Esercizi proposti – 9

In questo gruppo di esercizi, si assume che il tipo `'a ntree` sia così definito:

```
type 'a ntree = Tr of 'a * 'a ntree list
```

1. Sia data la seguente definizione di tipo per la rappresentazoine di espressioni come alberi n-ari:

```
type multi_expr =
    MultiInt of int
  | MultiVar of string
  | MultiDiff of multi_expr * multi_expr
  | MultiDiv of multi_expr * multi_expr
  | MultiSum of multi_expr list
  | MultiMult of multi_expr list
```

   Risolvere i problemi seguenti:

   (a) Scrivere una funzione `subexpr: multi_expr -> multi_expr -> bool` che, date due espressioni aritmetiche E_1 e E_2, determini se E_2 è una sottoespressione di E_1.

   (b) Scrivere una funzione `subst: multi_expr -> string -> multi_expr -> multi_expr` che, data un'espressione E, il nome di una variabile x e un'espressione E', costruisca l'espressione che si ottiene da E sostituendo con E' ogni occorrenza della variabile di nome x.

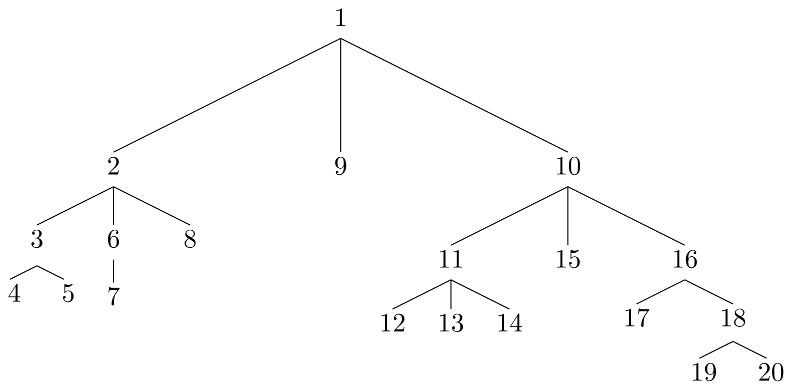
2. Nella visita in postordine degli alberi n-ari vengono prima visitati tutti i sottoalberi, poi la radice. Nella visita simmetrica viene prima visitato il sottoalbero sinistro, poi la radice, poi gli altri sottoalberi (se ve ne sono). Implementare due funzioni `postorder: 'a ntree -> 'a list` e `inorder: 'a ntree -> 'a list` che, dato un albero n-ario, riportino la lista dei suoi nodi nell'ordine in cui sarebbero visitati secondo i due algoritmi di visita.

3. Scrivere una funzione `foglie_in_lista: 'a list -> 'a ntree -> bool` che, data una lista `lst` e un albero n-ario `t`, determini se ogni *foglia* di `t` appartiene a `lst`.

4. Scrivere una funzione `num_di_foglie: 'a ntree -> int` che, applicata a un albero n-ario, riporti il numero di foglie dell'albero.

5. Una lista di interi non negativi L può determinare un sottoalbero di un albero n-ario T: quello che si ottiene, a partire dalla radice, scendendo, per ogni elemento n di L, al sottoalbero in posizione n nella lista dei sottoalberi (se esiste) – si ricordi che la posizione degli elementi in una lista si conta a partire da 0. Se la lista è più lunga del ramo cui essa conduce, oppure se a qualche livello non esiste un numero sufficiente di sottoalberi, allora L non determina alcun sottoalbero di T.

   Ad esempio, se T è l'albero sotto rappresentato



allora la lista [2;0] determina il sottoalbero che ha radice 11, la lista [2;2;1] quello che ha radice 18, [2;2;1;0] il sottoalbero costituito soltanto dal nodo 19. Le liste [1;2] e [0;1;1] non determinano alcun sottoalbero di T.

Scrivere una funzione `listaGuida: 'a list -> 'a ntree -> 'a`, che, data una lista L di interi e un albero n-ario T, riporti *la radice* del sottoalbero di T determinato da L, se L determina un sottoalbero di T, un errore altrimenti.

6. Se T è un albero n-ario etichettato da numeri interi, il costo di una foglia N di T è la somma di tutti i nodi che si trovano sul ramo che va dalla radice di T a N. Scrivere una funzione `foglia_costo: 'int ntree -> (int * int)` che, dato un albero n-ario di interi, restuisca l'etichetta e il costo della foglia più costosa dell'albero. L'albero può anche avere diversi nodi con la stessa etichetta.

7. Definire una funzione `tutte_foglie_costi: int ntree -> (int * int) list` che, applicata a un albero n-ario T etichettato da interi, riporti una lista di coppie, ciascuna delle quali ha la forma `(f,n)`, dove `f` è l'etichetta di una foglia in T e `n` il costo di tale foglia (dove il costo di una foglia è definito come nell'esercizio precedente). Anche in questo caso, l'albero può anche avere diversi nodi con la stessa etichetta.

8. (Dal compito d'esame di febbraio 2009).
Scrivere una funzione `ramo_da_lista: 'a ntree -> 'a list -> 'a -> 'a list` che, dato un albero T, una lista L senza ripetizioni e un'etichetta k, riporti, se esiste, un ramo di T dalla radice a una *foglia* etichettata da k che passi per tutti gli elementi di L esattamente una volta e contenga solo nodi etichettati da elementi di L (in pratica, il cammino deve essere una permutazione di L). Se un tale cammino non esiste, la funzione solleverà un'eccezione.

9. (Dal compito d'esame di settembre 2010).
Scrivere una funzione `ramo_di_primi: int ntree -> int` che, applicata a un albero n-ario di interi, riporti, se esiste, una **foglia** n dell'albero tale che il ramo dell'albero dalla radice a n sia costituito da tutti numeri primi.

10. (Dal compito d'esame di giugno 2011, adattato agli alberi n-ari).
Definire una funzione `path_non_pred: ('a -> bool) -> 'a ntree ->`

`'a list`, che, applicata a un predicato `p: 'a -> bool` e a un albero `t: 'a ntree`, riporti, se esiste, un cammino dalla radice a una foglia di `t` che non contenga alcun nodo che soddisfa `p`. La funzione solleverà un'eccezione se un tale cammino non esiste.

11. (Dal compito d'esame di settembre 2011, adattato agli alberi n-ari). Scrivere un predicato `same_structure: 'a ntree -> 'b ntree -> bool` che determini se due alberi n-ari hanno la stessa struttura (cioè se essi sono uguali quando si ignorano le rispettive etichette).

12. (Dal compito d'esame di luglio 2009, adattato agli alberi n-ari). Si considerino le seguenti dichiarazioni di tipo, per la rappresentazione di colori e associazioni di colori:

```
type col = Rosso | Giallo | Verde | Blu
type 'a col_assoc = (col * 'a list) list
```

Scrivere un programma con una funzione `ramo_colorato: 'a -> 'a col_assoc -> 'a ntree -> 'a list`, che, dato un valore `x`, un'associazione di colori e un albero n-ario, riporti – se esiste – un ramo a colori alterni, dalla radice dell'albero a una *foglia* etichettata da `x`. Se un tale ramo non esiste, solleverà un'eccezione (si veda l'esercizio 14 del gruppo 8 che propone lo stesso problema per gli alberi binari).