1) VERIFICARE SE I VALORI DELLE FOGLIE SONO TUTTI PARI

```
public boolean verificaPari(AlberoBinario a){
        if (a==null) return true;
        if (a.Destro()==null && a.Sinistro()==null
        && a.getVal()%2==1)
                return false;
        return verificaPari(a.Destro()) &&
verificaPari(a.Sinistro());
}
```

2) VERIFICARE SE L'ALBERO B E' CONTENUTO NELL'ALBERO A:

```
public boolean eContenuto(AlberoBinario
a,AlberoBinario B){
        if(a==null || b==null) return true;
        return appoggio(a,b.getVal()) &&
eContenuto(a,b.Sinistro()) &&
eContenuto(a,b.Destro());
}

public boolean appoggio(AlberoBinario a,int x){
        if(a==null) return false;
        return a.getVal()==x ||
appoggio(a.Sinistro(),x) ||    appoggio (a.Destro(),x);
}
```

complessità temp. migliore= theta(n)
complessità temp. peggiore= theta(n*m)

complessità spaz. migliore= theta(n)
complessità spaz. peggiore= theta(n*m)

Se gli alberi hanno dimensioni diverse,cioè se A ha n elementi e B ha m elementi, la complessità peggiore è theta(n*m). Se la dimensione è la stessa la complessità sarà theta(n^2)
Nel caso migliore il primo elemento di B non è contenuto in A

3) VERIFICARE SE L'ALBERO A E' NULLO OPPURE L'INTERO X CONTENUTO NELLA RADICE DELL'ALBERO A NON APPARE IN ALTRE POSIZIONI DELL'ALBERO

```
public boolean nonRipetuto(AlberoB a){
        if(a==null) return true;
                return appoggio(a.sinistro(),a.val())
|| (a.destro(),a.val());
}

public boolean appoggio(AlberoB a,int x){
        if(a==null) return false;
        if(a.destro()==null && a.sinistro==null)
                return !(a.val()==x) ||
appoggio(a.destro(),x) || appoggio(a.sinistro(),x);
}
```

4) VERIFICA SE LA FOGLIA CHE SI TROVA ALLA MINORE PROFONDITA' IN A APPARE AD UN LIVELLO MINORE DI L

```
public boolean pocoProfondo(AB a,int l){
        if(a==null) return true;
        if(a.destr()==null && a.sinistro()==null &&
<l) return true;
        if(a.destr()==null && a.sinistro()==null &&
>=l) return false;
        return pocoProfondo(a.sinistro(),l-1) &&
(a.destro(),l-1);
}
```

5) RESTITUISCE IL NUMERO DI NODI PRESENTI NELL'ALBERO A A LIVELLO L. SE IL LIVELLO L E' MINORE DI ZERO O PIU' IN GENERALE SE NON CI SONO NODI AL LIVELLO L NELL'ALBERO a, DEVE ESSERE RESTITUITO ZERO.

```
public static int contaPerLivello(AlberoBinario a,int
l){
        if(a==null || l<0) return 0;
            int cont=contaPerLivello(a.sinistro(),l-1)
        + contaPerLivello(a.destro(),l-1);
        if(l!=0 && a!=null)
        cont ++;
        return cont;
}
```

6) VERIFICA SE VI E' ALMENO UN NODO N NELL'ALBERO A TALE CHE L'INTERO X APPARE SIA NEL SOTTOALBERO SINISTRO CHE NEL SOTTOALBERO DESTRO.

```
public static boolean eRipetuto(AlberoB a,int x){
        if(a==null) return false;
        if(verifica(a.destro(),x) && (a.sinistro(),x)
return true;
        return eRipetuto(a.sinistro(),x) ||
eRipetuto(a.destro(),x);
}
public boolean verifica(AlberoB a,int x){
        if(a==null) return true;
        return(a.val()==x)||verifica(a.destro(),x)||veri
fica(a.sinistro(),x);
}
```

8) VERIFICA SE LA PARTE INFORMATIVA DI TUTTI I NODI FOGLIA DI A E' MAGGIORE DI K

```
public static boolean
analizzaNodiFoglia(AlberoBinario a,int k){
        if (a==null) return true;
        if(a.getSin()==null && a.getDes()==null)
return a.getVal() > k;
```

```
        return analizzaNodiFoglia(a.getSin(),k) &&
analizzaNodiFoglia(a.getDes(),k);
}
```

9) VERIFICA SE GLI ALBERI SONO IDENTICI CIOE' UNO E' LA COPIA DELL'ALTRO

```
public static boolean identici(AlberoB a,AlberoB b){
        if(a==null && b==null) return true;
        if(a==null || b==null) return false;
        if(a.val()==b.val()) return true;
        return identici(a.destro(),b.destro()) &&
identici(a.sinistro(),b.sinistro());
}
```

10) RESTITUISCE IL NUMERO DI NODI PRESENTI NELL'ALBERO

```
public static int contaNodi(AlberoB a){
        if(a==null) return 0;
        int
cont=contaNodi(a.destro())+contaNodi(a.sinistro());
        if(a!=null)
                cont++;
        return cont;
}
```

11) VERIFICA SE TUTTE LE FOGLIE CHE SI TROVANO IN A CONTENGONO UN VALORE CHE APPARE IN B

```
public static boolean foglie(AlberoB a,AlberoB b){
        if(a==null) return true;
        if(a.sinistro()==null && a.destro()==null){
                int foglia=a.val();
                if(!contiene(b,foglia))
                        return false;
        }
        return foglie(a.sinistro(),b) ||
foglie(a.destro(),b);
}

public boolean contiene(AlberoB b,int x){
        if(b==null) return false;
        if(b.val()==x) return true;
        return contiene(b.sinistro(),x) || contiente
(b.destro(),x);
}
```

12) VERIFICA SE PER OGNI NODO X DELL'ALBERO A I VALORI CONTENUTI NEL SOTTOALBERO SINISTRO DI X
SONO UGUALI A QUELLI DEL SOTTOALBERO DESTRO DI X

```
public static boolean verificaOrdinamento(AlberoB
a){
        if(a==null) return true;
        boolean b = verifica(a.sinistro(),a.destro());
        if(!b) return false;
```

```
        return verificaOrdinamento(a.sinistro()) &&
verificaOrdinamento(a.destro());
}

public boolean verifica (AlberoB a,AlberoB b){
        if(a==null && b==null) return true;
        if(a==null || b==null) return false;
        if(a.val() != b.val()) return false;
        return verifica(a.destro(),b.destro()) &&
verifica(a.sinistro(),b.sinistro));
}
```

7) VERIFICA SE L'ALBERO a CONTIENE UN INTERO x DI VALORE PARI E COMPRESO NELL'INTERVALLO [VAL MIN,VAL MAX]

```
public static boolean esisteValorePari(ABRB a,int
valmin,int valmax){
        if(a==null) return false;
        if(a.val()%2==0 && a.val() >= valmin &&
a.val() <= valmax)
                return true;
        if(a.val()>valmin && a.val()>valmax)

        return(esisteValorePari(a.sinistro(),valmin,v
almax)
        else

        return(esisteValorePari(a.destro(),valmin,val
max);
}
```

complessità temp. migliore= theta(1)
complessità temp. peggiore= theta(n) --> quando non esiste

complessità spaz. migliore= theta(1)
complessità spaz. peggiore= theta(log n) --> perchè è bilanciato

13) VERIFICA SE ESISTONO ALMENO DUE VALORI PARI COMPRESI TRA MIN E MAX

```
public int contaPari(AlberoBinario a,int min, int
max){
        if(a==null) return 0;
        int conta=contaPari(a.sinistro(),min,max) +
contaPari(a.destro(),min,max);
        if(a.val()%2==0 && a.val()>=min &&
a.val()<=max)
                return conta++;
        return contaPari(a.sinistro(),min,max) &&
contaPari(a.destro(),min,max);
}
```

14) VERIFICA SE IL VALORE V COMPARE NELLA STESSA PROFONDITA' NELL'ALBERO A E NELL'ALBERO B O SE IL VALORE V NON APPARE IN NESSUNO DEI DUE ALBERI

```
public boolean valoreStessaProfondità(AlberoB
a,AlberoB b,int v){
		if(a==null && b==null) return true;
		if(a==null || b==null) return false;
		if(a.val()==v && b.val()==v) return true;
		return
valoreStessaProfondità(a.sinistro(),b.sinistro(),v) ||

		valoreStessaProfondità(a.destro(),b.sinistro()
,v) ||

		valoreStessaProfondità(a.sinistro(),b.destro()
,v) ||

		valoreStessaProfondità(a.destro(),b.destro(),
v);
}
```