```
Esercizio 1 - Soluzione
(a)
module Graph =
(struct
   type ('a,'b)graph = 'a Node.node list * (('a,'b)Edge.edge list)
   (*
    AF(c) = <{n1,...,nk},{e1,...,eh}> if fst(c) = [nk,...,n1] && snd = [eh,...,e1]
    I(c) = fst(c) = [nk,...,n1] && snd = [eh,...,e1] =>
           (forall ei\in [1..h]) (Edge.getS(ei),Edge.getT(ei)\in {e1,...,eh})
   *)

   (* public and auxiliaries definitions *)
end:GRAPH);;
(b)
   let mk() = ([],[])
   let isIn eq y yy = (* auxiliary inclusion with equality *)
        let g a x = (a || eq x y) in List.fold_left g false yy
   let adde (nodes,edges) e = (* auxiliary add for singular edge *)
        let b1 = isIn (Edge.eq) e edges and
            b2 = not (isIn (Node.eq) (Edge.getS e) nodes) and
            b3 = not (isIn (Node.eq) (Edge.getT e) nodes) in
        if b1 || b2 || b3 then (nodes,edges) else (nodes,e::edges)
    let addE el (nodes,edges) =
        List.fold_left adde (nodes,edges) el
    let isEin e (nodes,edges) = isIn (Edge.eq) e edges
(c)
    let copy (nodes,edges) = (nodes,edges)
    let eq (nodes1,edges1) (nodes2,edges2) =
        if (List.length nodes1 = List.length nodes2)&&
           (List.length edges1 = List.length edges2)
        then let pN = (List.fold_left
                       (fun a n -> a && isIn (Node.eq) n nodes2) true nodes1) and
                 pE = (List.fold_left
                       (fun a n -> a && isIn (Edge.eq) n edges2) true edges1) in
             (pN && pE)
        else false



Esercizio 2 - Soluzione
(a)
public class Forest<A,B> extends Graph<A,B>{
    Vector<Edge<A,B>> edges;
    /*
     AF(c) = AF(c.super())
     I(c) = isEin(e) iff (e.equals(c.edges.get(i), for some i: 0<=i<c.edges.size())
     */
(b)
    /* costruttore e addN ereditati */
    public void addE(Vector<Edge<A,B>> E){
        if (E == null) return;
        Vector<Edge<A,B>> Etemp = new Vector<Edge<A,B>>();
        for (int i=0; i<E.size(); i++){
            Edge<A,B> temp = E.get(i);
            if (isEin(temp)) continue;
            if (indegree(temp.getT())) break;
            edges.add(temp); Etemp.add(temp);
        }
        super.addE(Etemp);
    }
```

```
61      private boolean indegree(Node<A> N){//N non null -- assunzione di uso
62          for (Edge<A,B> e: edges){
63              if (N.equals(e.getT())) return true;
64          }
65          return false;
66      }
67      /* isEin, nodes e toString ereditati */
68      public Forest<A,B> clone() throws CloneNotSupportedException{
69          Forest<A,B> ret = (Forest<A,B>) super.clone();
70          ret.edges = new Vector<Edge<A,B>>();
71          for(int i=0; i<=edges.size(); i++){
72              ret.edges.add(edges.get(i));
73          }
74          return ret;
75      }
76      /* eq ereditata */
77  (c)
78      public Vector<Node<A>> roots(){
79          Vector<Node<A>> ret = new Vector<Node<A>>();
80          for(Node<A> n: nodes()){
81              if (!indegree(n)) ret.add(n);
82          }
83          return ret;
84      }
85      public Vector<Node<A>> getSons(Node<A> N) throws IllegalArgumentException{
86          if (N == null) throw new IllegalArgumentException("getSons");
87          Vector<Node<A>> ret = new Vector<Node<A>>();
88          for(Edge<A,B> e: edges){
89              if (N.equals(e.getS())) ret.add(e.getT());
90          }
91          return ret;
92      }
93  }
94
95
96
97
98
```