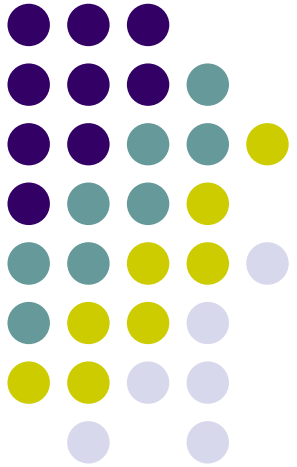
Binary Tree

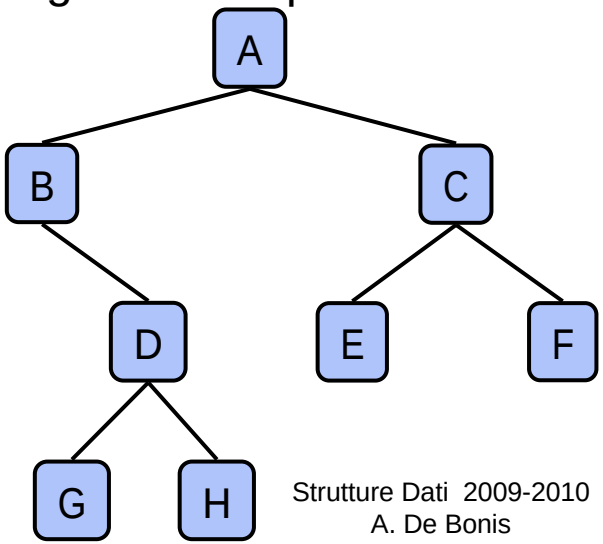
Corso: Strutture Dati

Docente: Annalisa De Bonis



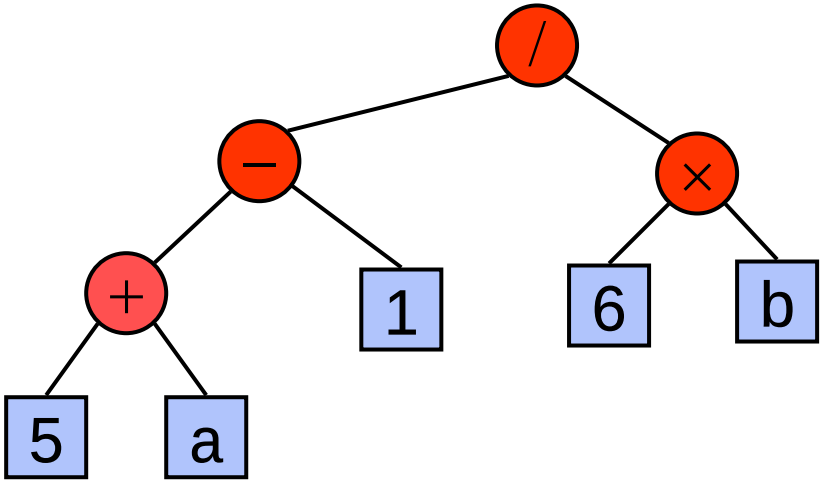
Albero binario

- Albero binario :
  - Ogni nodo interno ha al piu` due figli (esattamente due se l'albero e` un albero binario ***proprio***)
  - I figli di un nodo sono **una coppia ordinata**
- I figli di un nodo interno sono chiamati figlio destro e figlio sinistro
- Definizione alternativa (ricorsiva):
  - Un albero vuoto, oppure
  - Un albero costituito da una radice che ha una coppia ordinata di sottoalberi ognuno dei quali è un albero binario



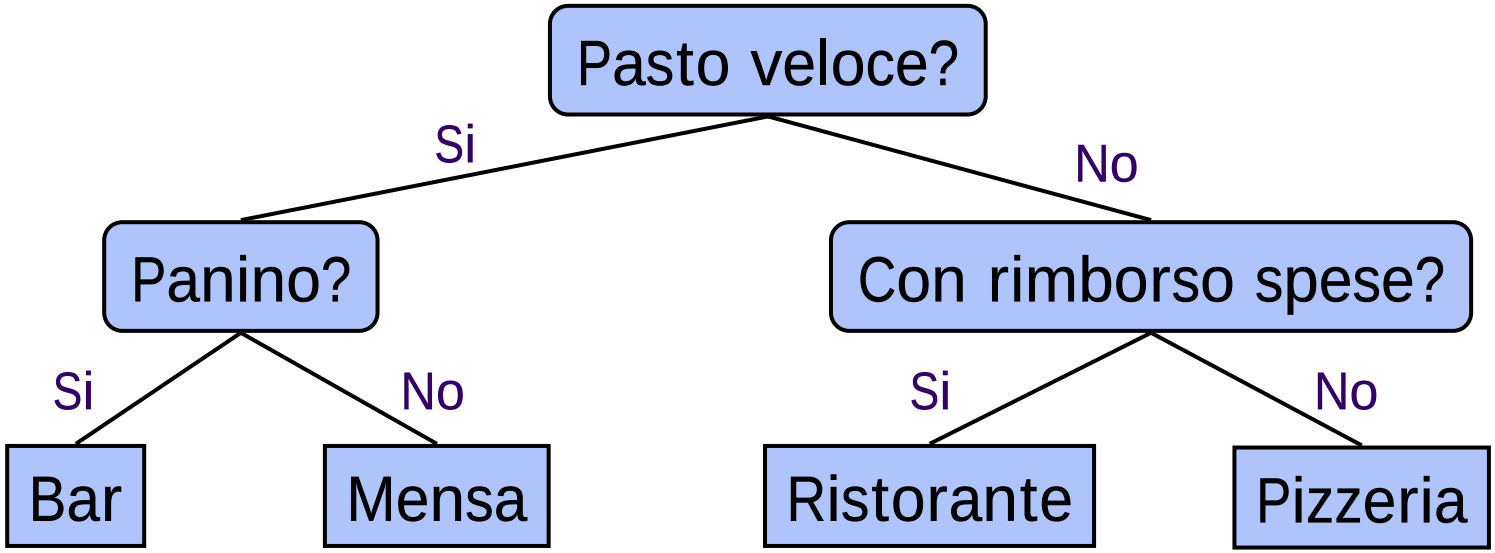
Applicazioni degli alberi binari:
albero sintattico per le espressioni

- Albero binario associato ad una espressione:
  - Nodi interni: operatori
  - Nodi esterni: operandi
- Esempio: $((5 + a) - 1) / (6 \times b)$



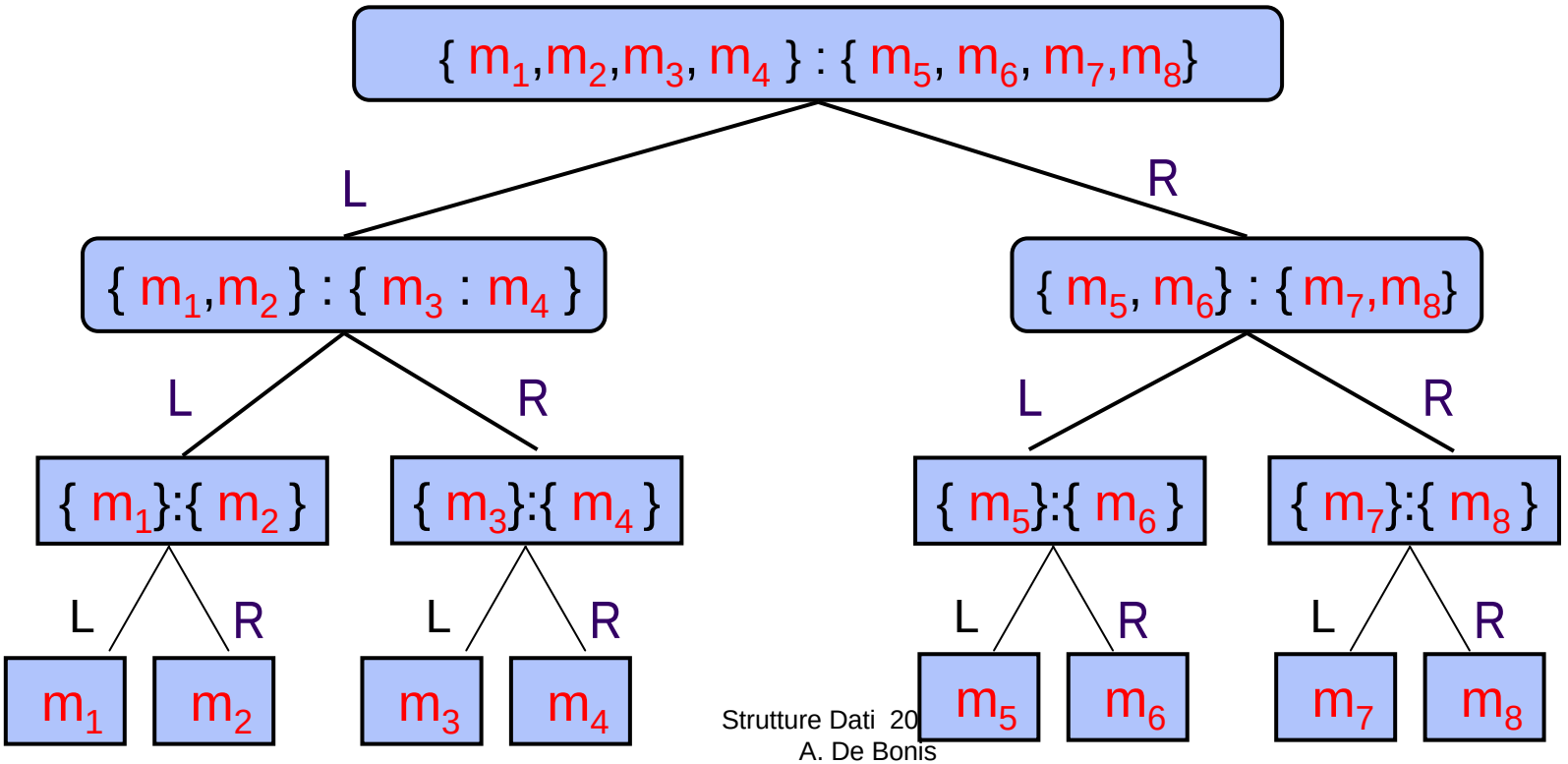
Applicazioni degli alberi binari:
Albero di decisione binario

- Albero binario associato ad un processo di decisione:
  - Nodi interni: domande con 2 possibili risposte
  - Nodi esterni: decisioni
- Esempio: schema di decisione per un pasto

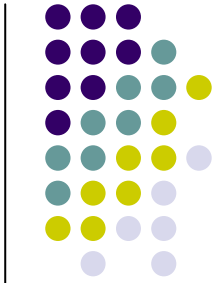


Albero di decisione binario

- **Esempio:** schema di decisione per trovare una moneta falsa (più pesante) tra 8 monete apparentemente identiche $\{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$
- usiamo una bilancia a due piatti
  - Possibile risposte: L = bilancia pende a sinistra
    R = bilancia pende a sinistra

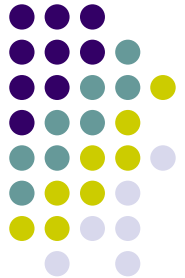


Il TDA Binary Tree



- Specializzazione del TDA Tree
- Metodi di accesso addizionali:
  - left(p)
    - Restituisce la posizione del figlio sinistro di p
    - Provoca un errore se p non ha figlio sinistro
  - right(p)
    - Restituisce la posizione del figlio destro di p
    - Provoca un errore se p non ha figlio destro
  - Metodi di interrogazione addizionali:
  - hasLeft (p)
    - Restituisce true sse p ha il figlio sinistro
  - hasRight (p)
    - Restituisce true sse p ha il filgio destro

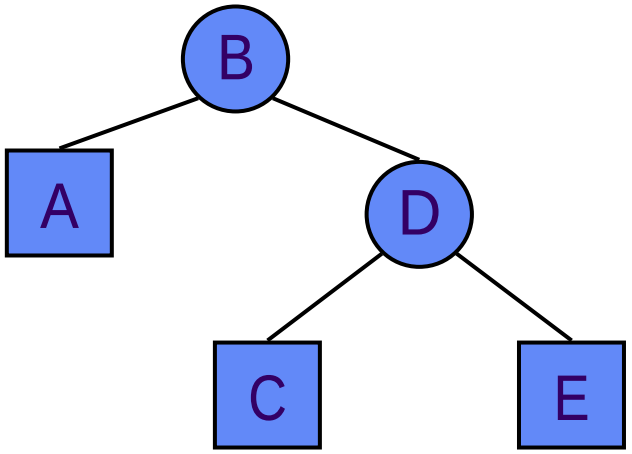
Interfaccia di BinaryTree in Java

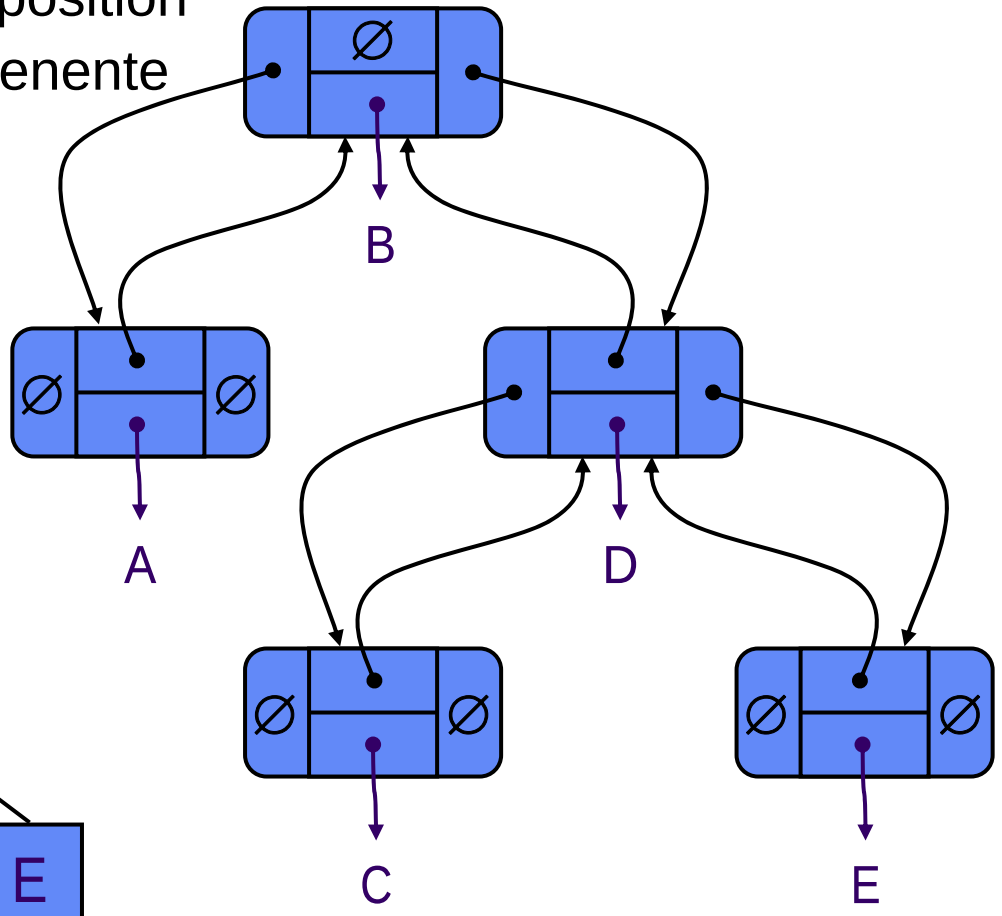


```
public interface BinaryTree<E> extends Tree<E> {

public Position<E> left(Position<E> v) throws
InvalidPositionException, BoundaryViolationException;

public Position<E> right(Position<E> v) throws
InvalidPositionException, BoundaryViolationException;

public boolean hasLeft(Position<E> v) throws
InvalidPositionException;

public boolean hasRight(Position<E> v) throws
InvalidPositionException;

}
```

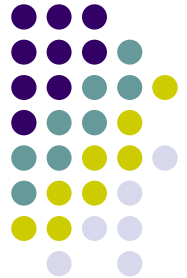
Implementazione BinaryTree con una struttura linkata

- Il nodo implementa il TDA position
- Un nodo è un oggetto contenente riferimenti a:
  - Elemento
  - Nodo genitore
  - Figlio sinistro
  - Figlio destro





L’interfaccia



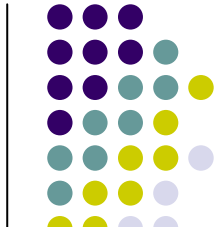
```
public interface BTPosition<E> extends Position<E> {
  public void setElement(E o);
  public BTPosition<E> getLeft();
  public void setLeft(BTPosition<E> v);
  public BTPosition<E> getRight();
  public void setRight(BTPosition<E> v);
  public BTPosition<E> getParent();
  public void setParent(BTPosition<E> v);
}
```

La classe BTNode

```
public class BTNode<E> implements BTPosition<E> {

private E element;

private BTPosition<E> left, right, parent;

public BTNode(E element, BTPosition<E> parent, BTPosition<E> left,
BTPosition<E> right) {

setElement(element);

setParent(parent);

 setLeft(left);

 setRight(right);

 }
```

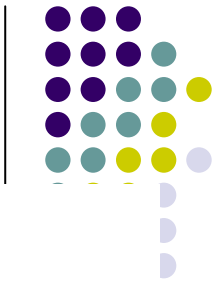
Continua nella prossima slide

La classe BTNode



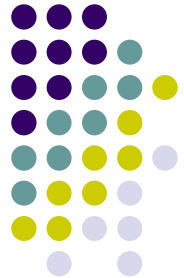
```
public E element() { return element; }
public void setElement(E o) { element=o; }
public BTPosition<E> getLeft() { return left; }
public void setLeft(BTPosition<E> v) { left=v; }
public BTPosition<E> getRight() { return right; }
public void setRight(BTPosition<E> v) { right=v; }
public BTPosition<E> getParent() { return parent; }
public void setParent(BTPosition<E> v) { parent=v; }
}
```

Implementazione di BinaryTree



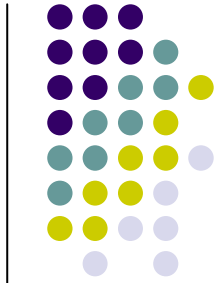
```
class LinkedBinaryTree <E> implements BinaryTree <E>{
// Variabili
    private int size; // numero di nodi
    private BTPosition<E> root;   // riferimento alla radice
// Costruttore
    public LinkedBinaryTree() {
        size = 0;
        root = null;
    }
// Metodi generici
…
// Metodi di interrogazione
…
// Metodi di accesso
…
// Metodi di aggiornamento
…
}
```

Il metodo positions() in LinkedBinaryTree



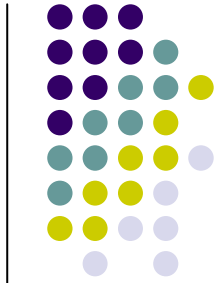
- positions() restituisce una collezione iterabile che contiene i nodi nell'ordine in cui sono visitati durante una visita preorder.
  - La visita richiede tempo O(n)

Il metodo PreorederPositions() invocato da positions()



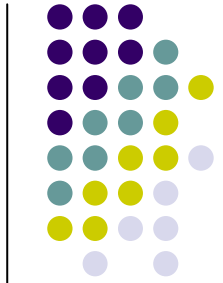
```
//effettua la visita preorder e memorizza i nodi visitati in una lista
protected void preorderPositions(Position<E> v,PositionList<Position<E>> pos)
        throws InvalidPositionException {
    pos.addLast(v);
    if (hasLeft(v)) preorderPositions(left(v), pos);
   if (hasRight(v)) preorderPositions(right(v), pos);
}
```

Il metodo Positions()



```
public Iterable<Position<E>> positions() {
   PositionList<Position<E>> positions = new
    NodePositionList<Position<E>>();
    if(size != 0)
      preorderPositions(root(), positions);
  return positions;
  }
```

Il metodo iterator()

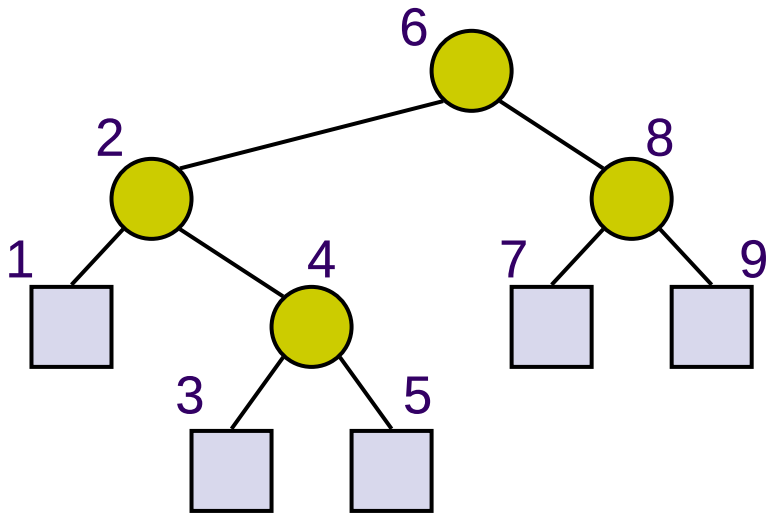


```
public Iterator<E> iterator() {
    Iterable<Position<E>> positions = positions();
   PositionList<E> elements = new NodePositionList<E>();
  Iterator<Position<E>> it=positions.iterator();
   while(it.hasNext())
     elements.addLast(it.next().element());
 return elements.iterator();
}
```

Alberi binari: Visita Inorder

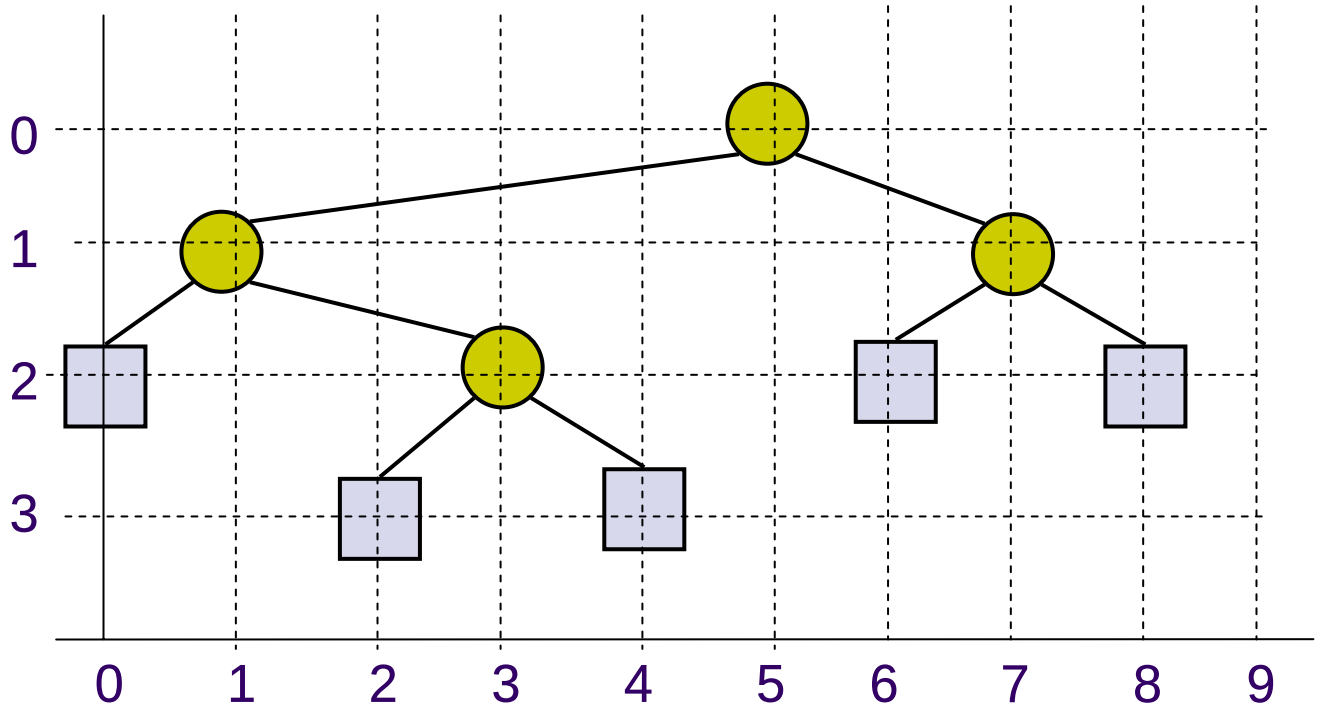
- Un nodo è ispezionato dopo il suo sotto-albero sinistro e prima del suo sotto-albero destro

```
Algorithm inOrder(v)
    if hasLeft (v)=true
        inOrder (leftChild (v))
    visit(v)
    if hasRight (v)=true
        inOrder (rightChild (v))
```

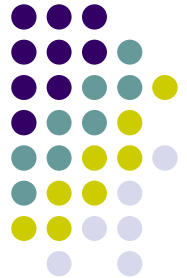


Applicazioni della visita inorder

- E` possibile disegnare un albero binario associando ad ogni nodo due coordinate
  - x(v) = # nodi visitati prima di v dalla visita inorder
  - y(v) = profondità di v

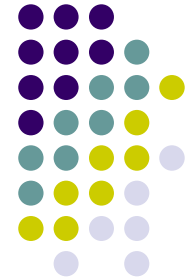


Esercizi



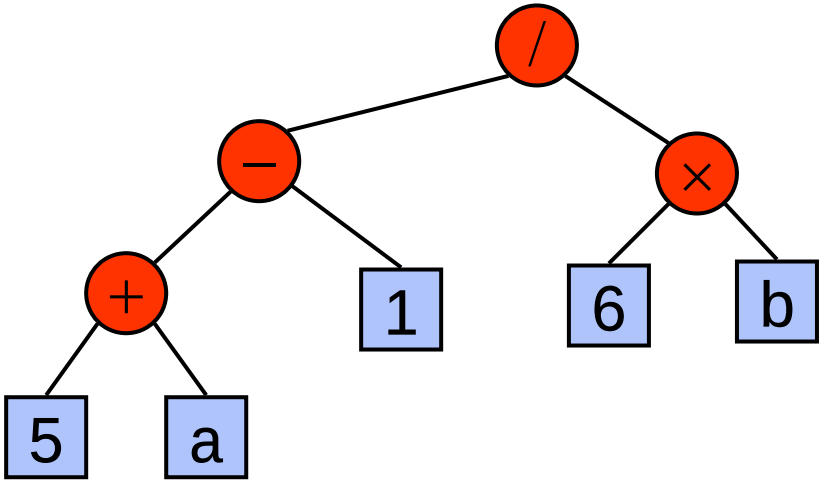
- Completare l'implementazione di BinaryTree con la classe LinkedBinaryTree.
- Testare la classe LinkedBinaryTree provando tutti i metodi
- Discutere complessità computazionale dei metodi
- Scrivere un metodo che cloni un albero binario
- Scrivere un metodo InsertLeft che prende in input un nodo N e un nodo p e rende N figlio sinistro di p.
- Scrivere un metodo InsertRight che prende in input un nodo N e un nodo p e rende N figlio destro di p.

Stampa di espressioni aritmetiche



- Specializzazione di una visita inorder
  - stampa operando o operatore quando si ispeziona il nodo
  - stampa “(“ prima di visitare il sotto-albero sinistro
  - stampa “)“ dopo aver visitato il sottoalbero destro

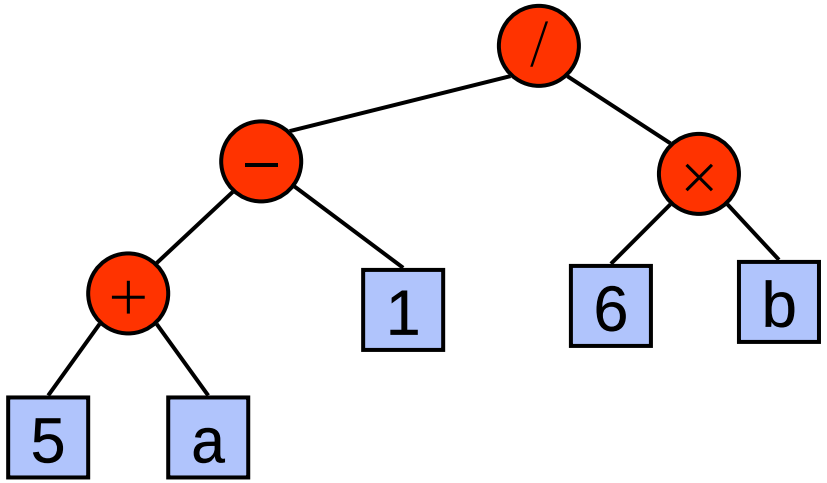
(((5 + a) - 1) / (6 × b))




```
Algorithm printExpression(v)
    if isInternal (v)
        print(“(”)
        printExpression (leftChild (v))
    print(v.element ())
    if isInternal (v)
        printExpression(rightChild (v))
        print (“)”)
```

Valutazione di espressioni aritmetiche

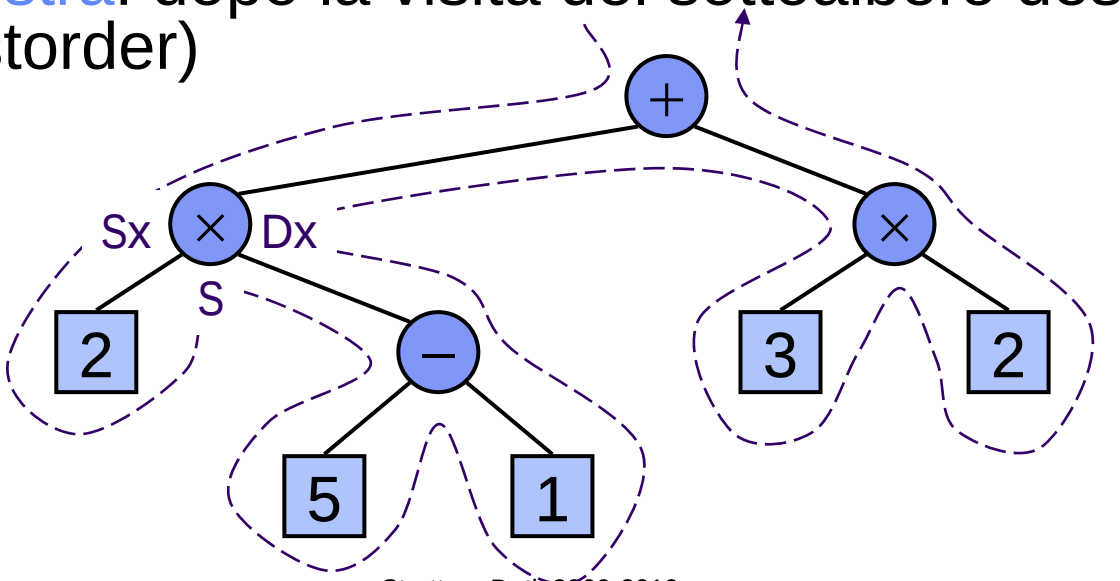
- Specializzazione di visita postorder
  - chiamate ricorsive calcolano valori sotto-alberi
  - ispezionando un nodo, combina i risultati ottenuti dai sottoalberi



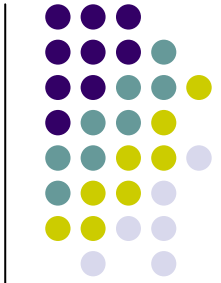

```
Algorithm evalExpr(v)
    if isExternal (v)
        return v.element ()
    else
        x ← evalExpr(leftChild (v))
        y ← evalExpr(rightChild (v))
        ◊ ← operator stored at v
        return x ◊ y
```

Cammino Euleriano

- E` una visita generica di un albero binario
- Casi speciali: visita preorder, postorder e inorder
- Attraversa l'albero ispezionando ogni nodo tre volte:
  - a sinistra: prima della visita del sottoalbero sinistro (preorder)
  - da sotto: tra le visite ai due sottoalberi (inorder)
  - a destra: dopo la visita del sottoalbero destro (postorder)

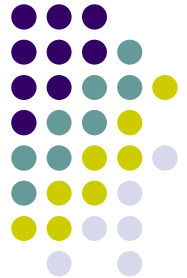


Algoritmo eulerTour



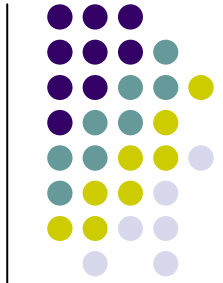
Algorithm *eulerTour*(*T*,*v*)
 effettua visita di *v* a sinistra
 if *v* ha un figlio sinistro *u* **then**
  eulerTour(*T*,*u*)
effettua visita di *v* da sotto
if *v* ha un figlio destro *w* **then**
  eulerTour(*T*,*w*)
effettua visita di *v* a destra

Design pattern: Template di metodi



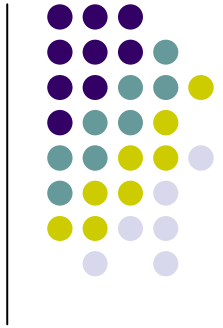
- Algoritmo generico che può essere specializzato ridefinendo alcuni metodi
  - Implementato con una *abstract class* di Java
- Esempio: template method EulerTour
  - Usa metodi di visita che possono essere ridefiniti da sottoclassi
  - EulerTour è chiamato ricorsivamente sul figlio sinistro e destro
  - Un oggetto TourResult con campi left, right e out mantiene i risultati delle chiamate ricorsive a EulerTour

La classe TourResult



```
/* I campi left e right servono a tenere traccia dei
   risultati della chiamate di eulerTour sul figlio sinistro
   e sul figlio destro di v.
  Il campo out tiene traccia del valore computato dalla
   chiamata di eulerTour su v
*/
public class TourResult<R> {
   public R left;
   public R right;
   public R out;
 }
```

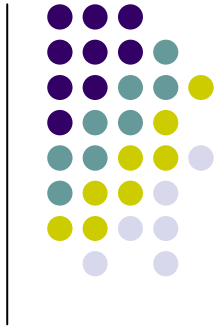
Template Method Euler Tour



```
public abstract class EulerTour<E, R> {

    protected BinaryTree<E> tree;

/** Esecuzione della visita. Questo metodo astratto deve essere
   specificato in una sottoclasse concreta*/

    public abstract R execute(BinaryTree<E> T);

/** Inizializzazione della visita */

     protected void init(BinaryTree<E> T) {

         tree = T;

      }
```

Continua nella slide successiva

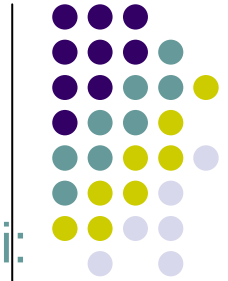
Template Method Euler Tour



```
/** Metodo template */
 protected R eulerTour(Position<E> v) {
 TourResult<R> r = new TourResult<R>();
  visitLeft(v, r);
 if (tree.hasLeft(v)) r.left = eulerTour(tree.left(v));
  visitBelow(v, r);
 if (tree.hasRight(v)) r.right = eulerTour(tree.right(v));
  visitRight(v, r);
return r.out;
  }
```

Continua nella slide successiva

Template Method Euler Tour



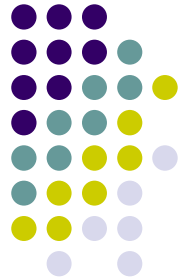
```
// Metodi ausiliari che possono essere ridefiniti nelle sottoclassi:

/** Metodo invocato per la visita a sinistra */

   protected void visitLeft(Position<E> v, TourResult<R> r) {}

 /** Metodo invocato per la visita da sotto */

     protected void visitBelow(Position<E> v, TourResult<R> r) {}

/** Metodo invocato per la visita a destra */

    protected void visitRight(Position<E> v, TourResult<R> r) {}

 }
```

Specializzazione di EulerTour per valutare espressioni

- Assunzioni
  - Nodi interni contengono operatori
  - Nodi esterni contengono operandi
- Il parametro E di BinaryTree e` istanziato con il tipo ExpressionTerm

```
public class ExpressionTerm {
    public Integer getValue() { return 0; }
    public String toString() {
    return new String("");
  }
}
```

Specializzazione di EulerTour per valutare espressioni



I nodi esterni contengono oggetti di tipo ExpressionVariable

```
public class ExpressionVariable extends ExpressionTerm {
 protected Integer var;
 public ExpressionVariable(Integer x) { var = x; }
 public void setVariable(Integer x) { var = x; }
 public Integer getValue() { return var; }
public String toString() { return var.toString(); }
 }
```

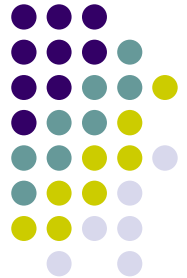
Specializzazione di EulerTour per valutare espressioni

I nodi interni contengono oggetti di tipo ExpressionOperator

```
public class ExpressionOperator extends ExpressionTerm {
 protected Integer firstOperand, secondOperand;
public void setOperands(Integer x, Integer y) {
 firstOperand = x; secondOperand = y;
}
}
```

```
public class AdditionOperator extends ExpressionOperator {
  public Integer getValue() { return (firstOperand + secondOperand);
}
 public String toString() { return new String("+"); } }
```

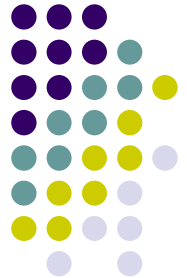
Specializzazione di EulerTour per valutare espressioni



```
public class EvaluateExpressionTour extends EulerTour <ExpressionTerm,
Integer> {
public Integer execute(BinaryTree<ExpressionTerm> T) {
   init(T);
  return eulerTour(tree.root());
}
```

Continua nella prossima slide

Specializzazione di EulerTour per valutare espressioni



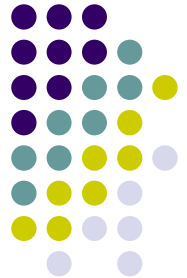
```
protected void visitRight(Position<ExpressionTerm> v,
TourResult<Integer> r)
{
 ExpressionTerm term = v.element();
 if (tree.isInternal(v)) {
   ExpressionOperator op = (ExpressionOperator) term;
   op.setOperands(r.left, r.right);
 }
 r.out = term.getValue(); }
 }
```

Specializzazione di EulerTour per stampare espressioni

```
public class PrintExpressionTour extends EulerTour<ExpressionTerm, String>
{
public String execute(BinaryTree<ExpressionTerm> T) {
  init(T);
  System.out.print("Espressione: ");
  eulerTour(T.root());
  System.out.println();
  return null; //niente da restituire }
```

Continua nella prossima slide

Specializzazion: di EulerTour per stampare espressioni



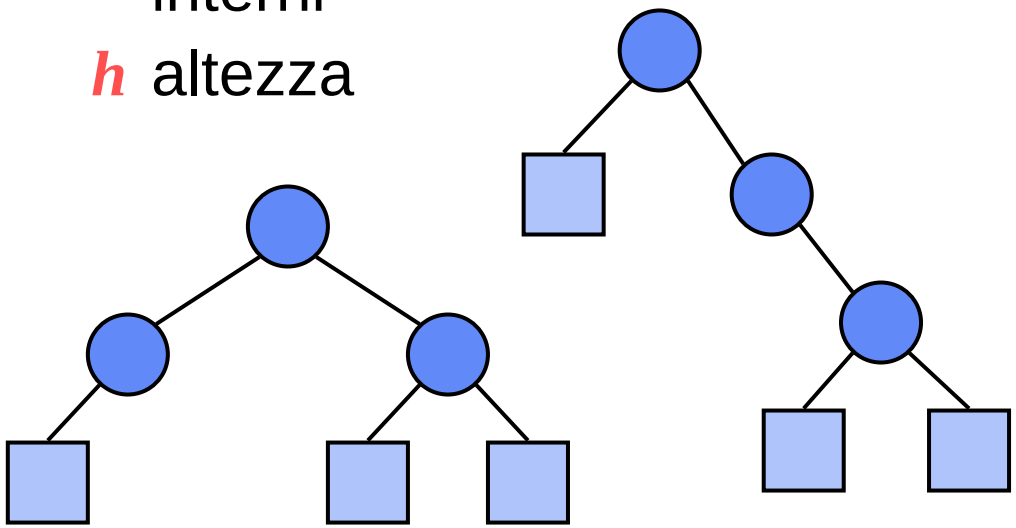
```
protected void visitLeft(Position<ExpressionTerm> v,
TourResult<String> r)
{
if (tree.isInternal(v)) System.out.print("("); }

protected void visitBelow(Position<ExpressionTerm> v,
TourResult<String> r) { System.out.print(v.element()); }

protected void visitRight(Position<ExpressionTerm> v,
TourResult<String> r) {
if (tree.isInternal(v)) System.out.print(")"); } }
```

Esercizio

- Scrivere la classe che specializza EulerTour per computare e stampare l'altezza di ciascun nodo di un albero binario

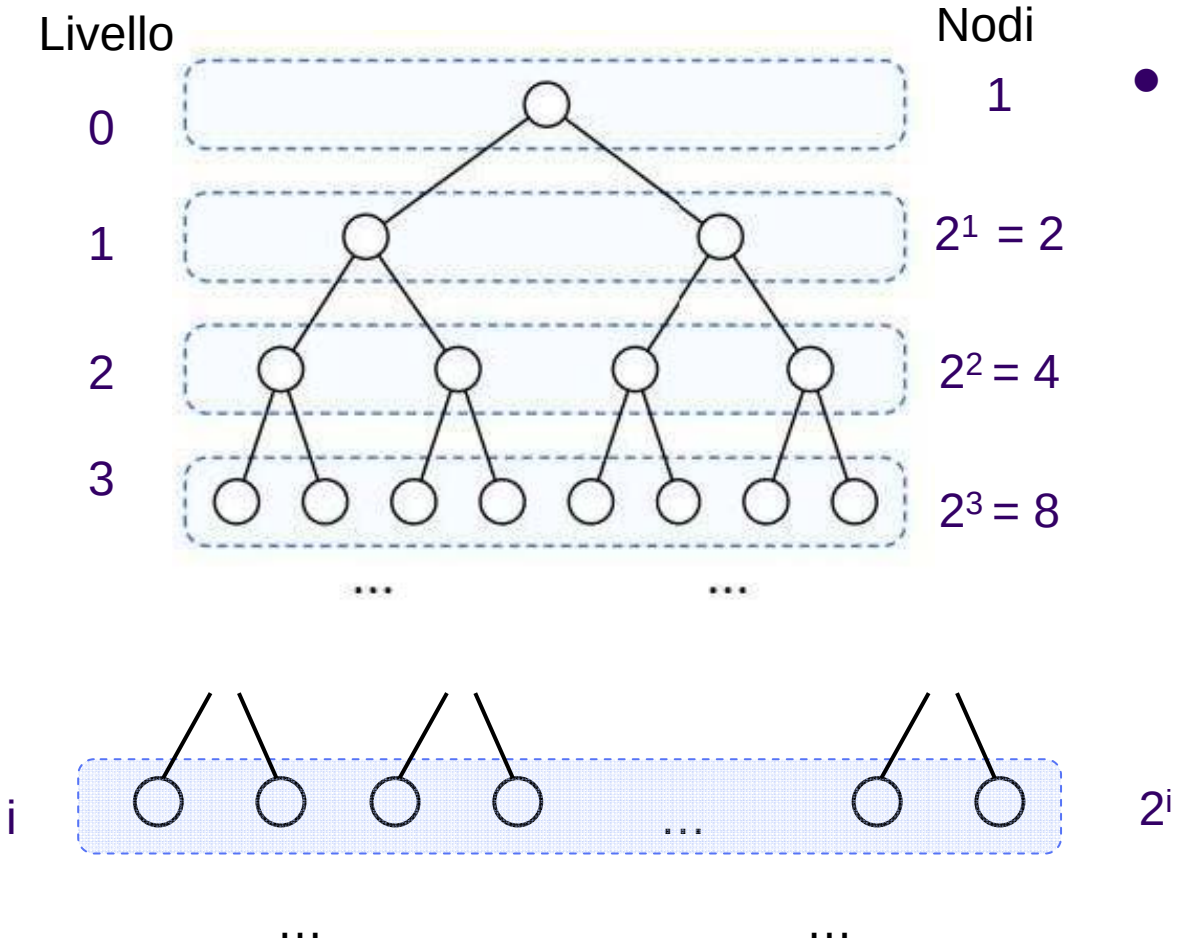
Proprietà alberi binari

- Notazione
  - n numero di nodi
  - e numero di nodi esterni
  - i numero di nodi interni
  - h altezza



- Proprietà:
  - $e \leq i + 1$
  - $n = i + e \geq 2e - 1$
  - $h \leq i$
  - $h \leq n - 1$

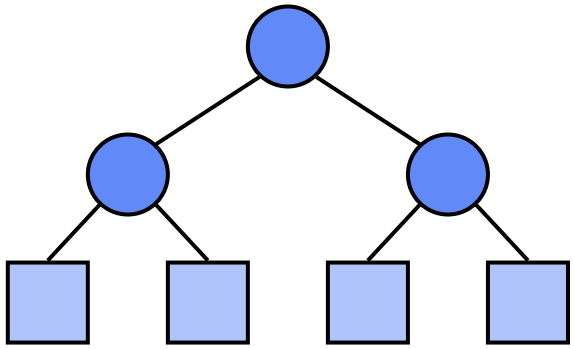
Proprietà alberi binari

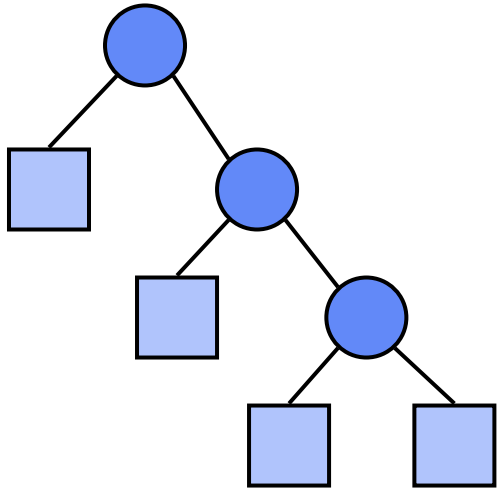


- Proprietà:
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $i \leq \sum_{i=0}^{h-1} 2^i = 2^h - 1$
  - $n = i + e \leq 2^{h+1} - 1$
  - $h \geq \log_2 (n + 1) - 1$

Proprietà alberi binari propri

- Notazione
  - n numero di nodi
  - e numero di nodi esterni
  - i numero di nodi interni
  - h altezza



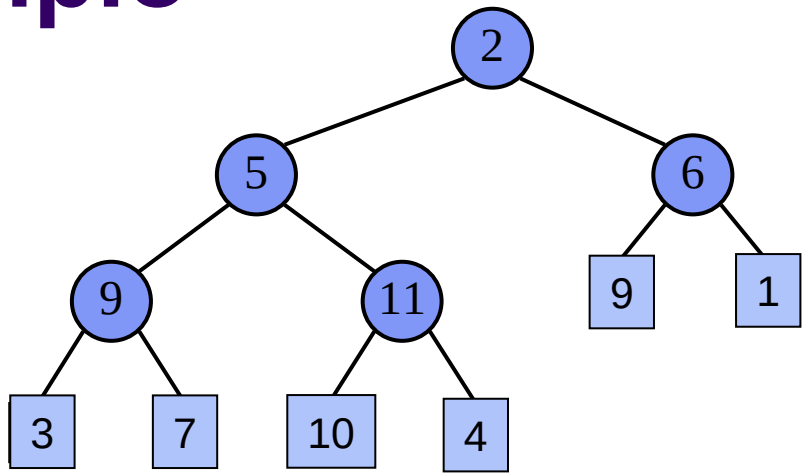


- Proprietà:
  - $e = i + 1$
  - $n = e + i = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $i \leq 2^h - 1$
  - $n = i + e \leq 2^{h+1} - 1$
  - $h \geq \log_2 (n + 1) - 1$

Implementazione BinaryTree: vettori

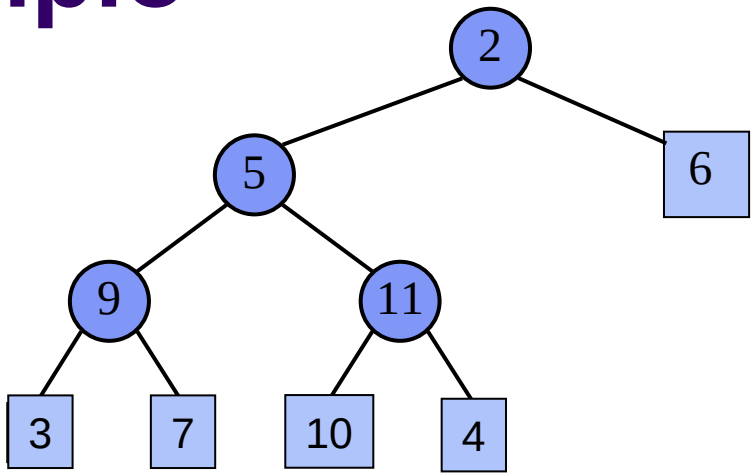
- #(u) è l'indice dell'array che contiene il riferimento al nodo u
  - Se u è radice, allora #(u)=1
  - Se u è il figlio sinistro di v, allora #(u)=2 #(v)
  - Se u è il figlio destro di v, allora #(u)=2 #(v) +1
- L'elemento di indice 0 non utilizzato
- Se l'albero non è pieno ci possono essere celle dell'array non utilizzate
  - Caso pessimo dimensione array 2^n

Esempio



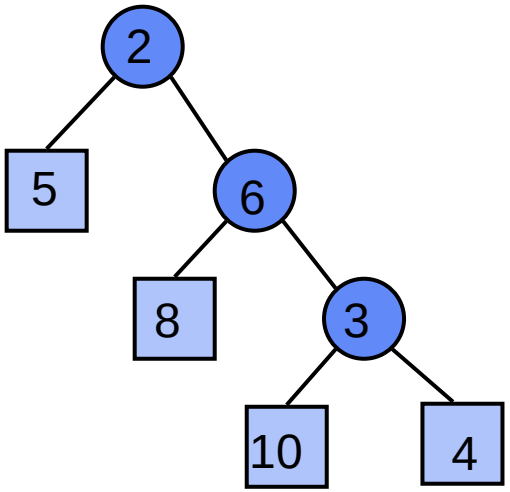
	2	5	6	9	11	9	1	3	7	10	4
0	1	2	3	4	5	6	7	8	9	10	11

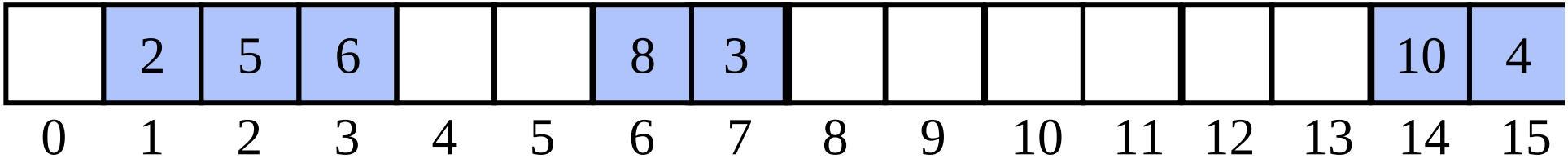
Esempio



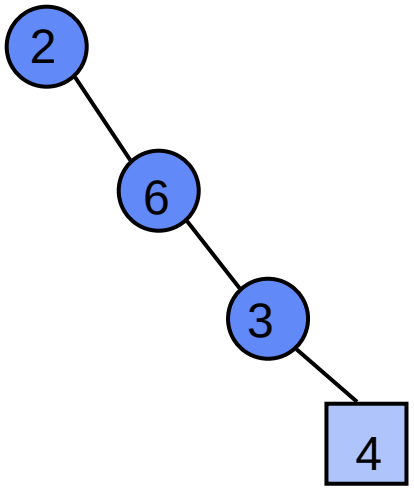
	2	5	6	9	11			3	7	10	4
0	1	2	3	4	5	6	7	8	9	10	11

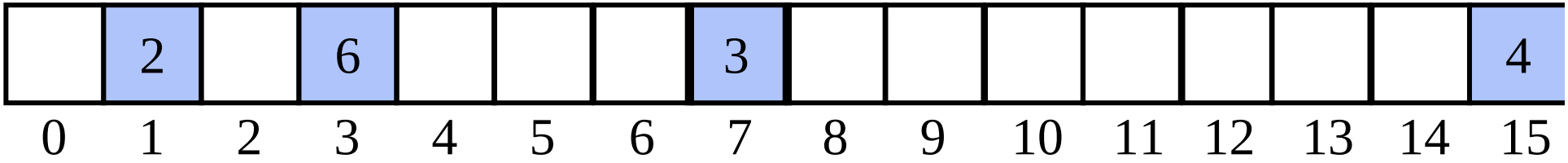
Esempio



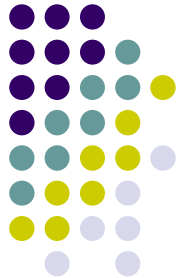


Esempio





Esercizio

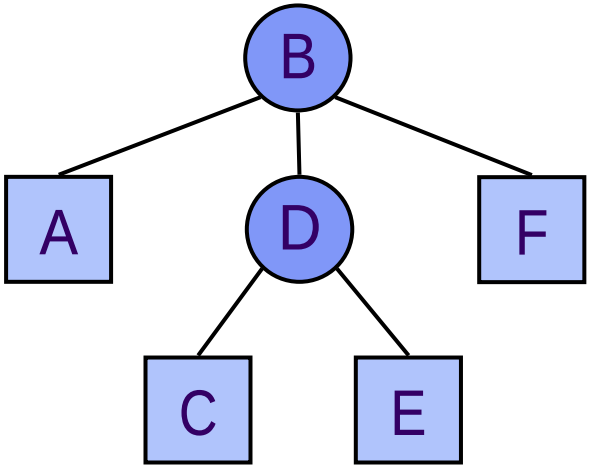


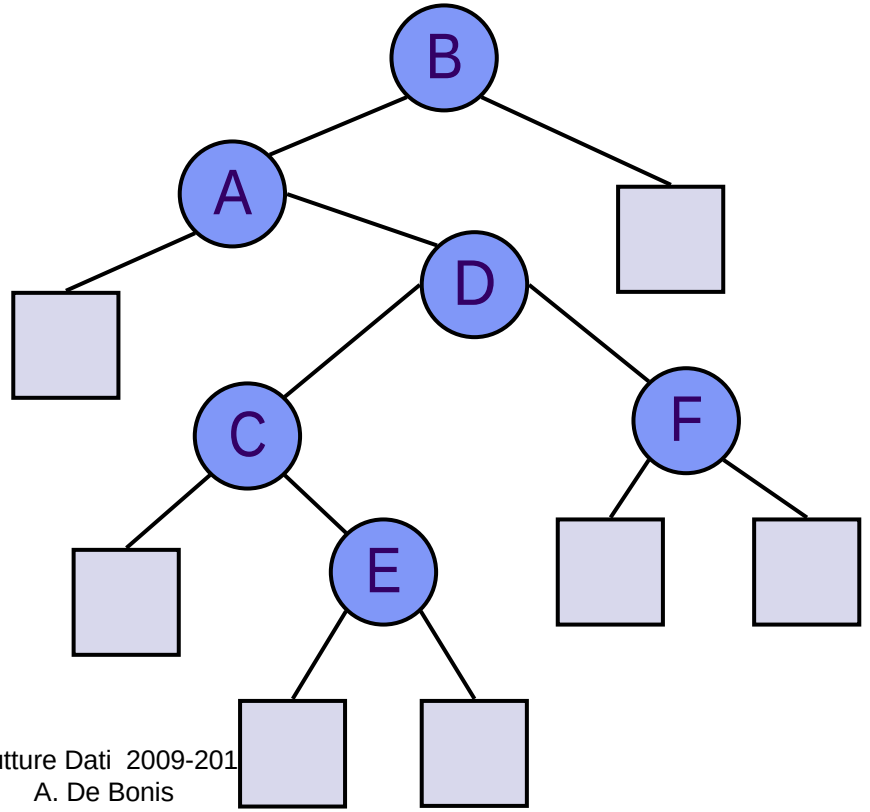
- Implementare l'interfaccia BinaryTree mediante un vettore

```
public class VectorBinaryTree<E>
                    implements BinaryTree<E> {
                    ……
                }
```

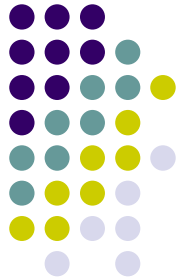
Implementazione di Tree con un albero binario proprio (LeftChild-RightSibling)

- primo figlio di u → figlio sinistro di u
- Il primo fratello a destra di u →figlio destro di u





Esercizio



- Implementare il TDA Tree mediante un BinaryTree