Laurea Triennale in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DEPARTMENT OF COMPUTER, CONTROL, AND MANAGEMENT ENGINEERING ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Alberi

Riferimento: Problem solving with Algorithms and Data Structures using Python

Introduzione

Alberi

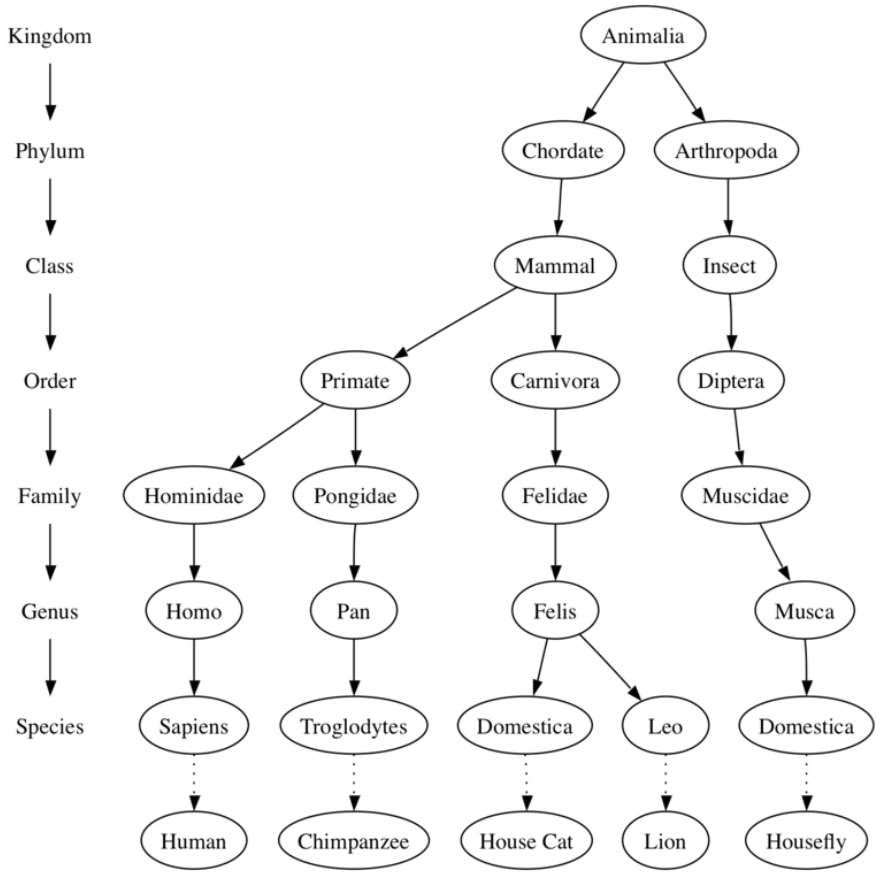
- Struttura dati molto comune con tantissime applicazioni in informatica
  - Sistemi operativi (e.g. struttura delle cartelle)
  - Basi di dati
  - Reti
  - …

- Tipicamente rappresentati con la radice in alto e le foglie verso il basso

Esempi/Proprietà

- struttura gerarchica composta da nodi ed archi
- categorie più generali in alto / più specifiche in basso
- ogni nodo, tranne le foglie, hanno uno o più figli
- i figli appartengono alla categoria più generale del padre

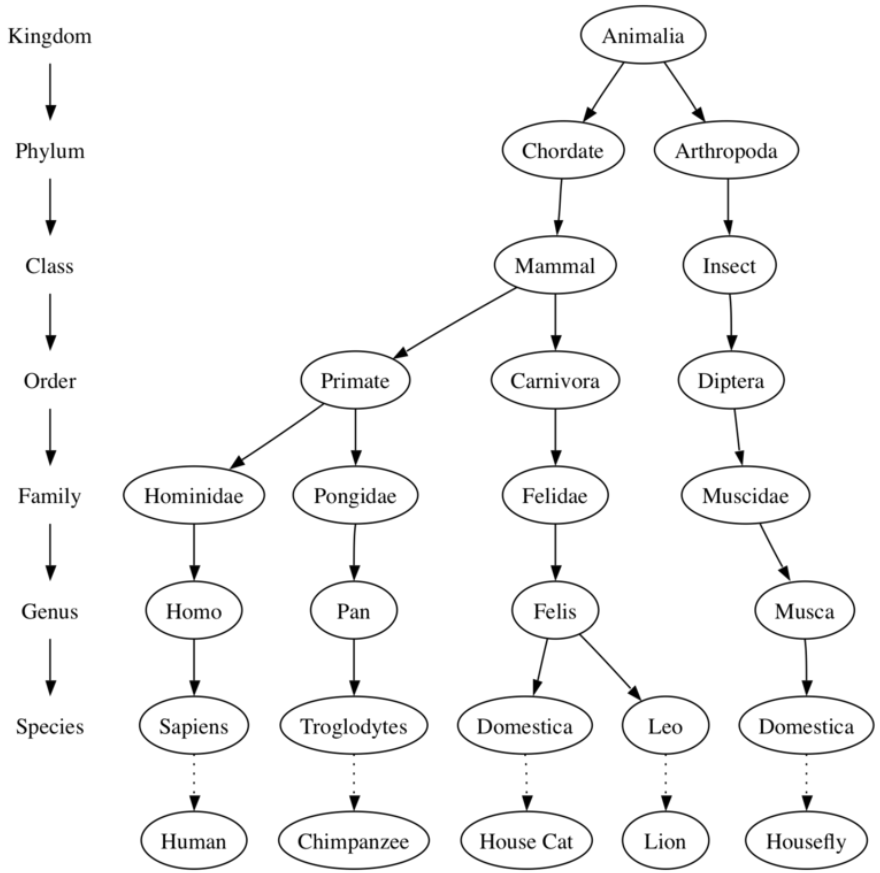
Esempio: Biologia



Esempi/Proprietà

- **i nodi che appartengono a padri diversi sono indipendenti fra di loro**
- **i nodi-foglie sono univoci**
- **i percorsi dalla radice alle fogli trascorrono una serie di nodi/categorie che caratterizzano le foglie**

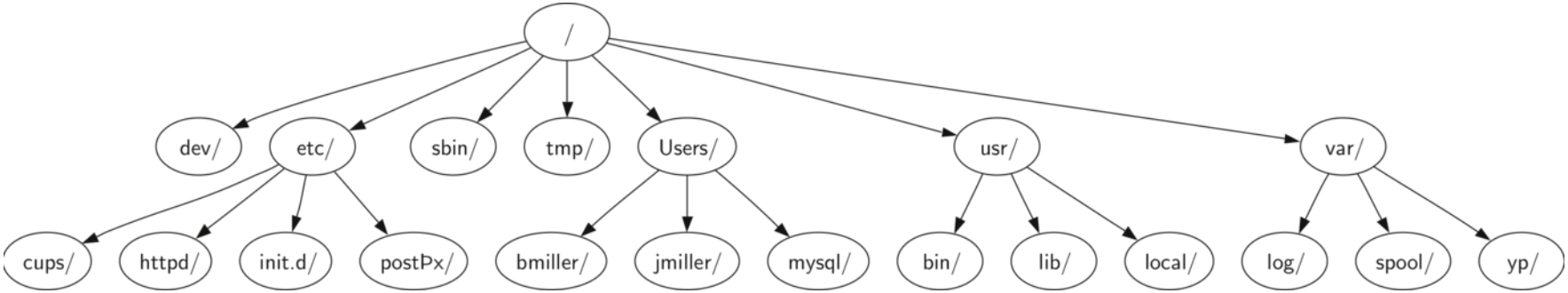
Esempio: Biologia



Esempi/Proprietà

- **ogni nodo (tranne le foglie) può essere considerato la radice di un sottoalbero (subtree)**
- **si può spostare un intero sottoalbero in un'altra posizione senza dover modificare tutti i livelli successivi (**per esempio commando `mv`**)**

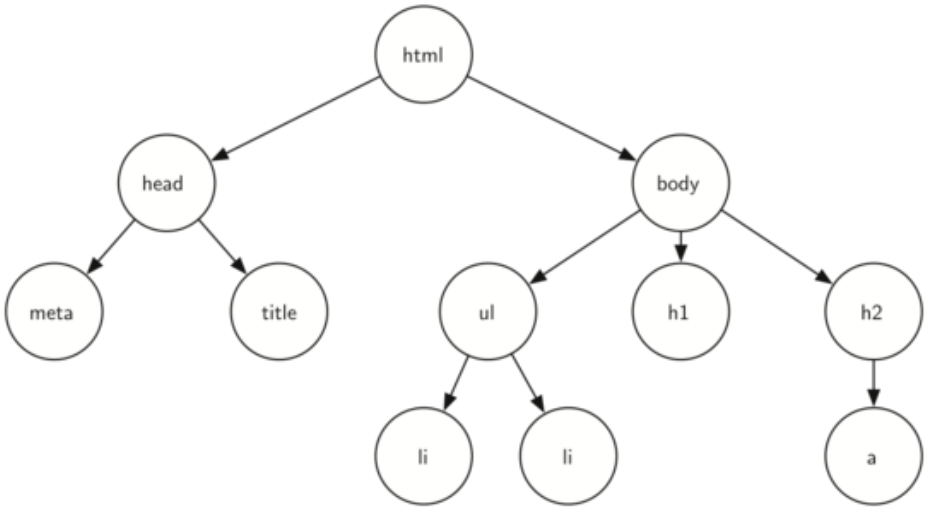
Albero delle cartelle in Unix



Esempi/Proprietà

- **HTML: contiene il head ed il body**
- **Head: contiene il titolo e i meta**
- **Body: contiene vari elementi che compongono la pagina web**

Pagina web in HTML



Definizioni/Terminologia (1/3)

Nodi (nodes)

- possono avere un nome «chiave»
- tipicamente contengono dati associati

Archi (edges)

- collegano due nodi e rappresentano una relazione fra di loro
- tutti i nodi tranne la radice hanno esattamente un arco in ingresso
- ogni nodo, tranne le foglie, può avere più archi in uscita

Definizioni/Terminologia (2/3)

Radice (root)

- l'unico nodo al livello più basso (zero) dell'albero
- non ha archi in ingresso

Figli (children)

- tutti i nodi che hanno un arco in ingresso dal nodo n sono chiamati figli del n

Padre (parent)

- un nodo è il padre di tutti i nodi con cui è collegato con archi in uscita

Fratelli (siblings)

- tutti i figli dello stesso nodo si chiamano fratelli

Definizioni/Terminologia (3/3)

Sottoalbero (subtree)

- l'insieme dei nodi e degli archi che contiene un padre (radice del sottoalbero) e tutti i suoi discendenti

Foglia (leaf)

- un nodo che non ha figli

Livello (level)

- corrisponde al numero di archi che contiene un percorso dalla radice fino al nodo in questione

Altezza (height)

- Il livello massimo fra tutti i nodi dell'albero

Definizione di un albero

Un albero consiste di un insieme di nodi ed un insieme di archi che collegano coppie di nodi.

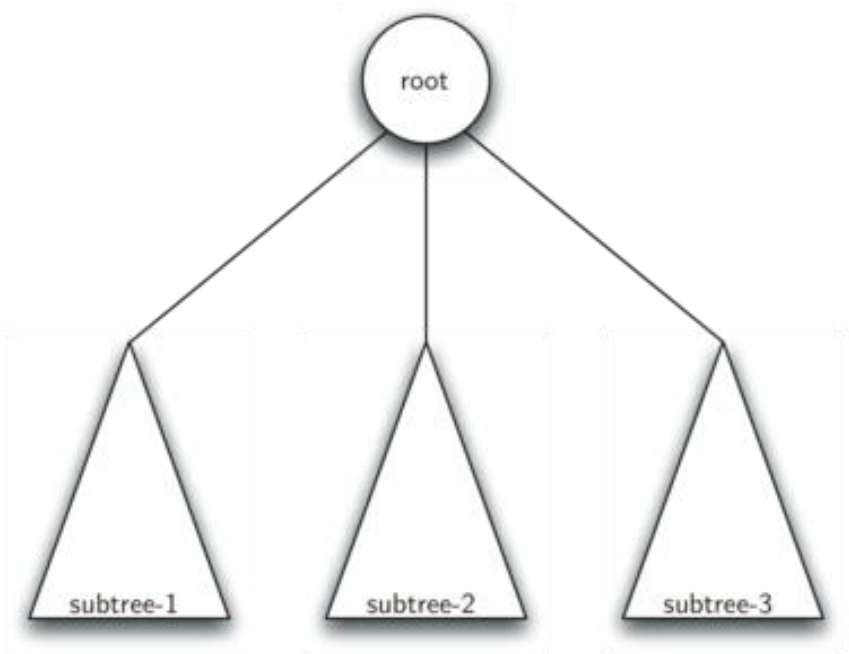
Un albero ha le seguenti proprietà:

- un nodo corrisponde alla radice dell'albero
- ogni nodo n, tranne la radice, è collegato con un arco da esattamente un nodo p, dove p e il padre di n
- c'è un unico percorso che collega un nodo con la radice del albero
- se ogni nodo ha al massimo due figli l'albero viene chiamato binario

Definizione di un albero (ricorsiva)

Un albero o è vuoto oppure è costituito da una radice e zero o più sottoalberi, ognuno dei quali è un albero.

La radice di un sottoalbero è collegata alla radice dell'albero-padre con un arco

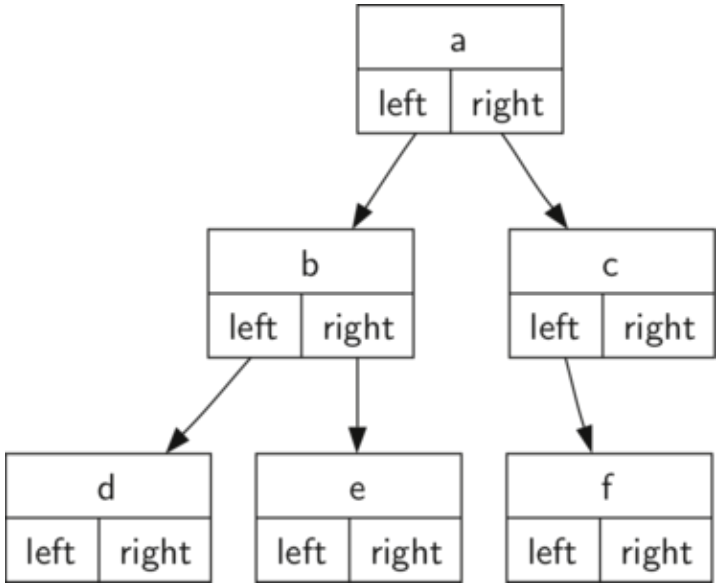


Albero binario – Implementazione (1/3)

Un modo di rappresentare un albero è usando nodi e riferimenti.

Per un albero binario definiamo la classe BinaryTree:

```
class BinaryTree:
    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None
```



Albero binario – Implementazione (2/3)

L'albero costruito ha solo un nodo (radice)

Per costruire l'albero servono due metodi per inserire il figlio sinistro e il figlio destro:

```
def insertLeft(self,newNode): # Analogamente per Right
    if self.leftChild == None:
        self.leftChild = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.leftChild = self.leftChild
        self.leftChild = t
```

- se non c'è un figlio il nodo viene semplicemente aggiunto
- se c'è già un figlio assegnato il nuovo nodo viene aggiunto e quello già esistente viene spostato come figlio del nuovo nodo

Albero binario – Implementazione (3/3)

Metodi per accedere ai figli e alla radice del albero:

```
def getRightChild(self):
    return self.rightChild

def getLeftChild(self):
    return self.leftChild

def setRootVal(self,obj):
    self.key = obj

def getRootVal(self):
    return self.key
```

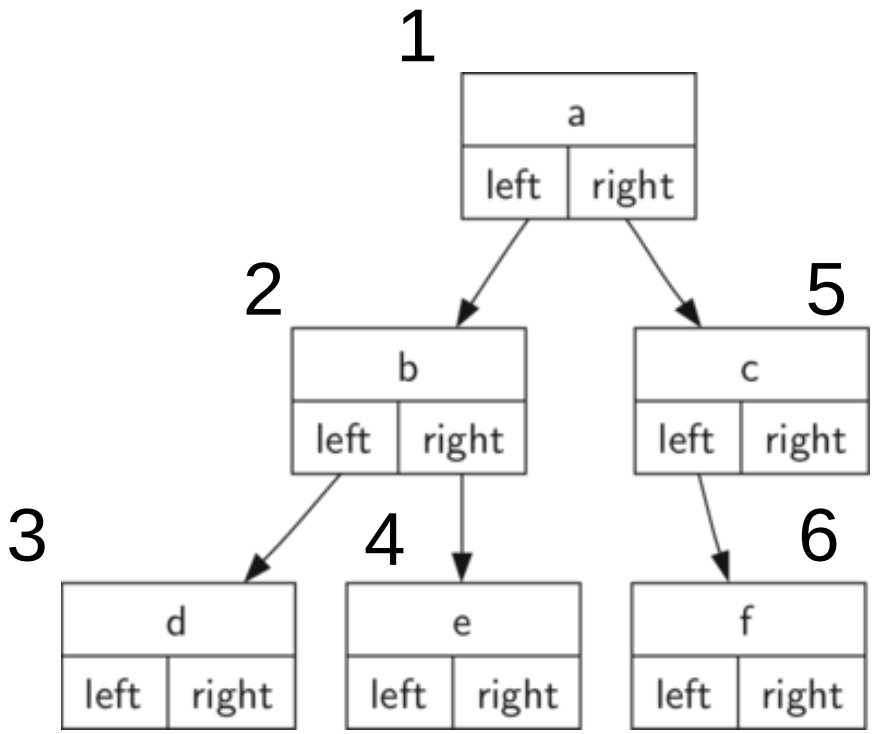
Metodi di visita

Preordine:

```
def preordine(tree):
    if tree == None:
        return
    print(tree.getRootVal())
    preordine(tree.getLeftChild())
    preordine(tree.getRightChild())
```

Per l'albero a destra:

a,b,d,e,c,f



In generale, invece di stampare il valore si fa qualcosa di più utile

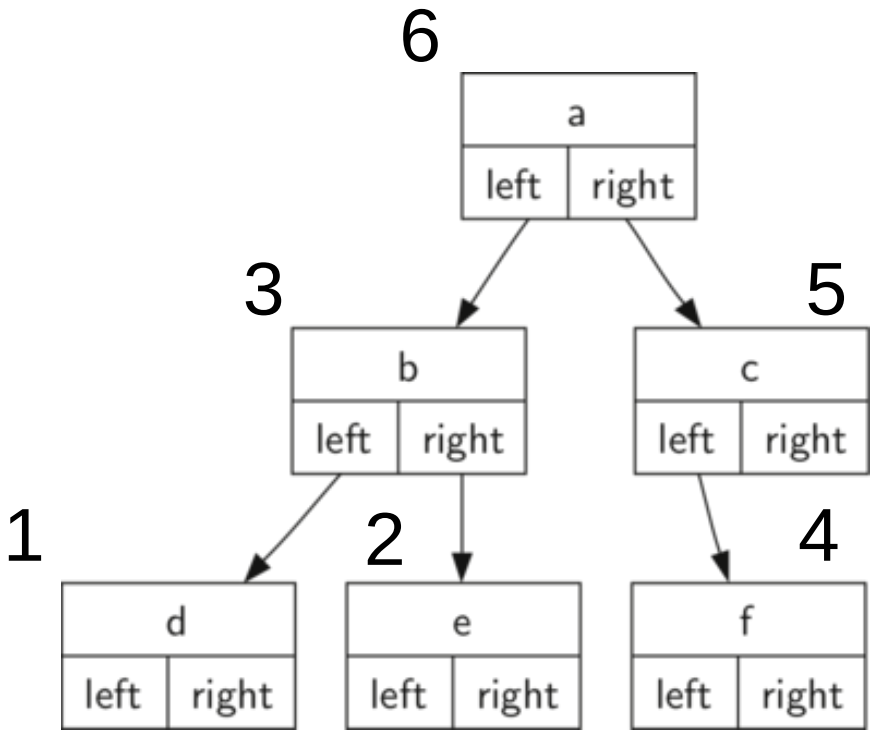
Metodi di visita

Postordine:

```
def postordine(tree):
    if tree == None:
        return
    postordine(tree.getLeftChild())
    postordine(tree.getRightChild())
    print(tree.getRootVal())
```

Per l'albero a destra:

d,e,b,f,c,a



In generale, invece di stampare il valore si fa qualcosa di più utile

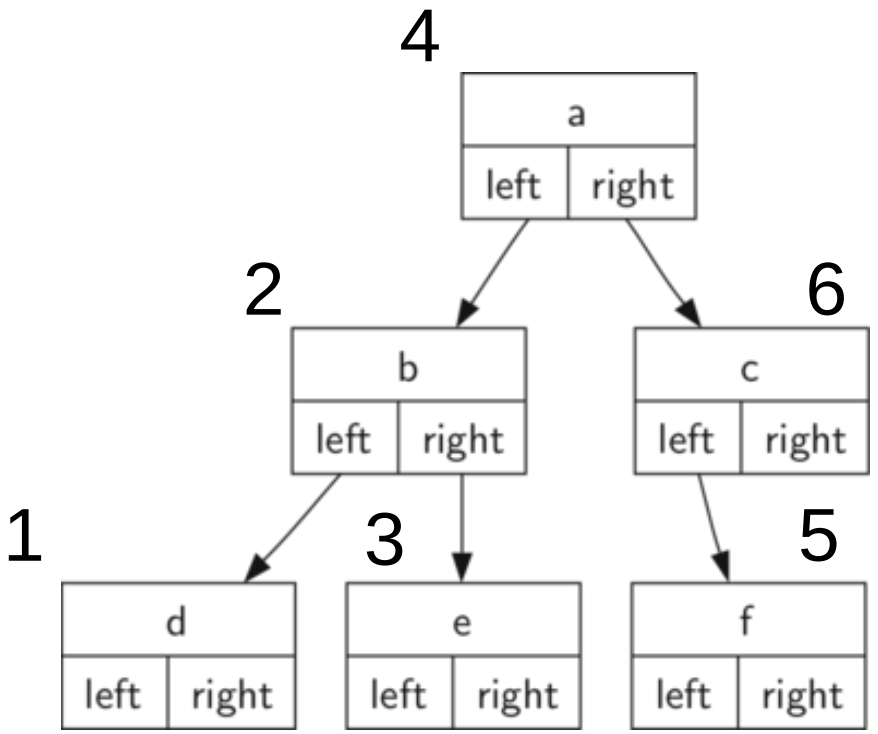
Metodi di visita

Simmetrica:

```
def simmetrica(tree):
    if tree == None:
        return
    simmetrica(tree.getLeftChild())
    print(tree.getRootVal())
    simmetrica(tree.getRightChild())
```

Per l'albero a destra:

d,b,e,a,f,c



In generale, invece di stampare il valore si fa qualcosa di più utile