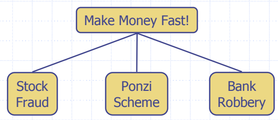
Trees

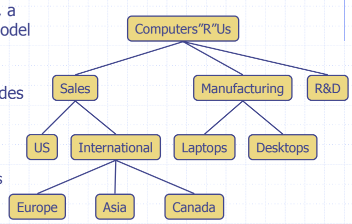


Outline and Reading

- Tree ADT (§6.1)
- Preorder and postorder traversals (§6.2.3)
- BinaryTree ADT (§6.3.1)
- Inorder traversal (§6.3.4)
- Euler Tour traversal (§6.3.4)
- Template method pattern (§6.3.5)
- Data structures for trees (§6.4)
- C++ implementation (§6.4.2)

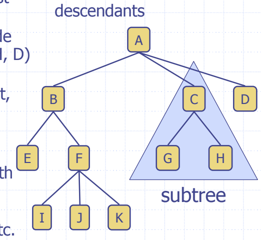
What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

- 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



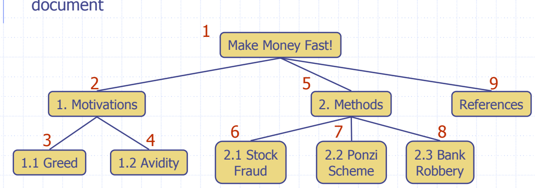
Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT

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

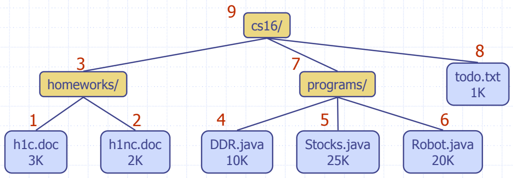
Algorithm *preOrder*(*v*)
  visit(*v*)
  for each child *w* of *v*
    preorder (*w*)



Postorder Traversal

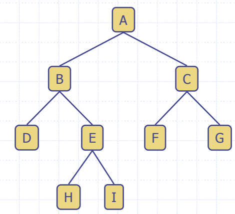
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

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



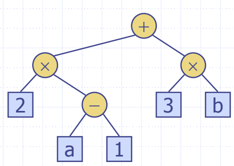
Binary Tree

- A binary tree is a tree with the following properties:
  - Each internal node has two children
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- 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
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



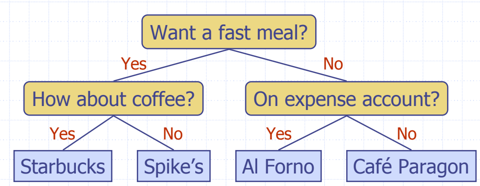
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



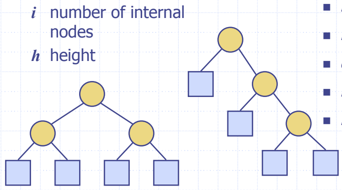
Decision Tree

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



Properties of Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height
- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$



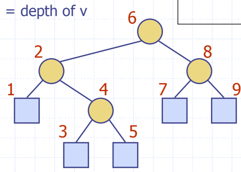
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

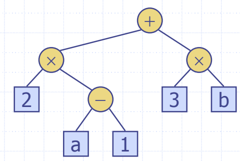
Algorithm *inOrder*(*v*)
  if *isInternal* (*v*)
    inOrder (*leftChild* (*v*))
  visit(*v*)
  if *isInternal* (*v*)
    inOrder (*rightChild* (*v*))



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 *isInternal* (*v*)
    print("(")
    inOrder (*leftChild* (*v*))
  print(*v.element* ())
  if *isInternal* (*v*)
    inOrder (*rightChild* (*v*))
    print (")")

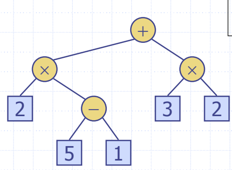


$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

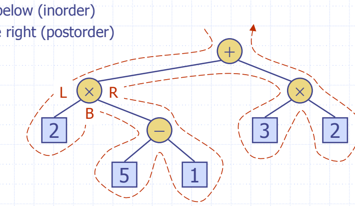
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm *evalExpr*(*v*)
  if *isExternal* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*leftChild* (*v*))
    $y \leftarrow$ *evalExpr*(*rightChild* (*v*))
    $\Diamond \leftarrow$ operator stored at *v*
    return $x \Diamond y$



Euler Tour Traversal

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



Template Method Pattern

- Generic algorithm that can be specialized by redefining certain steps
- Implemented by means of an abstract C++ class
- Visit methods that can be redefined by subclasses
- Template method eulerTour
  - Recursively called on the left and right children
  - A Result object with fields leftResult, rightResult and finalResult keeps track of the output of the recursive calls to eulerTour

```
class EulerTour {
protected:
   BinaryTree* tree;
   virtual void visitExternal(Position p, Result r) { }
   virtual void visitLeft(Position p, Result r) { }
   virtual void visitBelow(Position p, Result r) { }
   virtual void visitRight(Position p, Result r) { }
   int eulerTour(Position p) {
      Result r = initResult();
      if (tree–>isExternal(p)) { visitExternal(p, r); }
      else {
            visitLeft(p, r);
            r.leftResult = eulerTour(tree–>leftChild(p));
            visitBelow(p, r);
            r.rightResult = eulerTour(tree–>rightChild(p));
            visitRight(p, r);
            return r.finalResult;
      } // … other details omitted
```

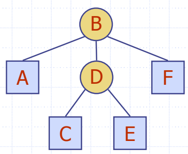

Specializations of EulerTour

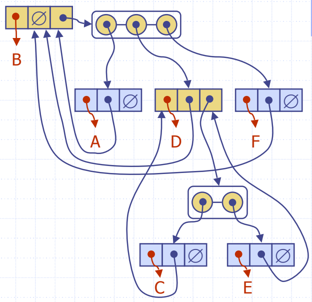
- We show how to specialize class EulerTour to evaluate an arithmetic expression
- Assumptions
  - External nodes support a function value(), which returns the value of this node.
  - Internal nodes provide a function operation(int, int), which returns the result of some binary operator on integers.

```
class EvaluateExpression
                  : public EulerTour {
protected:
   void visitExternal(Position p, Result r) {
      r.finalResult = p.element().value();
   }

   void visitRight(Position p, Result r) {
      Operator op = p.element().operator();
      r.finalResult = p.element().operation(
                        r.leftResult, r.rightResult);
   }

   // … other details omitted
};
```

Data Structure for Trees

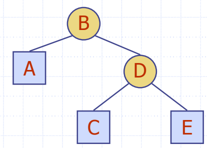
- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT

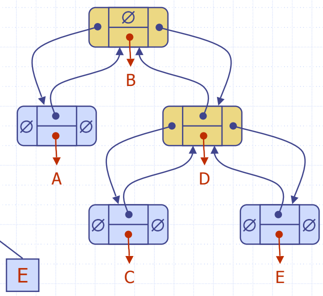




Data Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





C++ Implementation

- Tree interface
- BinaryTree interface extending Tree
- Classes implementing Tree and BinaryTree and providing
  - Constructors
  - Update methods
  - Print methods
- Examples of updates for binary trees
  - expandExternal(v)
  - removeAboveExternal(w)