# Alberi Binari di Ricerca e Dizionari

Laboratorio di Programmazione II
Corso di Laurea in Bioinformatica
Dipartimento di Informatica - Università di Verona

# Sommario





- Alberi binari di ricerca (BST)
- Implementazione BST
- Dizionari ed Hash Table
- Hash Table in Java

# Alberi Binari

# Alberi Binari



## Concetti di base

- Ogni nodo puo' avere al piu' due figli
- Informazioni del nodo:
  - Elemento (i.e., Chiave)
  - figlio destro
  - figlio sinistro
- BinaryNode.java

# Alberi binari di ricerca



## Concetti base

- Albero binario che soddisfa la seguente proprieta'
- Per ogni nodo $n$ sia $k$ chiave di $n$:
  - tutte le chiavi contenute nel suo sottoalbero sx sono $\leq$ di $k$
  - tutte le chiavi contenute nel suo sottoalbero dx sono $\geq$ di $k$

# Operazioni principali su BST



## Interfaccia BST

- Visite
  - pre-ordine, post-ordine, ampiezza
  - in-ordine: le chiavi risultano ordinati
- get: cerca una chiave nell'albero, sfrutta la definizione di BST per efficienza
- put: inserisce una chiave nell'albero mantenendo le proprieta' del BST
- remove: elimina una chiave dall'albero mantenendo le proprieta' del BST

# BST



## Implementazione

- Strutture collegate per albero binario
- Utilizzo interfaccia Comparable
- Interfaccia: BST.java
- Realizzazione: SimpleBST.java

# Ricerca in un BST



## Ricerca di chiavi in un BST

- Ricerca sfrutta l'ordine naturale delle chiavi
- Dato un nodo $n$ (inizialmente n = radice) ed una chiave da cercare *item*
  - se $n.chiave$ uguale a *item*, restituisco $n$
  - se $item < n.chiave$ cerco nel sottoalbero sinistro
  - se $item > n.chiave$ cerco nel sottoalbero destro
- Ipotesi: non ho mai due chiavi uguali nell'albero
- Ipotesi resa vera dalla procedura di inserimento

# Inserimento



## Inserimento chiavi in un BST

- metodo ricorsivo che attraversa l'albero fino a trovare la posizione corretta in cui inserire il nuovo nodo
- posizione corretta soddisfa le proprieta' del BST
- se la chiave e' gia' presente nell'albero non inserisco un nuovo nodo (restituisco quello gia' presente)
- metodo put(..) classe SimpleBST.java

# Cancellazione



## Cancellare chiavi da un BST

- Cancello il nodo con la chiave che voglio eliminare
- Debbo aggiornare i nodi in modo tale che l'albero sia ancora un BST
- Metodo remove(...) classe SimpleBST.java

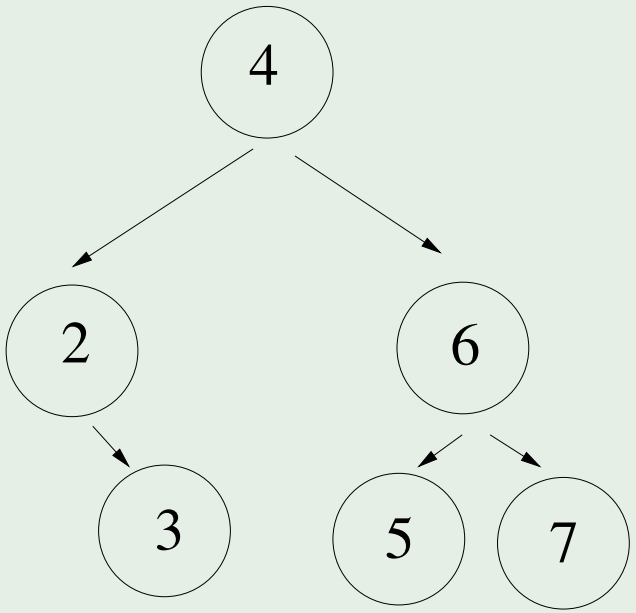
# Stampa albero



## Rappresentazione parentetica

- Rappresentazione intuitiva e semplice per alberi
- ( radice ( sottoalbero sx ) ( sottoalbero dx ) )

## Example (Rappresentazione parentetica)




( 4 ( 2 ( ) ( 3 ( ) ( ) ) ) ( 6 ( 5 ( ) ( ) ) ( 7 ( ) ( ) ) ) )

# Esercizi



## Esercizi su BST

- Implementare il metodo get(...) della classe SimpleBST.java
- Implementare il metodo ricorsivo recToString(...) della classe SimpleBST.java
- Implementare il metodo riempiLista(...) della classe SimpleBST.java
- Implementare il metodo parse(...) della classe SimpleBST.java
  - ipotizziamo di avere in input solo interi
  - utilizzare la classe StringTokenizer per dividere la stringa

# Dizionari e Hash Table

# Dizionari

## ADT Dizionario

- In generale un ADT che supporta le seguenti operazioni e' detto Dizionario
  1. membership (get o search)
  2. put
  3. remove

# Dizionari



## Il concetto di chiave

- Dizionari: insieme di coppie <elemento,chiave>
- Generalizzazione di Array: indicizziamo elementi con oggetti
- Ordine totale sulle chiavi
- Chiave univoca in un dizionario
- Operazioni:
  - put(key, element)
  - remove(key)
  - get(key)

# Implementazione dei Dizionari



## Possibili implementazioni

- BST sono una implementazione di un dizionario
- Basta aggiungere al nodo un campo elemento distinto dal campo chiave
- HashTable altra implementazione

# Hash Table in Java

# L'interfaccia Map

## L'interfaccia Map

- Permette di indicizzare oggetti (elementi) utilizzando altri oggetti (chiavi)
- Metodi fondamentali:
  - put(Object key, Object value)
  - get(Object key)
  - remove(Object key)
  - containsKey(), containsValue()
- Vedere Map nelle API java

# Implementazioni di Map



## Implementazioni di Map

- TreeMap:
  - implementazione basata su BST
  - red black tree: caso speciale di BST
- HashMap
  - implementazione basata su hashing

# Creazione ed utilizzo di HashMap



## Example (utilizzare HashMap)

```
HashMap<String, Integer> profilo =
new HashMap<String, Integer>();

profilo.put("A", 3);
profilo.put("C", 0);
profilo.put("G", 0);
profilo.put("T", 1);

System.out.println(profilo);
System.out.println("valore di A = "+profilo.get("A"));
```

# Scandire una HashMap



## Scandire una HashMap

- Posso accedere all'insieme delle chiavi con il metodo keySet();
- Scandire il vettore con gli iteratori
- Accedere agli elementi con il metodo get
- Oppure accedere alla collezione dei valori: metodo values()

# Scansione di HashMap



## Example (utilizzo keySet())

```
int sum = 0;
Set<String> keySet = profilo.keySet();
for (Iterator<String> iterator = keySet.iterator();
iterator.hasNext();) {

String nucleotide = iterator.next();
sum+=profilo.get(nucleotide);

}
System.out.println("somma = "+sum);
```

# Scansione di HashMap



## Example (utilizzo values())

```
sum = 0;
Collection<Integer> valueSet = profilo.values();
for (Iterator<Integer> iterator = valueSet.iterator();
 iterator.hasNext();) {

Integer integer = iterator.next();
sum+=integer;

}
System.out.println("somma = "+sum);
```

# Indicizzazione tramite oggetti



## il metodo hashCode()

- hashCode() e' un metodo di object che ha una sua implementazione di default
- Basata sulla locazione di memoria dell'oggetto(vedere API per dettagli)
- Posso ridefinire il metodo hashCode() come con qualsiasi altro metodo
- Overriding dell'hashCode() necessario se ho ridefinito equals()
  - Se due oggetti sono uguali debbono avere lo stesso codice di hash

# Esempio problemi hashCode()



## Example (problemi hashCode)

```
HashMap<Studente, List<Integer>> map =
new HashMap<Studente, List<Integer>>();

map.put(new Studente("001"),randomEsami(5));
map.put(new Studente("002"),randomEsami(2));
map.put(new Studente("003"),randomEsami(3));
System.out.println(map);

Studente s = new Studente("001");
List<Integer> listaEsami = map.get(s);
System.out.println(listaEsami);
```

# Overriding di hashCode()



## overriding di hashCode()

- hashCode() deve essere consistente con il metodo equals()
- esempio: Studente.java abbiamo fatto overriding di equals() ma non di hashCode()
- consistente deve essere basato sugli stessi campi

# Esempio overriding di hashCode()



## Example (overriding di hashCode)

In Studente.java

```
...
public int hashCode() {
return matricola.hashCode();
}
...
```

# Fare overriding di hashCode()



## fare overriding di hashCode()

- Utilizzare il metodo hashCode() di Object per i singoli campi
- Se ho tipi primitivi, creo l'oggetto corrispondente e chiamo hashCode()
- Quando ho piu' elementi posso semplicemente sommare gli hashCode()
- Questo risulta in un comportamento corretto: istanze uguali avranno lo stesso hashCode()
- Posso avere metodi per calcolare l'hashCode() che risultano in comportamenti piu' efficienti per le operazioni della Map
- approfondimento B. Eckel Thinking in java (disponibile on-line)

# Esercizi I



## Esercizi su Hash Table I

- implementare il metodo calcolaMediaStudenti(...) della classe ExampleHashCode.java[a]
- implementare il metodo migliorStudente(...) della classe ExampleHashCode.java
- modificare la classe CoordinatePolari.java in maniera tale da avere un corretto funzionamento del metodo main della classe TestHashCode.java [soluzione in CoordinatePolari.sol2]

[a] Scaricare il file Nucleotide.java per poter compilare ExampleHashCode.java

# Esercizi II



## Esercizi su Hash Table II

- implementare il metodo compareTo(...) della classe SongManager.java
- implementare il metodo hashCode(...) della classe SongManager.java[a]
- implementare il metodo bestSong(...) della classe SongManager.java

[a] Nota: il metodo non è presente e va quindi dichiarato ed implementato.