Soluzione appello del 25 luglio 2016

Esercizio1

```
class ParseTree<A,B extends A> implements ParseTreeAPI<A,B>{
    //Stato Concreto
    private A label;
    private Vector<ParseTree<A,B>> sons;
    //AF(c)=[] se c.label==null
    //AF(c)=[a-(t1,...,tk) se (a==c.label) && (k==c.sons.size()-1) &&
    //                        ti=AF(c.sons.get(i-1)) (per i\in[1..k])
    //I(c)=(if c.sons!=null then c.label!=null)
/* (punto b) */
//ADDITIONAL(equals,clone,elements,toString)
//equals ereditata perch√® mutable (coincide con ==)
    public ParseTree<A,B> clone() throws CloneNotSupportedException{
        ParseTree<A,B> res = (ParseTree<A,B>) super.clone();
        Vector<ParseTree<A,B>> sonsCopy = (Vector<ParseTree<A,B>>)sons.clone();
        res.sons = sonsCopy;
        return res;
    }
    public LinkedList<ParseTree<A,B>> elements(){
        // Calcola lista di tutti i sotto-alberi di this (this incluso)
        LinkedList<ParseTree<A,B>> res = new LinkedList<ParseTree<A,B>>();
        res.add(this);
        if (isEmpty()||isLeaf()) return res;
        for(ParseTree<A,B> t: sons){
            for(ParseTree<A,B> tt: t.elements()){res.add(tt);}}
        return res;
    }
}
```

Esercizio2

```
(* punto (a) *)
exception ValueNotFound;;
module type PATHD =
sig type pathD
    val zero: unit -> pathD
    val add: int -> pathD -> pathD
    val addP: pathD -> pathD -> pathD
    val isEmpty: pathD -> bool
    val size: pathD -> int
    val next: pathD -> int
    val rest: pathD -> pathD
end;;
(* punto (b.1) *)
module type PATHDE =
sig type pathDE
    val zero: unit -> pathDE
    val add: int -> pathDE -> pathDE
    val addP: pathDE -> pathDE -> pathDE
    val isEmpty: pathDE -> bool
    val size: pathDE -> int
    val next: pathDE -> int
    val rest: pathDE -> pathDE
    val sub: pathDE -> pathDE -> int
end;;
module PathDE =
(* punto (b.2) *)
(struct
    type pathDE = PathD.pathD
        (* AF(c) = c -- c √® gia un valore astratto in [PathD.pathD]
```

```
          I(c) = true  *)
(* punto (b.3) *)
    let zero() = PathD.zero()
    let add n p = PathD.add n p
    let addP p q = PathD.addP p q
    let isEmpty p = PathD.isEmpty p
    let size p = PathD.size p
    let next p = PathD.next p
    let rest p = PathD.rest p
(* punto (b.4) *)
    let rec sub p s =
            let rec prefix p s =
                    if isEmpty s then true
                    else ((next p)=(next s)) && (prefix (rest p) (rest s))
            in if (size p)<(size s) then raise ValueNotFound
               else if prefix p s then 0 else 1+(sub (rest p) s)
end:PATHDE)

Esercizio3
class ValueNotFound extends Exception{}
/* (punto 1) */
class PathDE extends PathD{
//stato concreto solo ereditato
        //AF(c) = AF(c.super)
        //I(c) = true
/* (punto 2) */
//Ereditate ma overridden per nuovo sottotipo calcolato
    public PathDE(){super();
    }
    public PathDE add(int x){return (PathDE) super.add(x);
    }
    public PathDE addP(PathAPI p){return (PathDE) super.addP(p);
    }
    public PathDE rest()throws invalidOperationException{
        return (PathDE) super.rest();
    }
//isEmpty e size sono integralmente ereditate
//Nuova operazione pubblica e (ausiliarie private)
    private boolean prefix (PathDE s){
        if (s.isEmpty()) return true;
        else try {PathDE temp = (PathDE)rest();
                  return ((next()==s.next()) && (temp.prefix(s.rest())));}
             catch(Exception e){return false;}
    }
    public int sub(PathDE s) throws ValueNotFound{
        if (size()<s.size()) throw new ValueNotFound();
        PathDE temp = this;
        for(int i=0;i<size()-s.size();i++){
            if(temp.prefix(s)) return i;
            try {temp = (PathDE)temp.rest();} catch(Exception e){}
        }
        throw new ValueNotFound();
    }
}
```