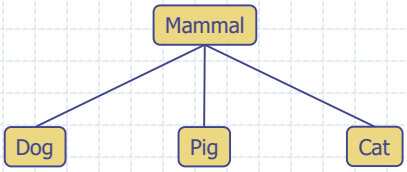
Trees



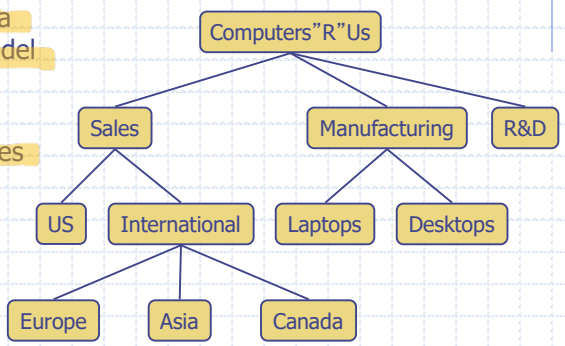
1

1

Ogni nodo albero tranne il nodo radice ha padre

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
  - Genealogical tree



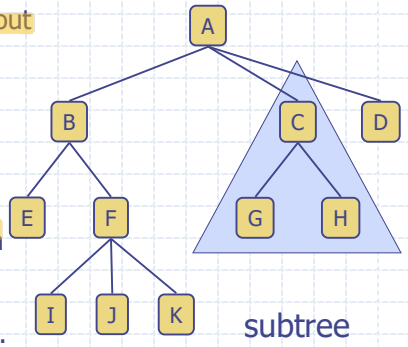
2

2

Se nodo non ha figlio: Nodo esterno o foglia

Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (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



3

Tree ADT

Scelta interfaccia position implementata con getElement

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)

→ Oggetto iterabile

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

Ovviamente anche metodi che aggiungono e rimuovono.

↓ addChild

4

Il nodo dell'albero deve essere di tipo position, ma come lo costruisco? Deve avere padre e figli.

Java Interface

Non ci sono tutti i metodi

Methods for a Tree interface:

```
1  /** An interface for a tree where nodes can have an arbitrary number of children. */
2  public interface Tree<E> extends Iterable<E> {
3    Position<E> root( );
4    Position<E> parent(Position<E> p) throws IllegalArgumentException;
5    Iterable<Position<E>> children(Position<E> p)
6                                    throws IllegalArgumentException;
7    int numChildren(Position<E> p) throws IllegalArgumentException;
8    boolean isInternal(Position<E> p) throws IllegalArgumentException;
9    boolean isExternal(Position<E> p) throws IllegalArgumentException;
10   boolean isRoot(Position<E> p) throws IllegalArgumentException;
11   int size( );
12   boolean isEmpty( );
13   Iterator<E> iterator( );
14   Iterable<Position<E>> positions( );
15 }
```

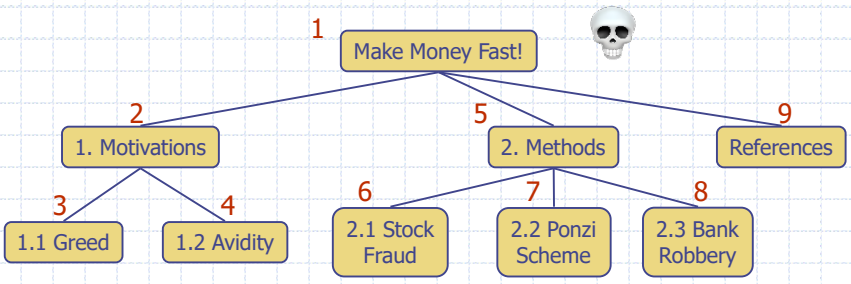
5

5

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*)



6

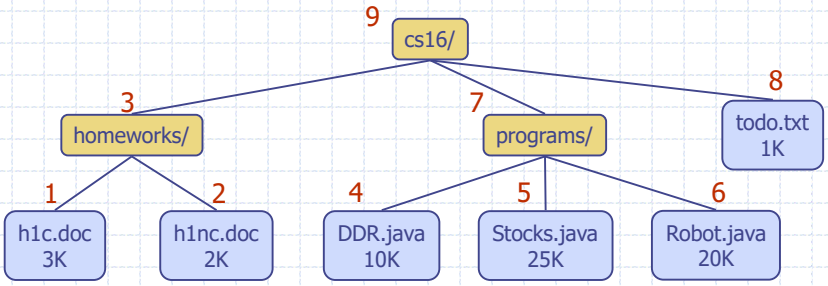
6

Regola di attraversamento: in preordine.
Si visita contenuto di un nodo prima di tutti i suoi figli.
Es: Testi scritti in capitoli e paragrafi. Stampo indice.

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)



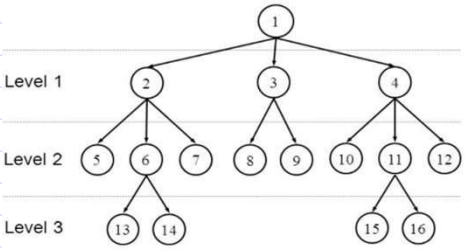
7

7

Breadth-first Traversal

- We want to visit the elements level-by-level (and left-to-right, as usual)
- This level-by-level traversal is called a *breadth-first traversal* because we explore the full width of the tree at a given level, before going *deeper*.

Breadth-first traversal:



Nodes are numbered in the order of a breadth-first traversal of the tree.

Visit the root first.

Visit all nodes at level 1, followed by all nodes at level 2, …

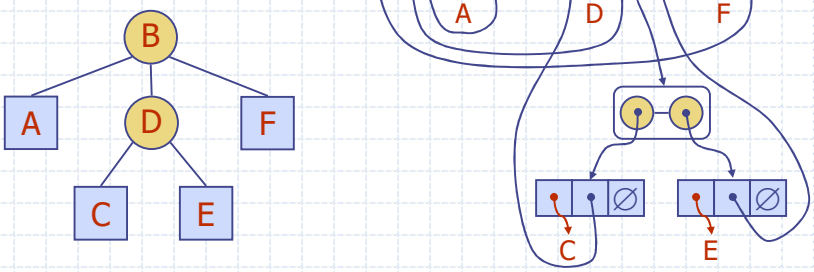
At each level visit nodes from left to right

8

8

Linked Structure for Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes



9

9

Java Implementation (Generic Tree)

```
public  class GenTree_mod<E>  implements Tree<E> {

  //---------------- nested Node class ----------------
  /** Nested static class for a  tree node. */
  protected static class Node<E> implements Position<E> {
    private E element;          // an element stored at this node
    private Node<E> parent;     // a reference to the parent node (if any)
    private ArrayList<Position<E>> childrens ; //vector of references to all child nodes

    /**
     * Constructs a node with the given element and neighbors.
     *
     * @param e  the element to be stored
     * @param above       reference to a parent node
     * @param childrens  references to all child nodes
     */
    public Node(E e, Node<E> above) {
      element = e;
      parent = above;
      childrens = new ArrayList<>();
    }

    // accessor methods
    public E getElement() { return element; }
    public Node<E> getParent() { return parent; }


    // update methods
    public void setElement(E e) { element = e; }
    public void setParent(Node<E> parentNode) { parent = parentNode; }

  } //----------- end of nested Node class -----------

  /** Factory function to create a new node storing element e. */
  protected Node<E> createNode(E e, Node<E> parent) {
    return new Node<E>(e, parent);
  }
```

Implementation of the single node of the **Tree** using a nested class **Node**

10

10

Java Implementation (Generic Tree)

```
public  class GenTree_mod<E>  implements Tree<E> {

// GenTree instance variables
  /** The root of the  tree */
  protected Node<E> root = null;

  /** The number of nodes in the
  private int size = 0;

  // constructor
  /** Construts an empty binary tree. */
  public GenTree_mod()  { }      // constructs an empty generic tree

 /** Implements methods of the interface Tree */
  public Position<E> root() {
    return root;
  }
  public Position<E> parent(Position<E> p) throws IllegalArgumentException {
    Node<E> node = validate(p);
    return node.getParent();
  }
  public int numChildren(Position<E> p) {

    Node<E> parent = validate(p);
    int count=parent.childrens.size();
    return count;
  }

   public boolean isInternal(Position<E> p) { return numChildren(p) > 0; }
   public boolean isExternal(Position<E> p) { return numChildren(p) == 0; }
   public boolean isRoot(Position<E> p) { return p == root(); }
   public int size() {return size;}
   public boolean isEmpty() { return size() == 0; }

 /** Methods to build the Tree */
 public Position<E> addRoot(E e) throws IllegalStateException {
    if (size()>0) throw new IllegalStateException("Tree is not empty");
    root = createNode(e, null);
    size = 1;
    return root;
  }
  public Position<E> addChild(Position<E> p, E e)
                      throws IllegalArgumentException {
    Node<E> parent = validate(p);
    Node<E> child = createNode(e, parent);
    int nchilds=numChildren(p);
    parent.childrens.add(nchilds,child);
    size++;
    return child;
  }

  protected Node<E> validate(Position<E> p) throws IllegalArgumentException {
    if (!(p instanceof Node))
      throw new IllegalArgumentException("Not valid position type");
    Node<E> node = (Node<E>) p;      // safe cast
    return node;
  }
```

11

Use of a Generic Tree (the node is a String)

```
public class TestGenTree_mod {


 public static void main(String argv[])

{

  GenTree_mod gtree= new GenTree_mod<String>();


System.out.println("Avvio riempimento albero");

gtree.addRoot("Elisabetta II");
Position root=gtree.root();
Position carlo=gtree.addChild(root,"Carlo");
Position william=gtree.addChild(carlo,"William");
gtree.addChild(william,"George");
gtree.addChild(william,"Charlotte");
gtree.addChild(carlo,"Harry");
Position anna=gtree.addChild(root,"Anna");
gtree.addChild(anna,"Peter");
gtree.addChild(anna,"Zara");
Position andrea=gtree.addChild(root,"Andrea");
gtree.addChild(andrea,"Beatrice");
gtree.addChild(andrea,"Eugenia");
Position edoardo=gtree.addChild(root,"Edoardo");
gtree.addChild(edoardo,"Lousie");
gtree.addChild(edoardo,"James");
```

Use of the **GenTree** class to create a **Genealogical tree**

12

12

Java Implementation (Generic Tree)

```
public Iterable<Position<E>> children(Position<E> p) throws IllegalArgumentException {
 Node<E> parent = validate(p);
 return  parent.childrens;
}


 private class ElementIterator implements Iterator<E> {
   Iterator<Position<E>> posIterator = positions().iterator();
   public boolean hasNext() { return posIterator.hasNext(); }
   public E next() { return posIterator.next().getElement(); } // return element!
   public void remove() { posIterator.remove(); }
 }

 /**
  * Returns an iterator of the elements stored in the tree.
  * @return iterator of the tree's elements
  */

 private void addChildrens(Position<E> p, List<Position<E>> snapshot) {
   snapshot.add(p);
   for (Position<E> c : children(p))
     addChildrens(c, snapshot);
   }

 public Iterator<E> iterator() { return new ElementIterator(); }

 public Iterable<Position<E>> positions() {
  ArrayList<Position<E>> snapshot = new ArrayList<>();
    if (!isEmpty())
     addChildrens(root(), snapshot);   // fill the snapshot recursively
  return snapshot;
    }
```

Implementation of a method **iterator**() of **Tree**

In preordine

13

13

Visit of a Generic Tree (the node is a String)

```
public class TestGenTree_mod {

 public static void main(String argv[])
{
 int i=0;

  GenTree_mod gtree= new GenTree_mod<String>
System.out.println("Avvio riempimento alber

gtree.addRoot("Elisabetta II");
Position root=gtree.root();
Position carlo=gtree.addChild(root,"Carlo")
Position william=gtree.addChild(carlo,"Will
gtree.addChild(william,"George");
gtree.addChild(william,"Charlotte");
gtree.addChild(carlo,"Harry");
Position anna=gtree.addChild(root,"Anna");
gtree.addChild(anna,"Peter");
gtree.addChild(anna,"Zara");
Position andrea=gtree.addChild(root,"Andrea
gtree.addChild(andrea,"Beatrice");
gtree.addChild(andrea,"Eugenia");
Position edoardo=gtree.addChild(root,"Edoar
gtree.addChild(edoardo,"Lousie");
gtree.addChild(edoardo,"James");

System.out.println("Dimensione btree= "+ gtree.size());

System.out.println("Stampa nodi albero tramite iteratore");
Iterator<String> iter=gtree.iterator();
while  (iter.hasNext()) {
    System.out.println("Valore nel nodo "+i+" = "+iter.next());
    i++;
 }
System.out.println("Stampa nodi albero");
Iterable<Position<String>> it1=gtree.positions();
for (Position p: it1)
 System.out.println(p.getElement());
}
}
```

```
algoritmi miei — -bash — 64×35
Avvio riempimento albero
Dimensione btree= 15
Stampa nodi albero tramite iteratore
Valore nel nodo 0 = Elisabetta II
Valore nel nodo 1 = Carlo
Valore nel nodo 2 = William
Valore nel nodo 3 = George
Valore nel nodo 4 = Charlotte
Valore nel nodo 5 = Harry
Valore nel nodo 6 = Anna
Valore nel nodo 7 = Peter
Valore nel nodo 8 = Zara
Valore nel nodo 9 = Andrea
Valore nel nodo 10 = Beatrice
Valore nel nodo 11 = Eugenia
Valore nel nodo 12 = Edoardo
Valore nel nodo 13 = Lousie
Valore nel nodo 14 = James
Stampa nodi albero
Elisabetta II
Carlo
William
George
Charlotte
Harry
Anna
Peter
Zara
Andrea
Beatrice
Eugenia
Edoardo
Lousie
James
iMac-di-Rocco:algoritmi miei roccoaversa$
```

14

14

Java Implementation (Visit Generic Tree)

```
public  class VisitGenTree<E>  extends GenTree_mod<E> {

  // constructor

  public VisitGenTree()  { super();}      // constructs an empty generic tree

public void  preorder(Position<E> p) {
    System.out.println(p.getElement()); // for preorder, we visit p before exploring subtrees
   for (Position<E> c : children(p))
     preorder(c);
   return ;
  }
public void  postorder(Position<E> p) {

   for (Position<E> c : children(p))
     postorder(c);
 System.out.println(p.getElement());     // for postorder, we visit p after exploring subtrees
   return ;
  }

 public void breadthfirst() {
   if (!isEmpty()) {
     Queue<Position<E>> queue = new LinkedQueue<>();
     queue.enqueue(root());                  // start with the root
     while (!queue.isEmpty()) {
       Position<E> p = queue.dequeue();      // remove from front of the queue
       System.out.println(p.getElement());       // visit  this position
       for (Position<E> c : children(p))
         queue.enqueue(c);                   // add childrens to back of queue
     }
   }
   return;
    }

} //---------- end of VisitGenTree class ----------
```

Extend the class **Tree** to implement different traversal algorithms

Different visits of a Generic Tree

```
public class TestVisitGenTree
 public static void main(Stri
{
 int i=0;

  VisitGenTree gtree= new Vis

System.out.println("Avvio rie

gtree.addRoot("Elisabetta II"
Position root=gtree.root();
Position carlo=gtree.addChild
Position william=gtree.addChi
gtree.addChild(william,"Georg
gtree.addChild(william,"Charl
gtree.addChild(carlo,"Harry")
Position anna=gtree.addChild(
gtree.addChild(anna,"Peter");
gtree.addChild(anna,"Zara");
Position andrea=gtree.addChil
gtree.addChild(andrea,"Beatri
gtree.addChild(andrea,"Eugeni
Position edoardo=gtree.addChi
gtree.addChild(edoardo,"Lousi
gtree.addChild(edoardo,"James

System.out.println("Dimension

System.out.println("Stampa no
gtree.preorder(root);

System.out.println("Stampa no
gtree.postorder(root);

System.out.println("Stampa no
gtree.breadthfirst();

}
}
```

```
algoritmi miei — -bash — 104×54
Lousie
James
iMac-di-Rocco:algoritmi miei roccoaversa$ java TestVisitGenTree
Avvio riempimento albero
Dimensione btree= 15
Stampa nodi albero in preordine
Elisabetta II
Carlo
William
George
Charlotte
Harry
Anna
Peter
Zara
Andrea
Beatrice
Eugenia
Edoardo
Lousie
James
Stampa nodi albero in postordine
George
Charlotte
William
Harry
Carlo
Peter
Zara
Anna
Beatrice
Eugenia
Andrea
Lousie
James
Edoardo
Elisabetta II
Stampa nodi albero in ampiezza
Elisabetta II
Carlo
Anna
Andrea
Edoardo
William
Harry
Peter
Zara
Beatrice
Eugenia
Lousie
James
George
Charlotte
iMac-di-Rocco:algoritmi miei roccoaversa$
```