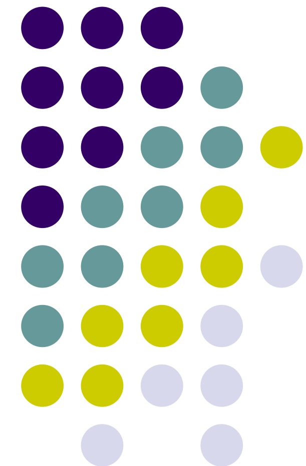
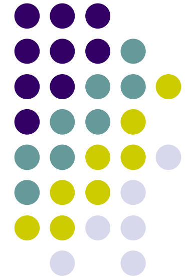
Tree

Corso: Strutture Dati

Docente: Annalisa De Bonis

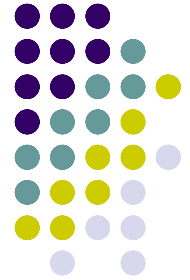


Concetto di albero

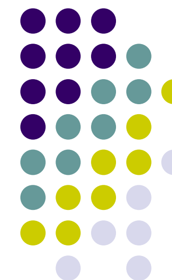


- Astrae il concetto di gerarchia
- Un albero consiste di un insieme di nodi tra cui vi è una relazione del tipo padre-figlio che soddisfa le seguenti proprietà:
  - se **T** non è vuoto allora contiene un nodo speciale chiamato **radice** che non ha padre
  - ciascun nodo diverso dalla radice ha un unico padre; i nodi aventi un certo nodo w come padre si dicono figli di w

Applicazioni

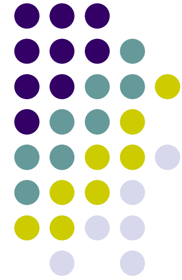


- La struttura ad albero rappresenta un modo naturale per organizzare alcuni tipi di informazioni:
  - Organigramma aziendali
  - File system
  - Siti Web
  - Ecc.

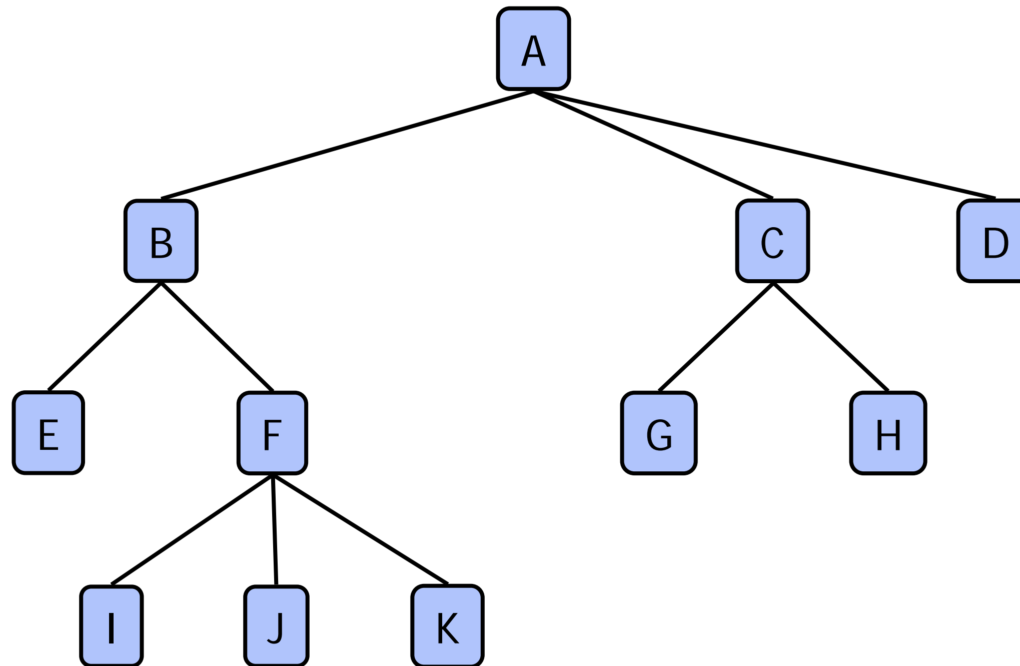


Esempio

- Electronics Srl
  - Vendite
    - EU
    - Non EU
      - Africa
      - Russia
      - Asia
  - Produzione
    - Calcolatrici
    - PC
  - Ricerca e Sviluppo

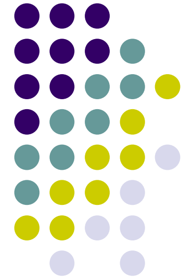


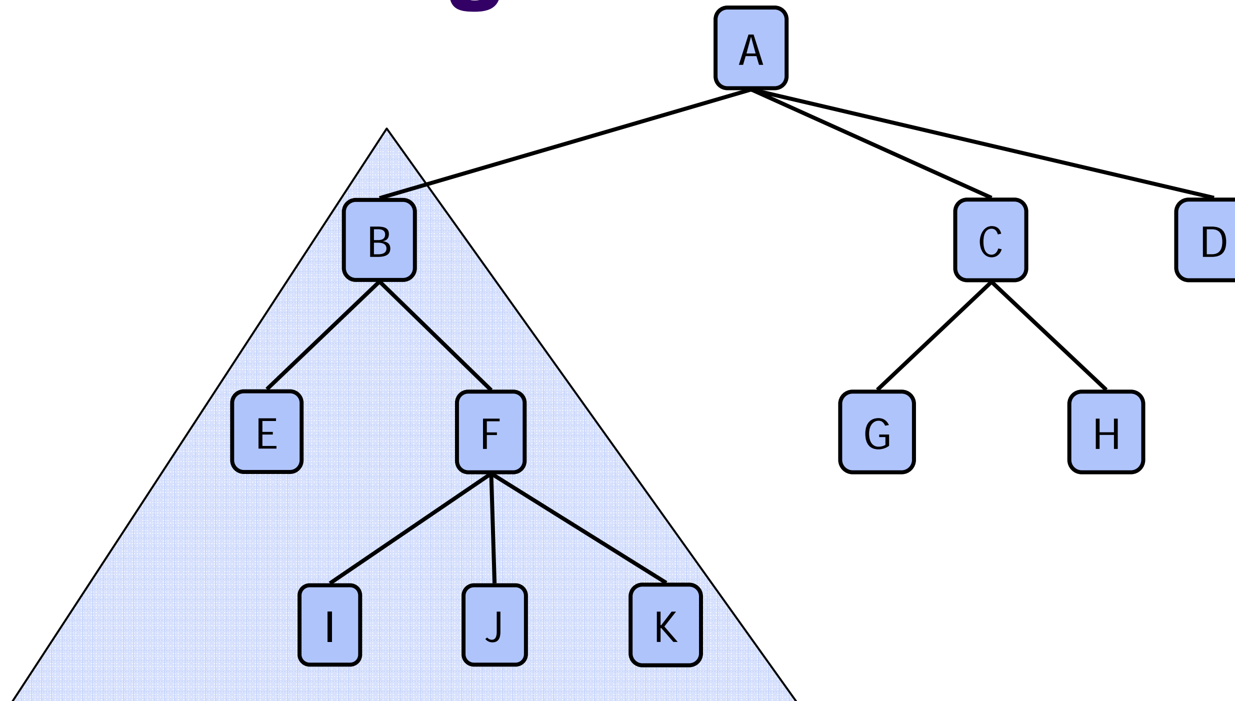
Terminologia



Radice: nodo senza genitore (A)
Nodo interno: nodo con almeno un figlio (A,B,C,F)
Foglia (nodo esterno): nodo senza figli (E, I, J, K, G, H, D)

Terminologia

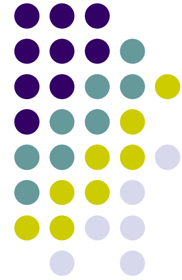


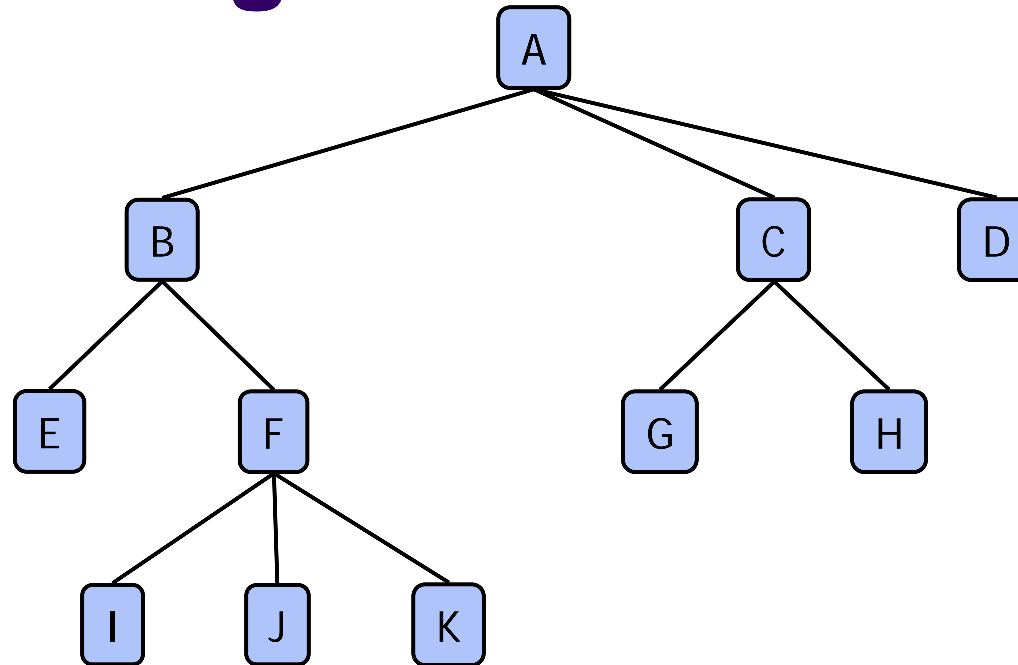


Discendente di un nodo: il nodo stesso o discendente di un figlio del nodo (discendenti di B : B,E,F,I,J,K)

Sottoalbero: albero formato da un nodo e tutti i suoi discendenti (sottoalbero con radice in B: B,E,F,I,J,K)

Terminologia

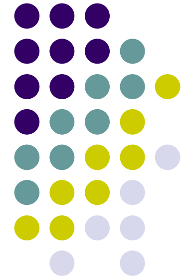


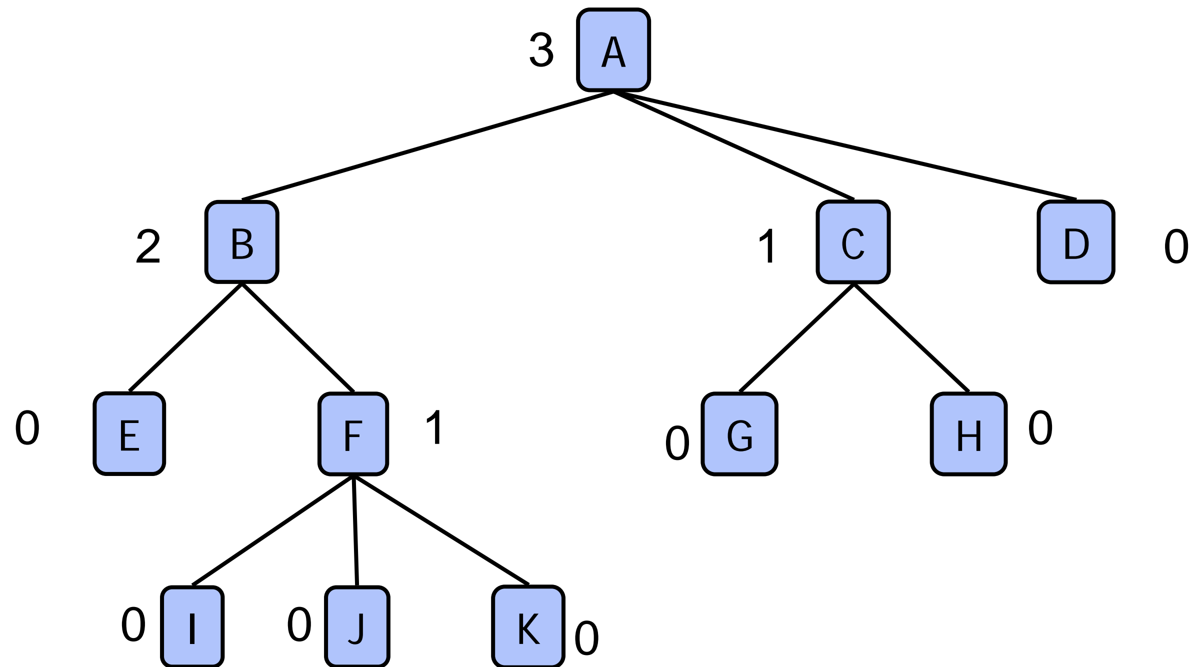


Antenato: il nodo stesso o antenato del genitore del nodo (G:G, A,C)

Profondità di un nodo: numero di antenati del nodo -1 (G: 2)

Terminologia



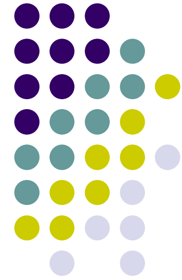


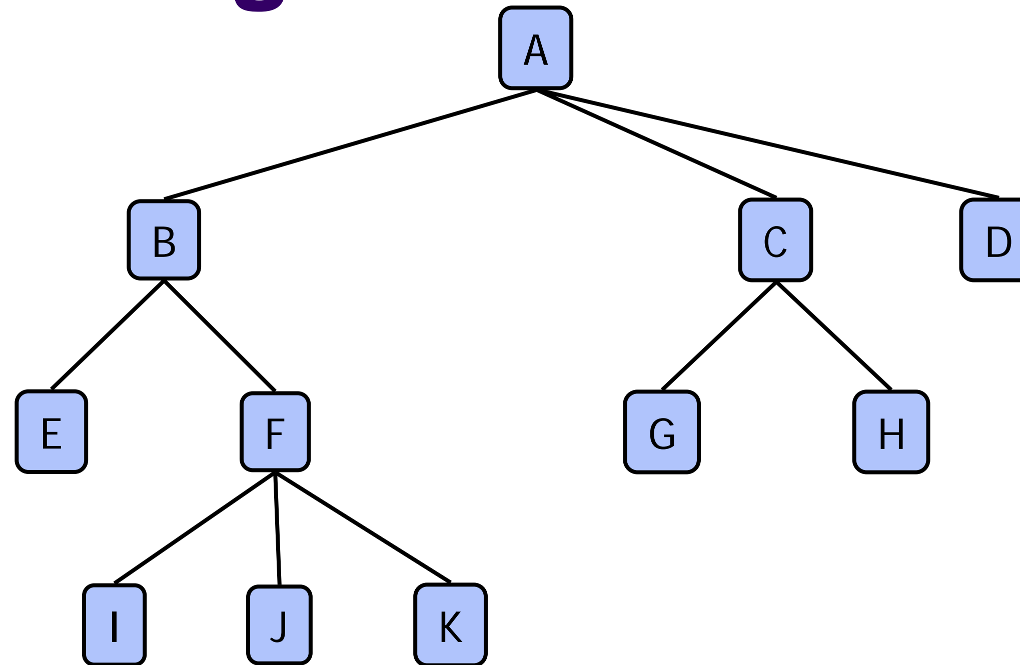
Altezza di un nodo: 0 se il nodo è una foglia;

altrimenti 1 + max altezza figli del nodo (C: 1)

Altezza di un albero non vuoto = altezza radice (3)

Terminologia



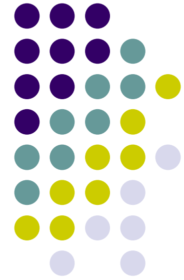


Arco: coppia di nodi che sono in relazione padre-figlio

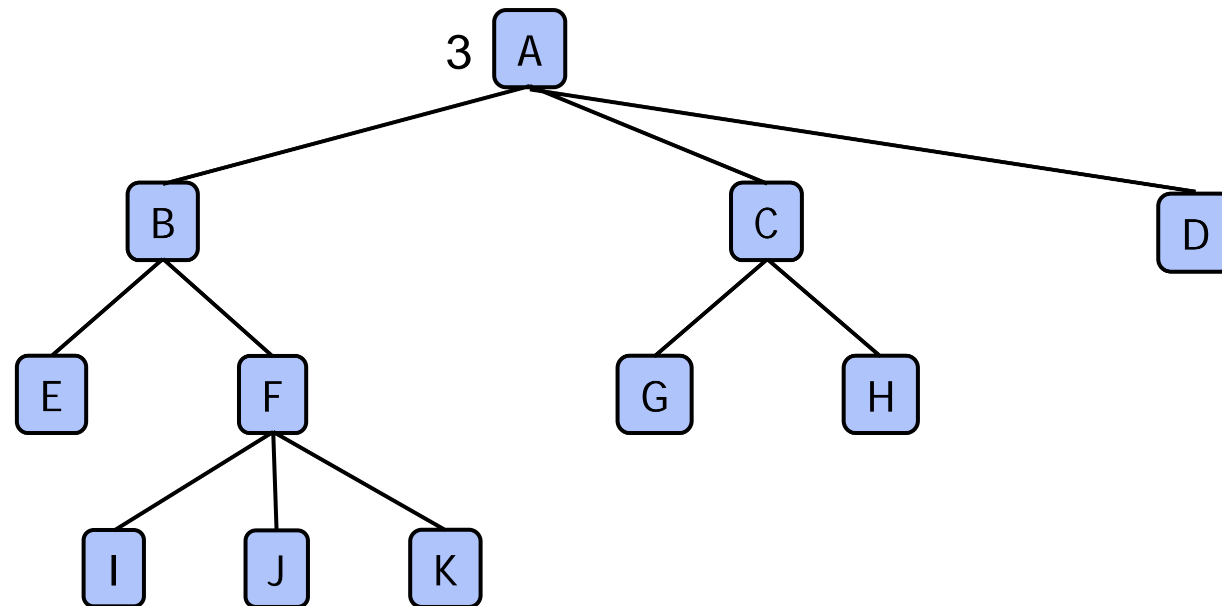
Percorso: sequenza di nodi in cui due nodi consecutivi sono in relazione padre-figlio (percorso da B a J: B,F,J)

Profondità nodo = numero di archi formati dai nodi lungo il percorso dalla radice al nodo (profondità di G: 2)

Proposizione

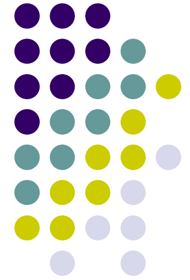


- Altezza di un albero non vuoto = profondità massima delle foglie
  - Giustificazione: Altezza dell'albero = numero di archi lungo il percorso dalla radice alla foglia più distante



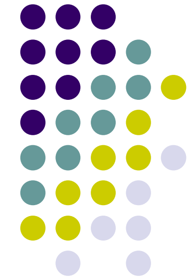
- Si può dimostrare per induzione sul numero di nodi

Proposizione



- $\sum_{v:\ v \text{ discendente di } w} c_v$ = numero discendenti di w-1
  - Giustificazione:
    - $\sum_{v:\ v \text{ discendente di } w} c_v$ = numero totale di figli di tutti i discendenti di w
    - I nodi che danno un contributo alla somma sono solo i figli dei discendenti di w
    - Ad esclusione di w, ciascun discendente di w è figlio di un (unico) discendente di w e quindi viene contato esattamente una volta nella sommatoria
      → ad esclusione di w, ciascun discendente di w dà un contributo pari ad 1 alla somma

Alberi ordinati



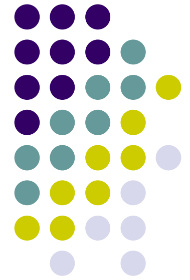
Libro

Capitolo 1 | Capitolo 2 | … | Capitolo 10

Paragrafo 1.1 | … | Paragrafo 1.5 | Paragrafo 2.1 | … | Paragrafo 2.10 | … …

… … … … … … … …

Calcolo della profondità di un nodo



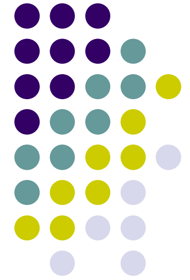
Algorithm depth(T,v)

```
if v è la radice then
    return 0
else
    parent ← padre di v
    return 1+depth(T,parent)
```

Se escludiamo il tempo impiegato per la chiamata ricorsive, depth(T,v) impiega tempo O(1)

Sommando su tutti gli antenati di v ➔ tempo totale = O(profondità di v)

Calcollo dell'altezza di un nodo

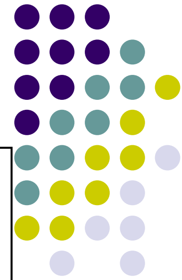


```
Algorithm height(T,v)
    if v è una foglia return 0
    else
      h ← 0
      for ciascun figlio f di v do
        hf←height(T,f)
        if hf>h then h←hf
    return h+1
```

- Se escludiamo il tempo impiegato per le chiamate ricorsive, height(T,v) impiega tempo $O(1+ c_v)$, dove c_v è il numero di figli di v
- Se invoco la prima volta la funzione sul nodo w, la funzione viene invocata ricorsivamente su tutti i discendenti di w

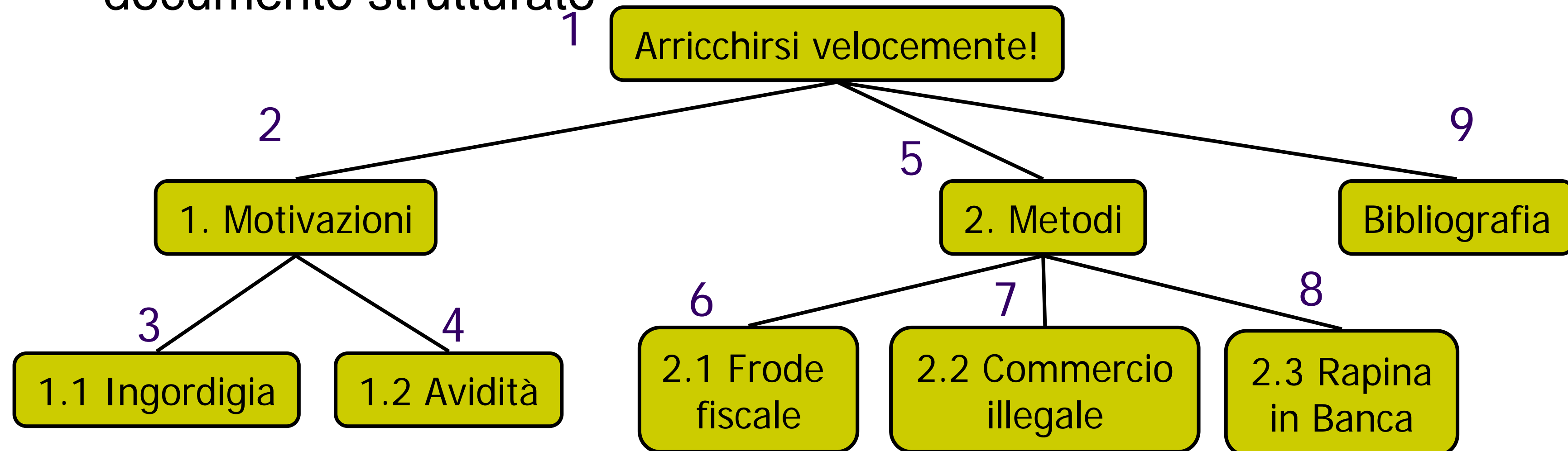
➔ Tempo totale= $\sum_{v:\ v \text{ discendente di } w} O(1+ c_v)$ = O(num. discendenti di w)

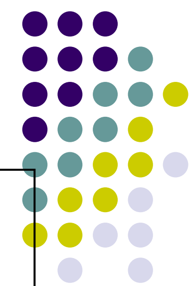
Visita Preorder



- Una visita di un albero attraversa tutti i suoi nodi in qualche ordine
- In una visita preorder, un nodo è ispezionato prima dei suoi discendenti
- Applicazione: stampa di un documento strutturato

Algorithm *preOrder*(*v*)
  visit(*v*)
  if(*v* è una foglia)
    return
  for each figlio ***w*** di ***v***
    preOrder (*w*)

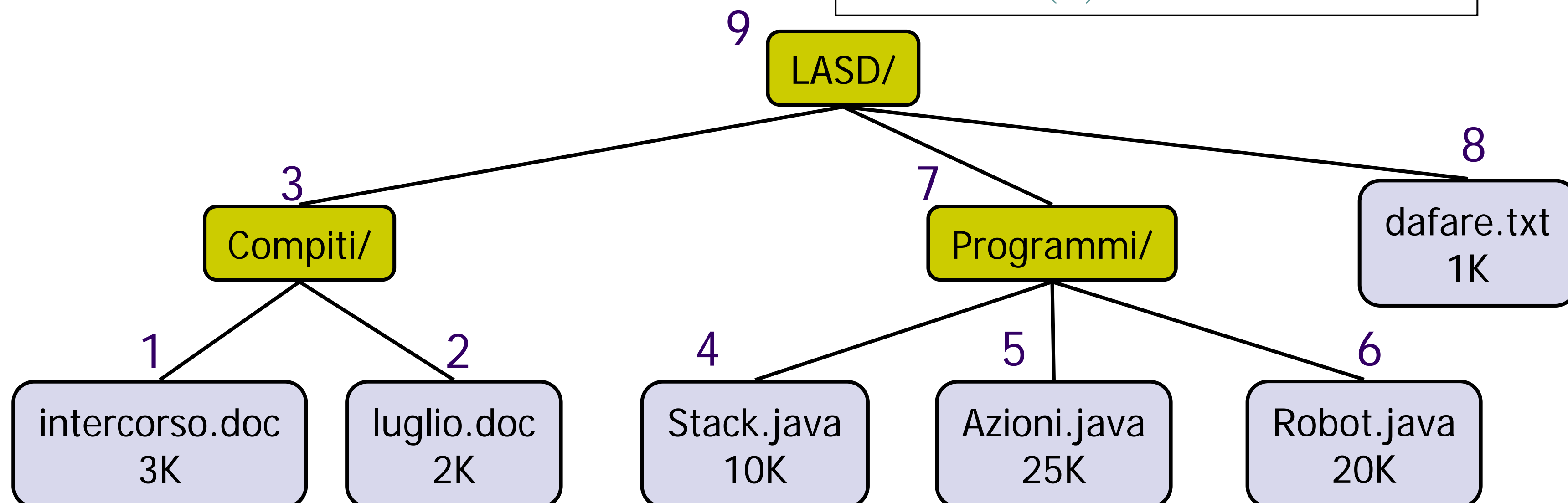




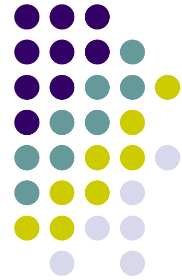
Visita Postorder

- Un nodo è ispezionato dopo i suoi discendenti
- Applicazione:
  - calcola lo spazio usato dai file in directory e sotto-directory

```
Algorithm postOrder(v)
    if v e` una foglia
        visit(v)
        return
    for each figlio w di v
        postOrder (w)
    visit(v)
```



Analisi degli algoritmi preOrder e postOrder

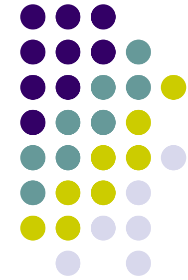


- Se escludiamo il tempo impiegato per le chiamate ricorsive, l'algoritmo impiega tempo $O(1+ c_v)$, dove c_v è il numero di figli di v
- Se cominciamo la visita dal nodo w, l'algoritmo viene richiamato ricorsivamente su tutti i discendenti di w

➔ Tempo totale = $\sum_{v:\ v \text{ discendente di } w} O(1+ c_v)$ = O(num. nodi sottoalbero radicato in w)

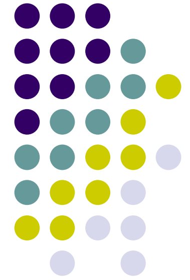
- La visita di tutto l'albero (prima chiamata sulla radice) richiede tempo O(numero nodi dell'albero)

Il TDA Tree



- Il TDA Position è usato come astrazione di nodo
- Il TDA Tree memorizza ciascun elemento in una posizione
- Position fornisce il metodo
  - element() che restituisce l'elemento memorizzato in quella posizione

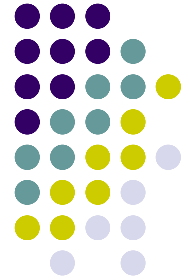
Il TDA Tree



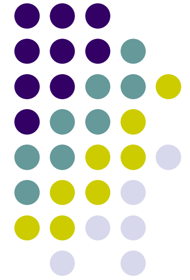
Metodi generici:

- size()
- isEmpty()
- iterator()
  - restiruisce un iteratore degli elementi memorizzati nei nodi dell'albero
- positions()
  - restituisce una collezione iterabile delle posizioni degli elementi dell'albero

Il TDA Tree



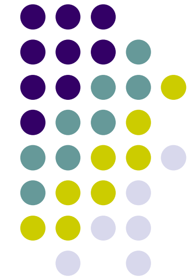
- Metodi di accesso:
  - root()
    - Restituisce la posizione della radice
  - parent(p)
    - Restituisce la posizione del padre del nodo in posizione p
  - children(p)
    - Restituisce una collezione iterabile delle posizioni dei figli del nodo in posizione p.



Il TDA Tree

- Metodi di interrogazione:
  - isInternal(p)
  - isExternal(p)
  - isRoot(p)
- Metodo di aggiornamento:
  - replace (p, e)
    - Rimpiazza l'elemento in posizione p con e, restituendo in output il vecchio elemento
- Ulteriori metodi di **aggiornamento** possono essere aggiunti nelle classi che implementano il TDA Tree

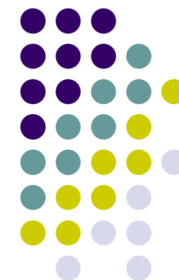
Interfaccia di Tree in Java



```
public interface Tree<E> extends Iterable<E> {
public int size();
public boolean isEmpty();
public Iterable<Position<E>> positions();
public E replace(Position<E> v, E e) throws
InvalidPositionException;
public Position<E> root() throws EmptyTreeException;
```

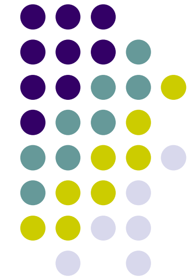
Continua nella prossima slide

Interfaccia di Tree in Java



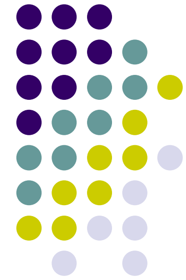
```
/*lancia BoundaryViolationException se v è la radice*/
public Position<E> parent(Position<E> v)  throws
InvalidPositionException,BoundaryViolationException;



/*lancia InvalidPositionException anche se v è una foglia*/
public Iterable<Position<E>> children(Position<E> v)
                                throws InvalidPositionException;
```

Interfaccia di Tree in Java

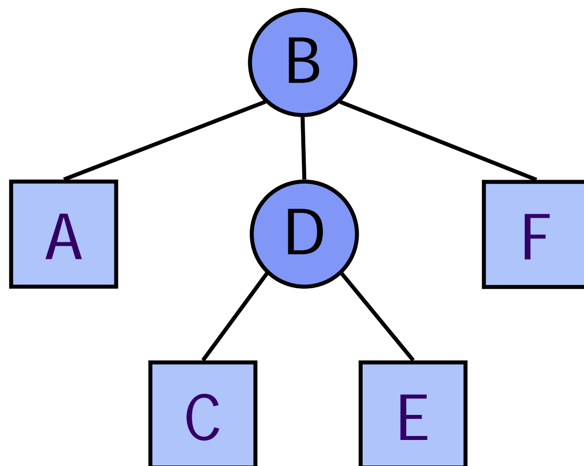


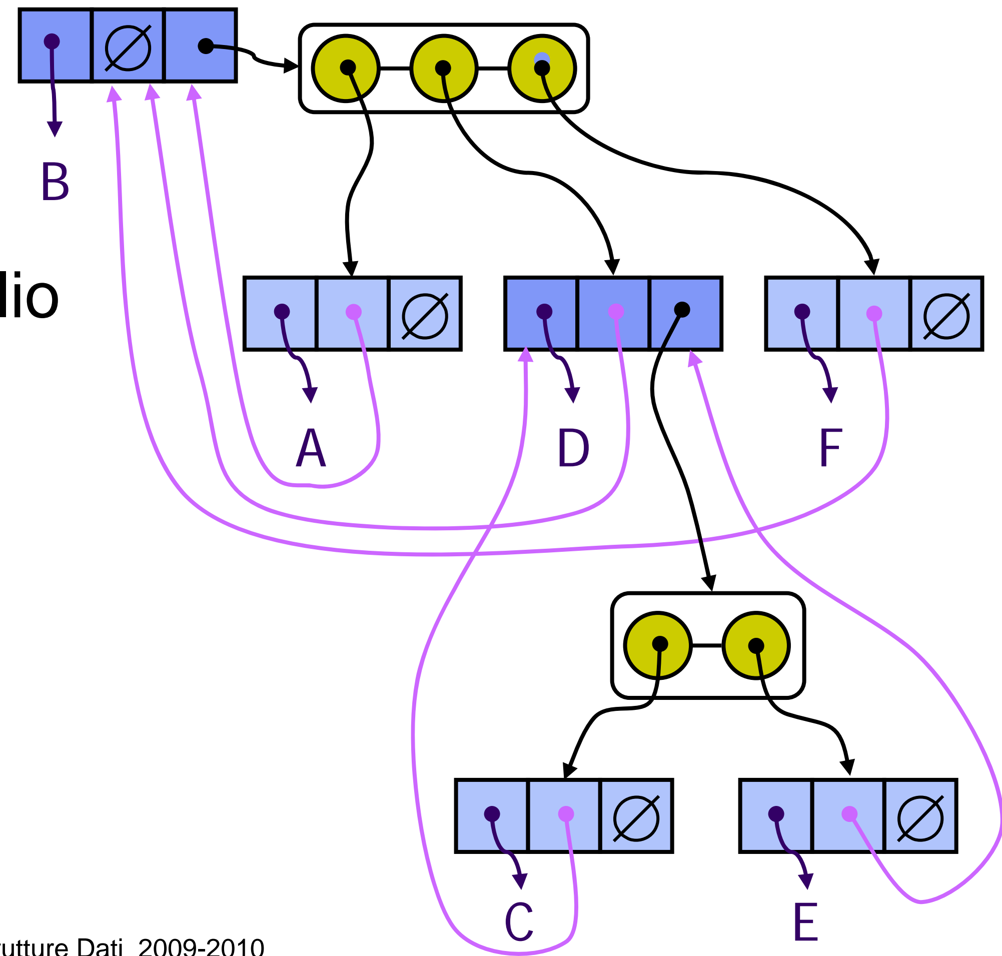
```
public boolean isInternal(Position<E> v)
                          throws InvalidPositionException;
public boolean isExternal(Position<E> v)
                          throws InvalidPositionException;
public boolean isRoot(Position<E> v)
                          throws InvalidPositionException;
}
```

Implementazione del TDA Tree

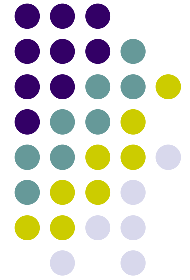


- Il TDA Position è implementato dalla classe TreeNode che contiene riferimenti a:
  - Elemento
  - Nodo genitore
  - Collezione dei nodi figlio



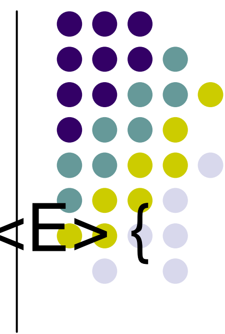


Interfaccia TreePosition



```
public interface TreePosition<E> extends Position<E>
   {
  // eredita element()
  public void setElement(E o);
  public PositionList<Position<E>> getChildren();
  public void setChildren(PositionList<Position<E>> c);
  public TreePosition<E> getParent();
  public void setParent(TreePosition<E> v);
}
```

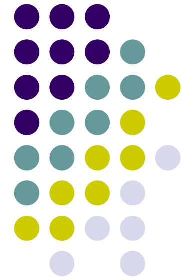
Classe TreeNode



```
public class TreeNode<E> implements TreePosition<E> {
 private E element;
 private TreePosition<E> parent;
 private PositionList<Position<E>> children;
 public TreeNode() { }
 public TreeNode(E element, TreePosition<E> parent,
        PositionList<Position<E>> children) {
   setElement(element);
   setParent(parent);
   setChildren(children);
 }
```

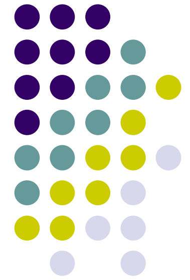
Continua nella slide successiva

Classe TreeNode



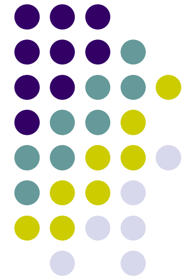
```
public E element() { return element; }
public void setElement(E o) { element = o; }
public PositionList<Position<E>> getChildren()  {
   return children;
 }
public void setChildren(PositionList<Position<E>> c) {
   children=c;
 }
public TreePosition<E> getParent() { return parent; }
public void setParent(TreePosition<E> v) {
    parent=v;
 }
}
```

La classe LinkedTree: il metodo checkPositions



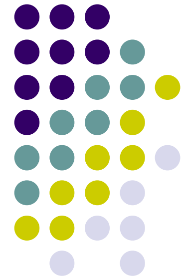
- checkPositions(Position<E> v)
  - Lancia InvalidPositionException se
    - v è null
    - v non è di tipo TreePosition

La classe LinkedTree: il metodo positions()



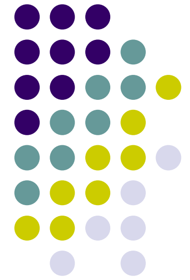
- Restituisce una collezione iterabile contenente i nodi dell'albero nell'ordine in cui sono visitati durante una visita preorder
  - La visita richiede tempo O(n)

Il metodo invocato da positions()



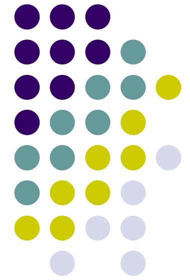
```
protected void preorderPositions(Position<E> v,
   PositionList<Position<E>> pos) throws InvalidPositionException {
  pos.addLast(v);
  if (isExternal(v)) return;
 //restituisce un iteratore delle position dei figli di v
  Iterator <Position <E>> itC = children(v).iterator();
  while itC.hasNext()
   preorderPositions(itC.next(),pos);
}
```

La classe LinkedTree: il metodo positions()



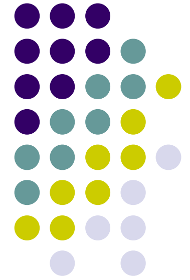
```
public Iterable<Position<E>> positions() {
PositionList<Position<E>> positions = new NodePositionList<Position<E>>();
 if(size != 0)
   preorderPositions(root(), positions);
return positions;
}
```

La classe LinkedTree: il metodo iterator()



```
public Iterator<E> iterator() {
 Iterable<Position<E>> positions = positions();//restituisce
                               //una collezione iterabile dei nodi
  PositionList<E> elements = new NodePositionList<E>();

 for(Position<E> pos: positions)
      elements.addLast(pos.element());
  return elements.iterator();  // restituisce un iteratore di elementi
 }
```

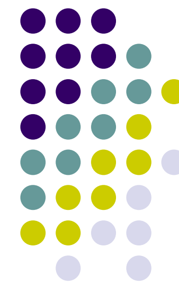
Esercizio



- Scrivere la classe LinkedTree che implementa il TDA Tree con TreeNode e scrivere un programma che testi la classe.

```
public class LinkedTree <E>implements Tree<E> {
        protected TreePosition<E> root;
        protected int numElem;
                ................
    }
```

Esercizi

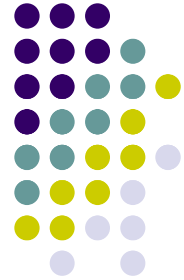


- Testare l'implementazione di **LinkedTree** scrivendo un programma **TestLinkedTree** che utilizzi tutti i metodi dell'interfaccia **Tree**

- Aggiungere a **LinkedTree** il metodo depth(v) che restituisce la profondità del nodo v

- Aggiungere a **LinkedTree** il metodo height() che restituisce l'altezza dell'albero

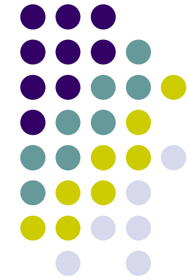
Esercizi



Aggiungere a **LinkedTree** i metodi

- **public** Position <E> addChild (E element,Position<E> v)
  - Che aggiunge un figlio alla fine della collezione di figli di v
- **public** Position<E> removeChild(Position<E> v) **throws** InvalidPositionException
  - Che rimuove il primo figlio di v
- **public** Position<E>addRoot(E e)**throws** NonEmptyTreeException

Esercizi



- Scrivere l'algoritmo ricorsivo che prende in input un albero di interi e computa la somma degli elementi dell'albero