Alberi binari

Struttura vuota
Un elemento detto radice (in questo contesto gli elementi chiamati nodi) a cui sono collegati due sottoalberi (sottoalbero di sinistra e sottoalbero di destra)

Visto che distinguiamo i due sottoalberi l'albero si dice ordinato.

```
class Nodo{
	int info;
	Nodo sinistra;
	Nodo destra;
	public Nodo(int i, Nodo s, Nodo d){
		info = i;
		sinistra = s;
		destra = d;
	}

class AlberoBinario{
	Nodo radice=null;

	void inserisci(Nodo n){}
	Nodo estrai(int chiave){return null;}

	/*
	Alcuni esempi di visita
	*/
```

```
/*
Dimensione del problema:
        numero dei nodi

        caso terminale: numero dei nodi = 0, cioè l'albero è vuoto
        caso generale: ottengo la soluzione supponendo di saper risolvere il problema su alberi più piccoli cioè il sottoalbero di sinistra ed il sottoalbero di destra

    N.B. quando scriviamo un metodo ricorsivo dobbiamo prevedere un parametro che indica la dimensione del sottoproblema che stiamo affrontando in questo momento; nel nostro caso ci serva un parametro che indica la radice del sottoalbero su cui stiamo lavorando e quindi un parametro di classe nodo
    */

    /**
    Visualizzazione con visita anticipata (preorder)
    */

    private void visualizzaAnticipata(Nodo radice){
        if (radice != null)
            {
            System.out.println(radice.info);
            visualizzaAnticipata (radice.sinistra);
            visualizzaAnticipata (radice.destra);
            }

    }
```

```
	/**
	Visualizzazione con visita simmetrica (inorder)
	*/

	private void visualizzaSimmetrica(Nodo radice){
		if (radice != null)
			{
			visualizzaSimmetrica (radice.sinistra);
			System.out.println(radice.info);
			visualizzaSimmetrica (radice.destra);
			}
	}

	/**
	Visualizzazione con visita posticipata (postorder)
	*/

	private void visualizzaPosticipata(Nodo radice){
		if (radice != null)
			{
			visualizza(radice.sinistra);
			visualizza(radice.destra);
			System.out.println(radice.info);
			}
	}
public void visualizzaAnticipata(){
	visualizzaAnticipata(radice);

	}
```

```
public void visualizzaSimmetrica(){
	visualizzaSimmetrica (radice);

	}

public void visualizzaPosticipata(){
	visualizzaPosticipata(radice);

	}

private int conta (Nodo radice){
	if (radice==null)
		return 0;
	else
		return conta(radice.sinistra)+ conta(radice.destra)+1;
	}

public int conta (){
	return conta (radice);
}

private int profondita (Nodo radice){
	if (radice==null)
		return 0;
	else
		{
		int ps = profondita(radice.sinistra);
		int pd = profondita(radice.destra);
		if (ps < pd)
			return pd+1;
		else
```

```
            return ps+1;

    }

public int profondita (){
    return profondita (radice);

}
```

Albero binario di ricerca

È un albero binario in cui i valori dei nodi dei sottoalbero di sinistra sono minori della loro radice, mentre i valori dei nodi dei sottoalberi di destra sono maggiori delle loro radici

```
class AlberoBinarioRicerca extends AlberoBinario {

    private boolean ricerca (int chiave, Nodo radice){
        if (radice == null)
            return false;
        else
            if (radice.info == chiave)
                return true;
            else
                if (radice.info < chiave)
                    return ricerca(chiave,radice.destra);
                else
                    return ricerca(chiave,radice.sinistra);
    }
```

```
	public boolean ricerca (int chiave){
		return ricerca (chiave, radice);
	}

public boolean ricerca_it (int chiave){
	Nodo r = radice;
	while ( r != null && r.info != chiave) {
		if (r.info < chiave)
			r = r.destra;
		else
			r = r.sinistra;
	}
/*
if (r ==null)
	return false;
else
	return true;
*/
	return r != null;
}

/**
Metodo per l'inserimento di un elemento
*/

/*
In C (sfruttiamo il passaggio dei parametri per riferimento)
```

```
 void inserisci(int el; Nodo &radice){
      if (radice==null)
            {
            radice = (Nodo) malloc(sizeof(Nodo));
            radice.info=el;
            radice.sinistra=null;
            radice.destra=null;
            }
      else if (el<radice.info)
            inserisci(el, radice.sinistra);
            else
                  inserisci(el, radice.destra);
}
*/

private void inserisci(int el, Nodo radice){
      if (el < radice.info)
            if (radice.sinistra == null)
                  radice.sinistra = new Nodo(el,null, null);
            else
                  inserisci(el, radice.sinistra);
      else
            if (radice.destra == null)
                  radice.destra = new Nodo(el,null, null);
            else
                  inserisci(el, radice.destra);

}

public void inserisci(int el) {
```

```
        if (radice==null)
                radice= new Nodo(el,null,null);
        else
                inserisci(el,radice);
}


private void visualizzaStruttura(Nodo radice, int ind){
    if (radice!=null)
    {
      visualizzaStruttura(radice.sinistra,ind+5);
      for (int i = 0; i < ind; i++){
          System.out.print(" ");
      }
      System.out.println(radice.info);
      visualizzaStruttura(radice.destra,ind+5);
    }
}

 public void visualizzaStruttura(){
     visualizzaStruttura(radice,0);
 }




}
```