Programmazione funzionale (OCaml)

https://taroccoesbrocco.github.io/progfunz.html

Esonero 2 — 10 giugno 2025

Esercizio 1 (15 punti)

1. (7 punti) Definire una funzione `from_stringlist_to_string : string list -> string` tale che `from_stringlist_to_string lst` restituisca la stringa che si ottiene concatenando le stringhe nella lista `lst` inserendo uno spazio tra due stringhe successive di `lst` (ma senza alcuno spazio prima della prima stirnga e dopo l'ultima stringa). In particolare, nessuno spazio deve essere inserito se `lst` contiene nessuno o un elemento. Per esempio,

   - `from_stringlist_to_string [] = ""`
   - `from_stringlist_to_string ["ci"] = "ci"`
   - `from_stringlist_to_string ["ci";"ao"] = "ci ao"`.
   - `from_stringlist_to_string ["ci";"a";"o"] = "ci a o"`.

   Suggerimento: Utilizzare l'operatore infisso di concatenazione di stringhe `^`.

2. (8 punti) Si consideri la struttura dati per gli alberi n-ari con nodi di tipo `'a` definita a lezione come

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

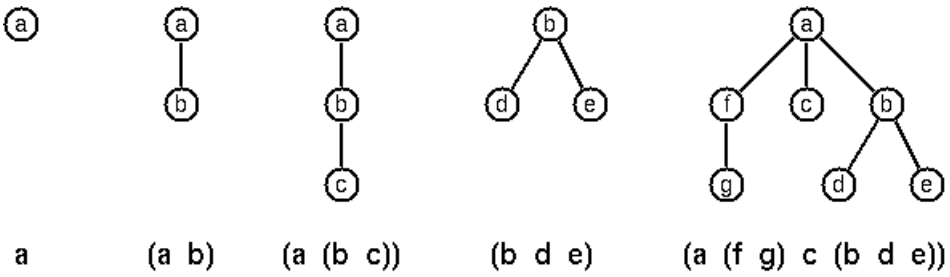
   Definire una funzione `lispy : char ntree -> string` tale che `lispy t` restituisca la stringa che rappresenta l'albero `t` di caratteri nella maniera seguente:

   (a) se `t` è un nodo `'c'` senza figli, allora `lispy t` è la stringa `"c"`;

   (b) se `t` un nodo `'c'` con figli `t1`, ..., `tn`, allora `lispy t` è la stringa `"(c s1 ... sn)"` dove `s1`, ..., `sn` sono le stringhe ottenute applicando ricorsivamente `lispy` a `t1`, ..., `tn`, rispettivamente.

   Per esempio (si vedano anche le rappresentazioni grafiche qui sotto per avere un'idea):

   - `lispy (Tr('a',[])) = "a"`
   - `lispy (Tr('a', [Tr('b',[])])) = "(a b)"`
   - `lispy (Tr('a', [Tr('b', [Tr('c',[])])])) = "(a (b c))"`
   - `lispy (Tr('a', [Tr('b',[]); Tr('c',[])])) = "(a b c)"`
   - `lispy (Tr('a', [Tr('f', [Tr('g',[])]); Tr('c',[]); Tr('b', [Tr('d',[]); Tr('e',[])])])) = "(a (f g) c (b d e))"`.



   Suggerimenti: Si faccia attenzione agli eventuali spazi tra i nodi. Si possono utilizzare:

   - la funzione `from_stringlist_to_string` del punto precedente, anche se non è stata definita;
   - la funzione `String.make : int -> char -> string` tale che `String.make n c` è una stringa di lunghezza `n` che ripete `n` volte il carattere `c`. Per esempio, `String.make 1 'c' = "c"`.

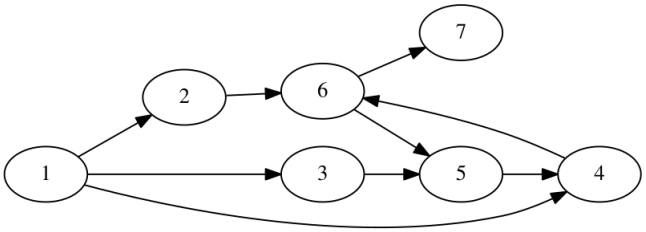
Esercizio 2 (20 punti)

Si consideri la struttura dati per i grafi *non orientati* con nodi di tipo `'a` definita a lezione come

```
type 'a graph = ('a * 'a) list
```

Per esempio, la versione non orientata del grafo di interi (cioè di tipo `int graph`) qui sotto è rappresentata da

```
let graph = [(1,2); (1,3); (1,4); (2,6); (3,5); (4,6); (5,4); (6,5); (6,7)].
```



Si assume che nei grafi non ci siano nodi ripetuti, cioè che tutti i nodi abbiano etichette diverse, che tra due nodi ci sia al più un arco, e che non ci siano nodi isolati, cioè che ogni nodo abbia almeno un arco incidente.

1. (5 punti) Definire una funzione `degree : 'a graph -> 'a -> int` tale che `degree g n` restituisca il grado del nodo `n` nel grafo `g`, cioè il numero di nodi che sono immedatamente accessibili da `n` (ignorando l'orientamento). Per esempio, `degree graph 1 = 3` e `degree graph 6 = 4`.

2. (5 punti) Definire una funzione `nodes : 'a graph -> 'a list` tale che `nodes g` restituisca la lista nei nodi del grafo `g`, senza ripetizioni. Per esempio, `nodes graph = [7; 5; 6; 4; 3; 1; 2]`. È preferibile (per ottenere il punteggio massimo) utilizzare una funzione ausiliaria che sia *ricorsiva di coda/iterativa*, cioè tale che dopo ogni chiamata ricorsiva le uniche operazioni possibili siano `let ... in` e `if ... then ... else`.

3. (5 punti) Definire una funzione `nodes_with_degree : 'a graph -> ('a * int) list` tale che `nodes_with_degree g` restituisca la lista dei nodi del grafo `g`, ciascun nodo con il suo grado. Per esempio, `nodes_with_degree graph = [(7, 1); (5, 3); (6, 4); (4, 3); (3, 2); (1, 3); (2, 2)]`. Si possono utilizzare le funzioni `degree` e `nodes` dei punti precedenti anche se non sono state definite.

   Suggerimento: Si può utilizzare la funzione `List.combine : 'a list -> 'b list -> ('a * 'b) list` che trasforma una coppia di liste della stessa lunghezza in una lista di coppie. Per esempio, `combine [a1;...;an] [b1;...;bn] = [(a1,b1);...;(an,bn)]`.

4. (5 punti) Definire una funzione `ordered_nodes : 'a graph -> 'a list` tale che `ordered_nodes g` restituisca la lista dei nodi del grafo `g` ordinata in modo *decrescente* secondo il grado. Per esempio, `ordered_nodes graph = [6; 5; 4; 1; 3; 2; 7]`. Si può utilizzare la funzione `nodes_with_degree` del punto precedente anche se non è stata definita.

   Suggerimento: Si possono utilizzare le funzioni `List.sort` e `compare` viste a lezione.