```
(*
 Esercizio1.
    Sia stree un tipo astratto per alberi NON MODIFICABILI, con nodi
    interni etichettati con tipo generico A e nodi foglia etichettati con
    tipo generico B. In aggiuta ai costruttori per albero vuoto e non,
    STree fornisce le seguenti operazioni:
        isEmpty [isRoot, isLeaf] che, applicata ad un stree s, restituisce
            true se s è l'abero vuoto [ha radice, è foglia], false altrimenti;
        getRoot che, applicata ad un stree s, restituisce l'etichetta della
            radice, se s ha radice, eccezione altrimenti;
        getSons che, applicata ad un stree s, restituisce la lista degli alberi
            figli della radice, se s ha radice, eccezione altrimenti;
        getLeaf che, applicata ad un stree s, restituisce l'etichetta del nodo
            se s è una foglia, eccezione altrimenti;
        update che, applicata ad un stree s, ad un cammino p dalla radice,
            e ad un albero r, restituisce un albero che differisce da s nel
            sottoalbero al cammino p. Tale sottoalbero se esiste è rimpiazzato
            da r, altrimenti lascia tutto invariato.
    Si fornsca un API ed un ADT (con le sole implementazioni di ...)
    Allo scopo si utilizzino int list per cammini, dove n::ns dato un albero
    <u - t1 ... tn ... tm> indica il cammino ns del sottoalbero tn.
*)

module type STREE =
 sig type ('a,'b) stree
    val empty: unit -> ('a,'b) stree
    val leaf: 'b -> ('a,'b) stree
    val root: 'a -> ('a,'b) stree list -> ('a,'b) stree
    val isEmpty: ('a,'b) stree -> bool
    val isRoot: ('a,'b) stree -> bool
    val isLeaf: ('a,'b) stree -> bool
    val getRoot: ('a,'b) stree -> 'a
    val getLeaf: ('a,'b) stree -> 'b
    val getSons: ('a,'b) stree -> ('a,'b) stree list
    val update: ('a,'b) stree -> int list -> ('a,'b) stree -> ('a,'b) stree
end;;


module Stree =
(struct
 exception NoRoot of string
 exception NoLeaf of string
 include List
 type ('a,'b) stree = E | L of 'b | R of ('a * (('a,'b) stree list))
    let empty() = E
    let leaf u = L u
    let root u rr = R(u,rr)
    let isEmpty s = match s with
        | E -> true
        | _ -> false
    let isRoot s = match s with
        | R _ -> true
        | _ -> false
    let isLeaf s = match s with
        | L _ -> true
        | _ -> false
    let getRoot s = match s with
        | R(u,_) -> u
        | _ -> raise (NoRoot "getRoot")
    let getLeaf s = match s with
```

```
        | L u -> u
        | _ -> raise (NoLeaf "getLeaf")
    let getSons r = match r with
        | R(_,rr) -> rr
        | _ -> raise (NoRoot "getSons")
    let rec update s p r = match (s,p) with
        | (_,[]) -> r
        | (R(u,rr),p::pp) ->
                let rec fromTo n m = if n<=m then n::fromTo (n+1) m else []
                in let rrPaired = combine (fromTo 1 (length rr)) rr
                   in let g = fun (i,ti) -> if i=p then update ti pp r else ti
                      in R(u,map g rrPaired)
        | _ -> s
end:STREE);;

(*
 Esercizio2.
    Utilizzando Stree, si costruiscano gli alberi:
         t1 = <"a"-<"b"-<1><"a"->><"c"-<"d"-<3><5>><2>>>;
         t2 = <3>
    e si calcoli l'albero ottenuto rimpiazzando il sottoalbero di t1 al cammino
    [2;1;1] con t2.

 SOLUZIONE
*)

let td = Stree.root "d" [(Stree.leaf 3);(Stree.leaf 5)];;
let tc = Stree.root "c" [td;(Stree.leaf 2)];;
let tb = Stree.root "b" [(Stree.leaf 1);(Stree.root "a" [])];;
let t1 = Stree.root "a" [tb;tc];;
let t2 = Stree.leaf 3;;
Stree.update t1 [2;1;1] t2;;


(*
 Esercizio. Variante
    ......
        update che, applicata ad un stree s, ad un cammino p dalla radice,
            e ad un albero r, restituisce un albero che differisce da s nel
            sottoalbero al cammino p. Tale sottoalbero è rimpiazzato da r se
            ha radice; update solleva un'eccezione in tutti gli altri casi.
    ......

SOLUZIONE


module StreeV =
(struct
 exception NoRoot of string
 exception NoLeaf of string
 exception InvalidlArgument of string
 include List
 type ('a,'b) stree = E | L of 'b | R of ('a * (('a,'b) stree list))
    let empty() = E
    let leaf u = L u
    let root u rr = R(u,rr)
    let isEmpty s = match s with
        | E -> true
        | _ -> false
    let isRoot s = match s with
```

```
        | R _ -> true
        | _ -> false
    let isLeaf s = match s with
        | L _ -> true
        | _ -> false
    let getRoot s = match s with
        | R(u,_) -> u
        | _ -> raise (NoRoot "getRoot")
    let getLeaf s = match s with
        | L u -> u
        | _ -> raise (NoLeaf "getLeaf")
    let getSons r = match r with
        | R(_,rr) -> rr
        | _ -> raise (NoRoot "getSons")
    let rec update s p r = match (s,p) with
        | (R _, []) -> r
        | (R(u,rr), p::pp) ->
            (let rec fromTo n m = if n <= m then n::fromTo (n+1) m else []
            in let rrPaired = combine (fromTo 1 (length rr)) rr
                   and g = fun (i,ti) -> if i=p then update ti pp r else ti
               in try if isRoot(assoc p rrPaired)
                      then R(u,map g rrPaired)
                      else raise (NoRoot "update")
                  with Not_found -> raise (Invalid_argument "update: path"))
        | _ -> raise (NoRoot "update")
end:STREE);;
*)

(*
 Esercizio4.
    Si estenda Stree in un tipo NStree per alberi i cui nodi hanno etichette
    diverse tra loro e sono, in aggiunta, dotati delle operazioni:
        dLeafs e dRoots fornenti la lista delle etichette delle foglie, e
        delle radici presenti nell'albero.
    Si fornisca API e ADT del nuovo tipo.
*)

(*
Esercizio5.
    Si definisca la fuzione la seguente funzione per il calcolo della frontiera
    cosi definita:
        fron applicata ad un albero calcola la lista contenente la frontiera
            dell'albero indicando quali nodi sono foglie e quali nodi interni
            non espansi.
    Allo scopo si commenti e si provveda a fornire un tipo per gli elementi della
    lista che devono poter contenente valori per etichette sia di tipo A che di
    tipo B.
*)

(*
 Esercizio5.
    Si estenda Stree in un tipo KStree per alberi con out-degree massimo K.
*)

(*
 Esercizio6.
    Si estenda Stree in un tipo dotato anche delle operazioni:
        paths che dato un albero e un cammino ad un suo nodo, calcola tutti i
            cammini dal nodo alle foglie.
*)
```