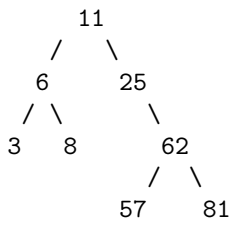
Programmazione ad Oggetti
Mod. 2

1/6/2023

Studente _____ Matricola _____

1. Vogliamo realizzare in Java 8+ una classe `BST`, parametrica su un tipo generico `T`, che rappresenta alberi binari di ricerca (Binary Search Tree) i cui nodi sono decorati con valori di tipo `T`. Un albero binario di ricerca è un normale albero binario, tuttavia gli elementi al suo interno vengono mantenuti con un certo ordine dato da una peculiare proprietà ricorsiva. Si prenda in esame il seguente esempio che per semplicità utilizza degli interi:



Il nodo radice 11 ha un valore superiore a tutti i nodi del sotto-albero sinistro ed inferiore a tutti i nodi del sotto-albero destro. E questo è vero non solo prendendo in esame la radice dell'albero, ma prendendo qualsiasi nodo in qualunque punto dell'albero!

Quando si inserisce un nuovo nodo nell'albero è necessario metterlo nel punto giusto, rispettando questa proprietà, così che l'albero sia sempre in uno stato valido.

Segue l'implementazione da completare della classe `BST`.

```
public class BST<T> implements Iterable<T> {
    protected final Comparator<? super T> cmp;
    protected Node root;

    protected class Node {
        protected final T data;
        protected Node left, right;

        protected Node(T data, Node left, Node right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    public BST(Comparator<? super T> cmp) {
        this.cmp = cmp;
    }

    public void insert(T x) {
        root = insertRec(root, x);
    }
```

```
    protected Node insertRec(Node n, T x) { /* da implementare */ }

    protected void dfsInOrder(Node n, Collection<T> out) { /* da implementare */ }

    @Override
    public Iterator<T> iterator() { /* da implementare */ }

    public T min() { /* da implementare */ }
    public T max() { /* da implementare */ }
}
```

(a) [7 punti] Si implementi ricorsivamente il metodo `insertRec()` badando a rispettare la proprietà degli alberi binari di ricerca enunciata sopra. Per determinare se discendere nel sotto-ramo sinistro oppure in quello destro è necessario utilizzare il `Comparator` passato in costruzione per confrontare l'argomento `x` con il campo `data` del nodo `n`.

(b) Gli alberi devono essere *iterabili* e l'iteratore deve attraversare l'albero in DFS (*Depth-First Search*), fornendo gli elementi di tipo `T` secondo il criterio denominato *in-order*, che consiste nel visitare prima il sotto-ramo sinistro, poi il nodo, poi il sotto-ramo destro.

i. [4 punti] Si implementi il metodo `dfsInOrder()` facendo in modo che i dati contenuti nei nodi visitati in *in-order* vengano aggiunti alla collection passata come secondo argomento.

ii. [2 punti] Si implementi il metodo `iterator()` nella maniera più semplice possibile, ovvero sfruttando il metodo `dfsInOrder()`.

(c) [4 punti] Si implementino i metodi `min()` e `max()` facendo attenzione alla complessità computazionale. Essendo l'albero sempre ordinato è facile trovare il minimo ed il massimo in $O(\log_2(n))$, dove n è il numero totale di nodi dell'albero.

(d) [2 punti (bonus)] Si definisca una sottoclasse di `BST` parametrica su un generic `T` vincolato ad essere sottotipo di `Comparable<? super T>`. Tale sottoclasse sposta la logica di confronto sul type parameter. Si implementi solamente un costruttore senza argomenti che chiama il super-costruttore passando una opportuna lambda come argomento di tipo `Comparator`.

2. Vogliamo implementare in C++ una classe `pair` templatizzata su due tipi generici `A` e `B` che rappresentano i tipi del primo e del secondo elemento della coppia. Si utilizzi la versione del linguaggio che si preferisce, è sufficiente dichiarare quale.

(a) [4 punti] La classe `pair` deve rispettare gli stili della generic programming e comportarsi come un valore: si definiscano gli opportuni costruttori, tra cui quello per copia, e l'operatore di assegnamento.

(b) [9 punti] Si implementino tutti i metodi e gli operatori ritenuti necessari o interessanti.

(c) [2 punti (bonus)] Si implementi un costruttore per copia templatizzato su due tipi generici `C` e `D` che è in grado di costruire una `pair<A,B>` dato un argomento di tipo `const pair<C, D>&`.

Question:	1	2	Total
Points:	17	13	30
Bonus Points:	2	2	4
Score:			