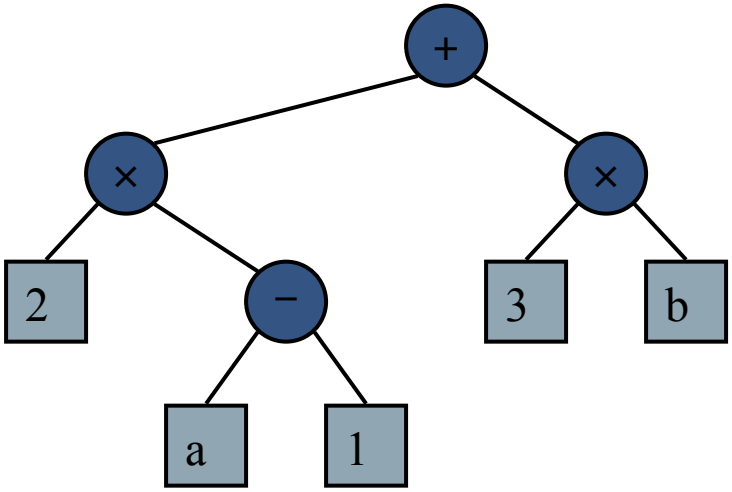
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree



Arithmetic Expression Tree

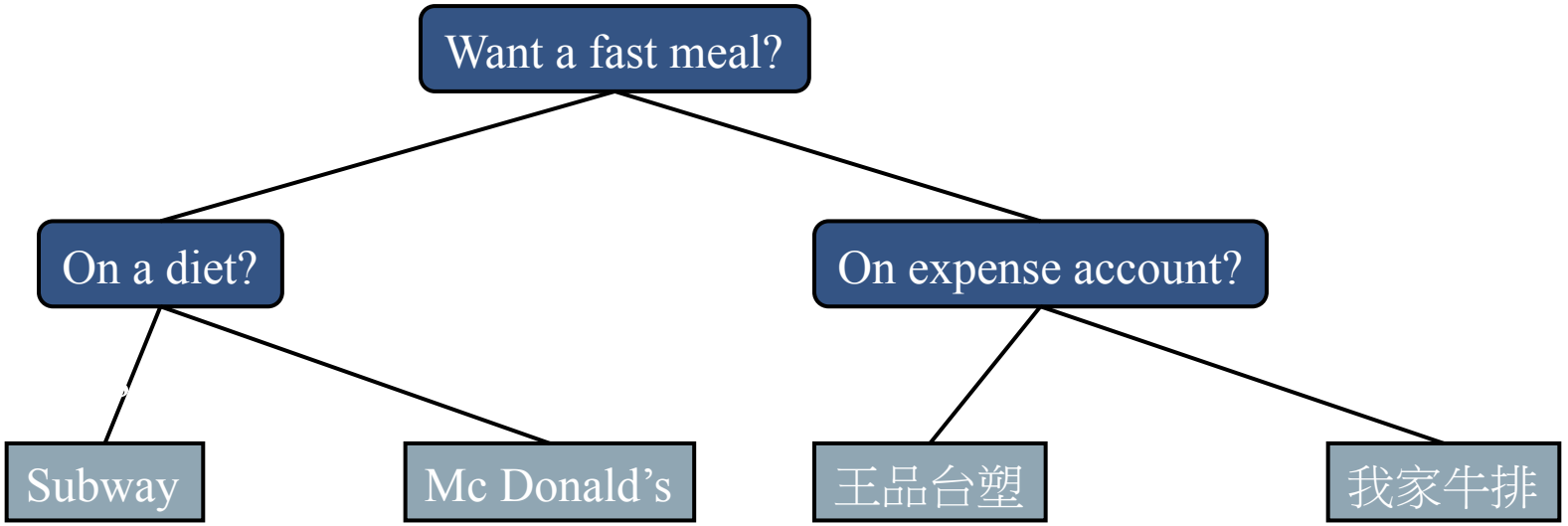
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression:

  $(2 \times (a - 1) + (3 \times b))$



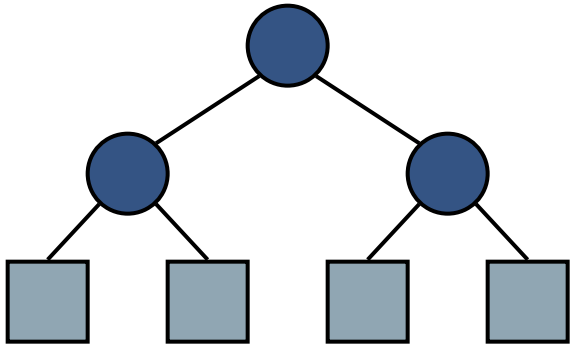
Decision Tree

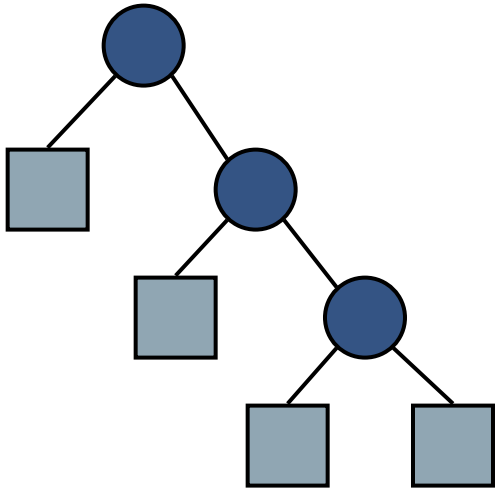
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Proper Binary Trees

- Each internal node has exactly 2 children





Proper Binary Trees

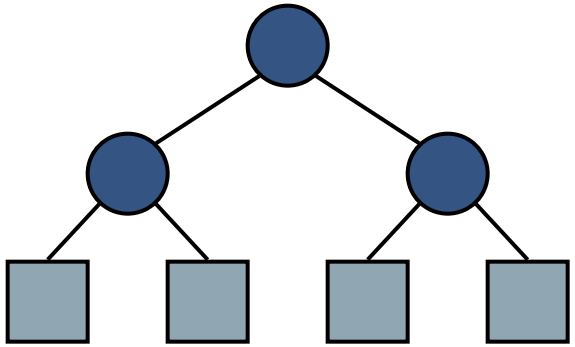
- n :number of total nodes
- e :number of external nodes
- i :number of internal nodes
- h :height (maximum depth of a node)

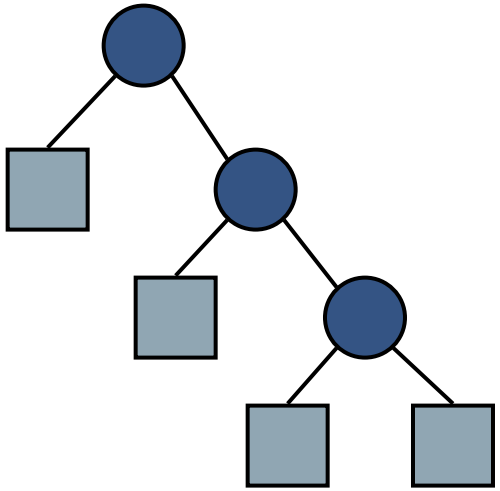
Properties:

1. $e = i + 1$
2. $n = 2e - 1$
3. $h \leq i$
4. $h \leq (n - 1)/2$
5. $e \leq 2^h$
6. $h \geq \log_2 e$
7. $h \geq \log_2 (n + 1) - 1$

Properties

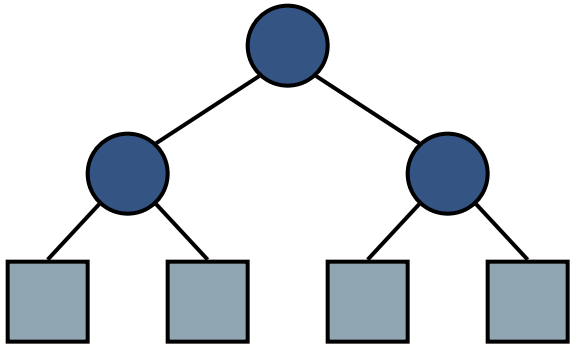
- 1. e = i+1
- 2. n = e+i = 2e-1 = 2i+1

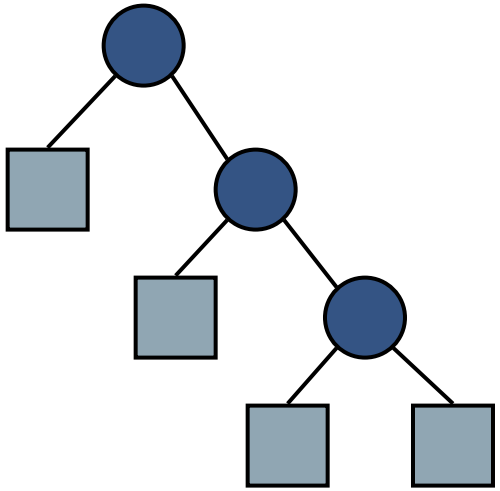




Properties

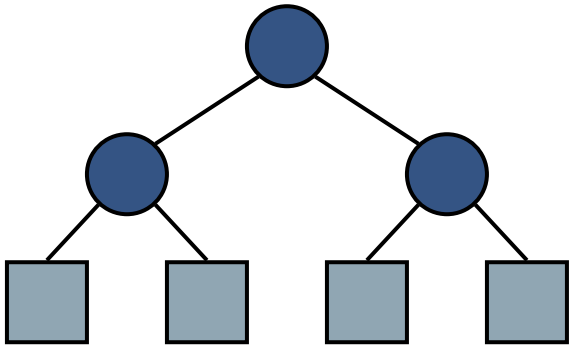
- 3. h <= i
- 4. h<= (n-1)/2

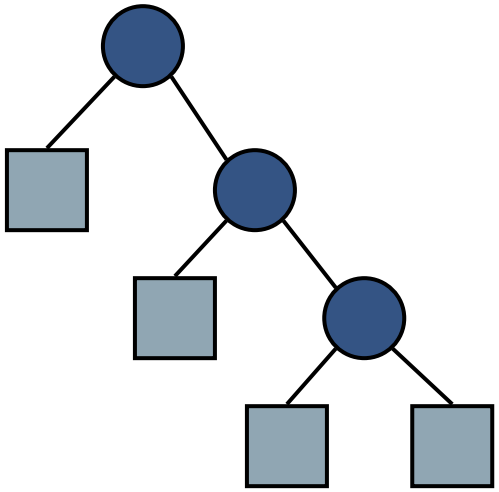




Properties

- 5. $e <= 2^h$
- 6. $h >= \log_2 e$
- 7. $h >= \log_2 ((n+1)/2) = \log_2(n+1) - 1$





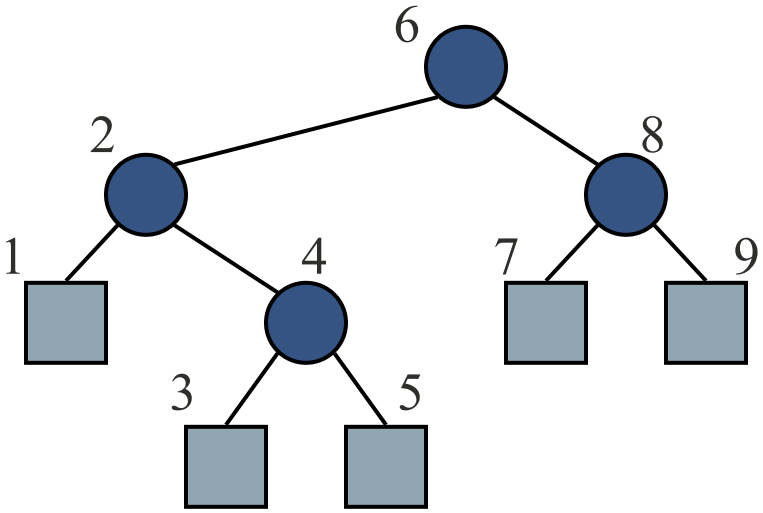
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Inorder Traversal

- A node is visited after its left subtree and before its right subtree

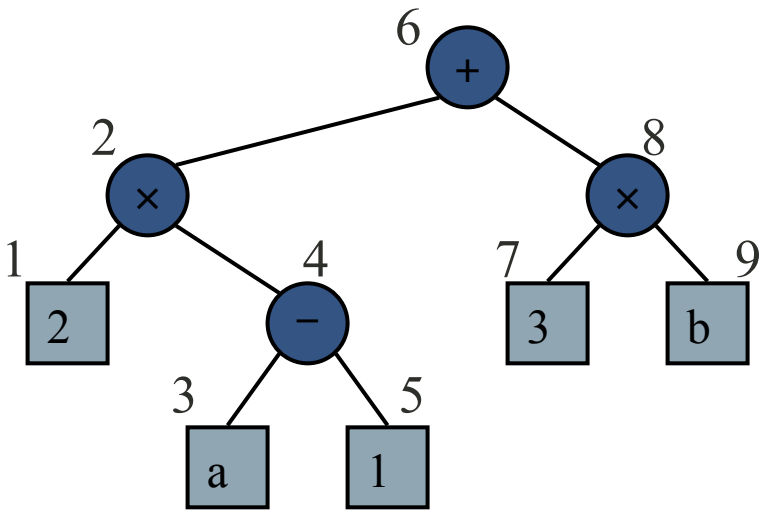
```
Algorithm inOrder(v)
    if hasLeft (v)
        inOrder (left (v))
    visit(v)
    if hasRight (v)
        inOrder (right (v))
```



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

```
Algorithm printExpression(v)
    if hasLeft (v)
        print("(")
        printExpression (left(v))
    print(v.element ())
    if hasRight (v)
        printExpression (right(v))
        print (")")
```

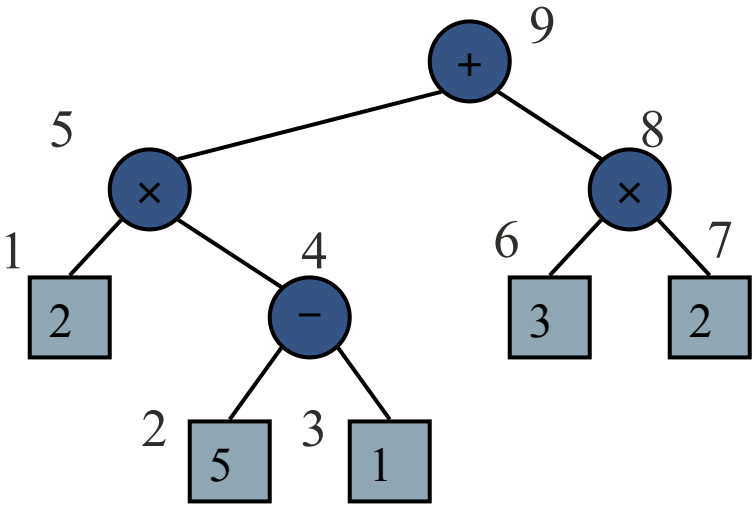


((2 × (a − 1)) + (3 × b))

Evaluate Arithmetic Expressions

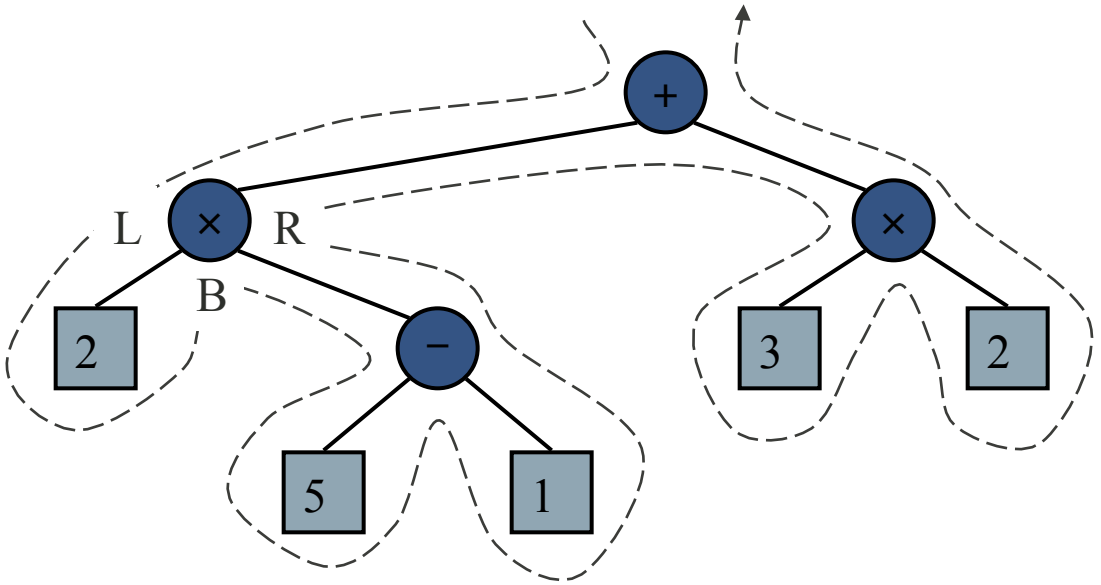
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

```
Algorithm evalExpr(v)
    if isExternal (v)
        return v.element ()
    else
        x ← evalExpr(leftChild (v))
        y ← evalExpr(rightChild (v))
        ◊ ← operator stored at v
        return x ◊ y
```



Euler Tour Traversal

- Generic traversal of a binary tree
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



A template method pattern

- A generic computation mechanism
- Specialized for an application by redefining the visit actions

Algorithm eularTour(T,v)
Perform the action for visiting node v on the left
If v has a left child u in T **then**
    eularTour(T, u)
Perform the action for visiting node v from below
If v has a right child w in T **then**
    eularTour(T, w)
Perform the action for visiting node v on the right

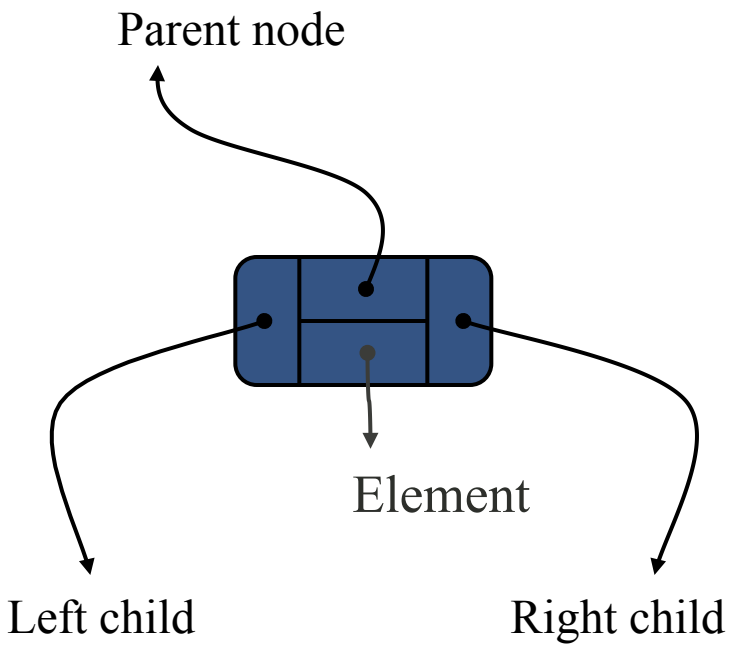
An Application of EularTour

- printExpression
  - On the left action: print (
  - From below action: print v
  - On the right action: print)

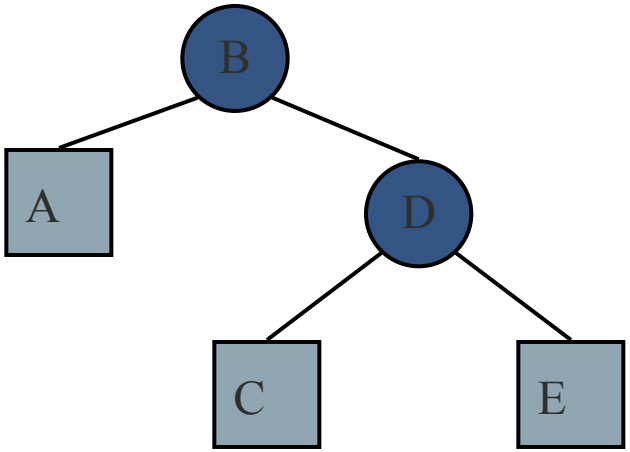
Algorithm printExpression(T,v)
if T.isInternal(v) then print "("
If v has a left child u in T **then**
    printExpression(T, u)
print(v)
If v has a right child w in T **then**
    printExpression(T, w)
if T.isInternal(v) then print ")"

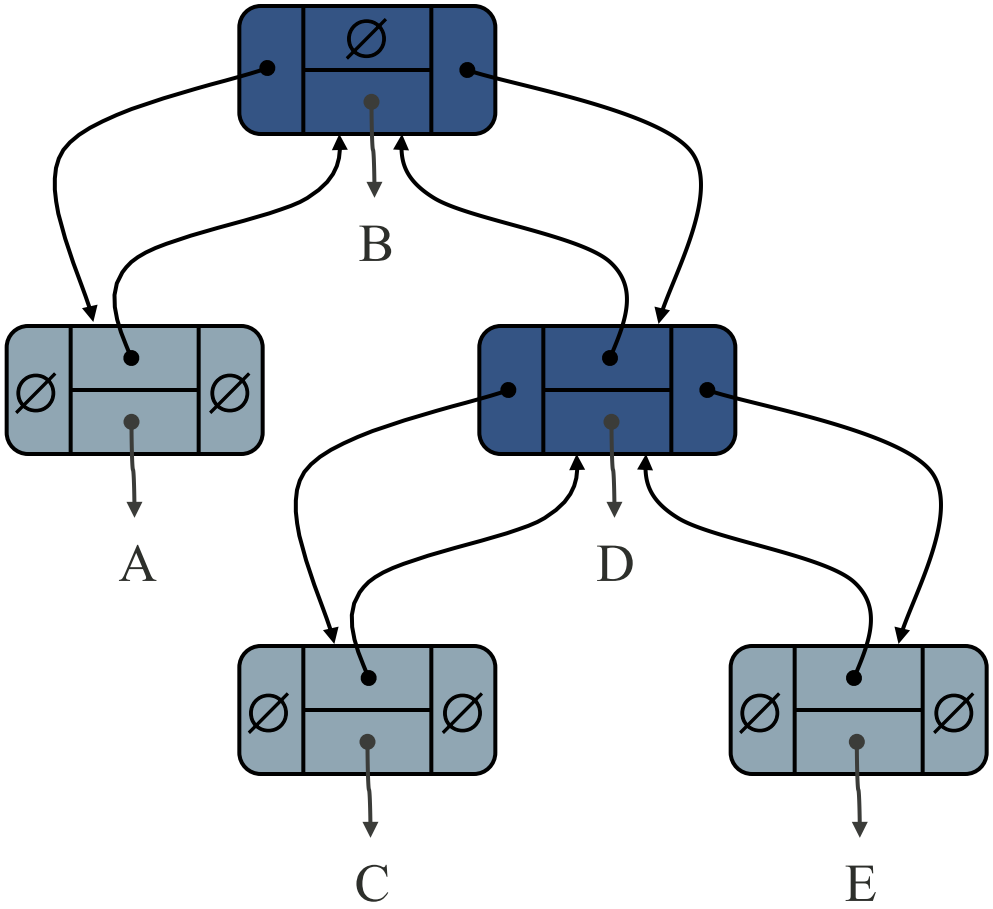
A Linked Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node



A Linked Structure for Binary Trees

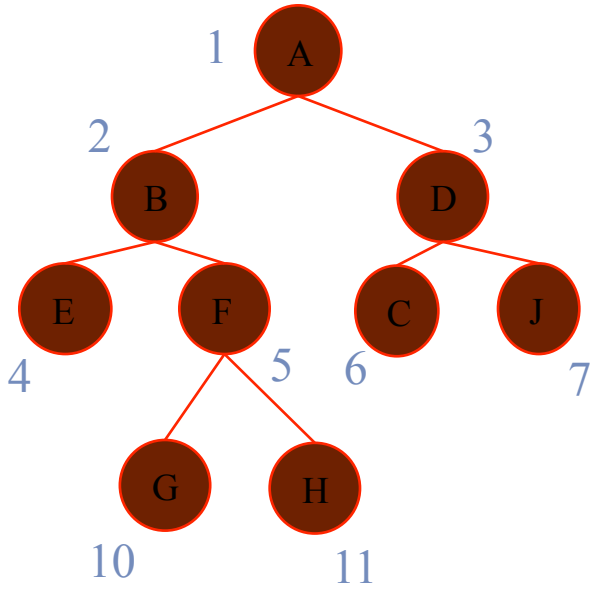


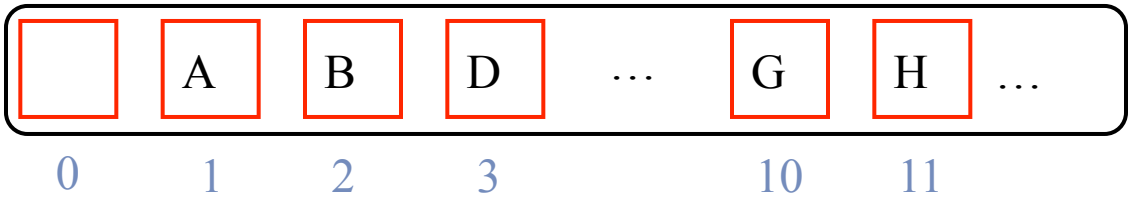


An Array-Based Representation

- Nodes are stored in an array A
- Node v is stored at A[rank(v)]
  - rank(root) = 1
  - Left in even: if node is the left child of parent(node), rank(node) = 2 · rank(parent(node))
  - Right in odd: if node is the right child of parent(node), rank(node) = 2 · rank(parent(node)) + 1
- A[0] is always empty
- A[i] is empty if there is no node in the ith position
- The array size N is $2^{(h+1)}$

An Array-Based Representation





HW 6 (Due on 11/1)

Compute the score of a ***website***!

- Construct a tree and its nodes according to a given website
  - An element (referred by a node) represents one web page and has three fields: (name, url, score)
- Given a keyword and its weight, compute the score of each node
  - Score = number of appearance * weight
  - The score of a node = the score of the content of its url + the scores of its children
  - This can be done by a postorder traversal of a tree
- Output the hierarchy of the website (with names and scores) using parentheses
  - This can be done by an eular tour

An example input

You will be given a website like:

- Soslab, http://soslab.nccu.edu.tw/Welcome.html
  - Publication, http://soslab.nccu.edu.tw/Publications.html
  - Projects, http://soslab.nccu.edu.tw/Projects.html
    - AppBeach, http://soslab.xyz:7777
    - Stranger, https://vlab.cs.ucsb.edu/stranger/
  - Member, http://soslab.nccu.edu.tw/Members.html
  - Course, http://soslab.nccu.edu.tw/Courses.html

An example output

Given a set of keywords, (Yu,1.2), (Fang, 1.8) you shall output something like

```
( Soslab, 56.6
    (Publication, 18)
    (Projects, 15.6
        (AppBeach, 2.6)
        (Stranger, 8.8)
    )
    (Member, 9.2)
    (Course, 4.8)
)
```

Fang Yu, 56.6 indicates that the sum of the score in the content of the given url (http://soslab.nccu.edu.tw) and its sub links

Coming Up…

- We will have a programming test in the lab on Nov. 8
- The project proposal is due on Nov. 15
- We will talk about heap (some kind of a tree) on Nov. 1.
  - Read Chapter 8