Programmazione Funzionale – Settembre 2021

Nota: è indispensabile specificare il tipo e dare una descrizione dichiarativa di ogni funzione ausiliaria utilizzata (anche locale), altrimenti non verrà presa in considerazione (ad eccezione delle funzioni il cui tipo e specifica sono già dati nel testo).

1. Scrivere una funzione `apply`: $(\alpha \times \alpha)$ `list` $\to \alpha$ `list` $\to \alpha$ `list`, tale che `apply asslist elle` riporti la lista che si ottiene da `elle` sostituendo ogni elemento che occorre come chiave in `asslist` con il valore ad esso associato, e lasciando immutati gli altri (`asslist` è una lista associativa in cui chiavi e valori sono dello stesso tipo). Ad esempio si avrà `apply [(2,4);(3,6);(5,10);(8,16);(11,22)] [4;5;7;11] = [4;10;7;22]`.

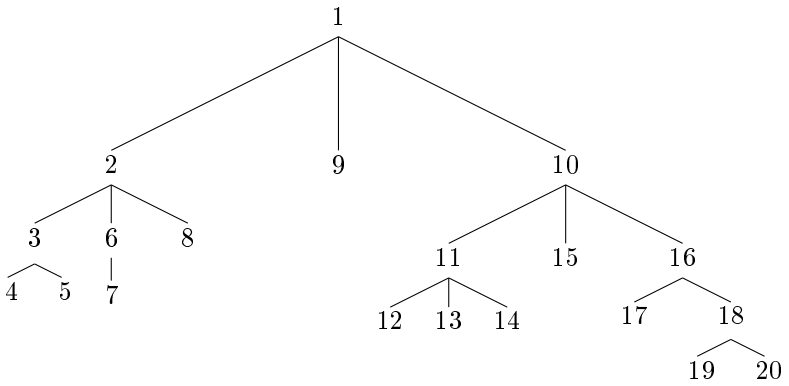
   Scrivere due versioni della funzione: una che non utilizzi funzioni di ordine superiore, e un'altra che faccia ricorso a `List.map`.

2. Definire un tipo di dati α `ntree` per la rappresentazione di alberi n-ari e scrivere una funzione

   `cerca_foglia`: ($\alpha \times$`int`) `list` $\to \alpha$ `ntree` $\to \alpha$,

   che, applicata a una lista associativa `guida` e a un albero `t` riporti la foglia di `t` che si ottiene percorrendo il ramo che (partendo dalla radice) per ogni nodo intermedio x, se n è il valore associato a x in `guida`, scende all'n-esimo sottoalbero di x, fino ad arrivare appunto a una foglia. Se `guida` non associa alcun valore a x oppure gli associa n ma l'n-esimo sottoalbero non esiste, allora la funzione solleverà un'eccezione.

   Ad esempio, se `t` è la rappresentazione dell'albero:



   e `guida=[(1,3);(2,2);(3,1);(10,1);(16,2);(11,2)]`, allora `cerca_foglia guida t` riporterà 13 (dalla radice 1 si va sul suo terzo figlio (10), da qui al primo figlio (11), e da 11 al secondo figlio: 13, che è una foglia). Invece `cerca_foglia [(1,3);(2,2);(3,1);(16,2);(11,2)] t` e `cerca_foglia [(1,4);...] t` solleveranno un'eccezione.