Esercizi proposti – 5

1. Definire:

   (a) funzioni ricorsive di coda equivalenti alle funzioni filter_vicini e raccolti così definite (le funzioni `in_labirinto` e `find_content` sono state definite a lezione):

   ```
   (* filter_vicini int -> (int * int) list -> (int * int) list *)
   let rec filter_vicini dim = function
       [] -> []
     | casella::rest ->
         if in_labirinto dim casella
         then casella::filter_vicini dim rest
         else filter_vicini dim rest

   (* raccolti : ('a * 'b list) list -> 'b list -> 'a list *)
   let rec raccolti contenuti = function
       [] -> []
     | casella::rest ->
         (find_content contenuti casella) @ (raccolti contenuti rest)
   ```

   (interessa l'ordine degli elementi nel risultato?).

   (b) `combine: 'a list -> 'b list -> ('a * 'b) list`, tale che:

   ```
   combine [x1;x2;...;xn] [y1;y2;...;yn] =
           [ (x1,y1); (x2,y2); .... (xn,yn) ]
   ```

   La funzione solleva un'eccezione se le due liste in input hanno lunghezza diversa. (Notare che il modulo `List` contiene una funzione con questo nome, ma qui si chiede di ridefinirla per esercizio).

   (c) `split: ('a * 'b) list -> 'a list * 'b list`, tale che:

   ```
   split [(x1,y1); (x2,y2); .... (xn,yn)] =
         ([x1;x2;...;xn],[y1;y2;...;yn])
   ```

   (Notare che il modulo `List` contiene una funzione con questo nome, ma qui si chiede di ridefinirla per esercizio).

   (d) `cancella: 'a -> ('a * 'b) list -> ('a * 'b) list` che implementa la cancellazione di una chiave da una lista associativa. Per come sono gestite le liste associative, in cui l'inserimento (in testa) di un nuovo legame non cancella gli eventuali altri legami esistenti per la stessa chiave, vanno cancellate tutte le coppie che hanno come primo elemento la chiave data (Notare che il modulo `List` contiene la funzione `remove_assoc`, che però cancella solo la prima coppia con la chiave data).

2. Se si rappresentano insiemi finiti mediante liste senza ripetizioni, implementare le operazioni di unione, intersezione, differenza, tutte di tipo `'a list -> 'a list -> 'a list`.

   Definire inoltre una funzione `subset: 'a list -> 'a list -> bool` che rappresenti la relazione insiemistica di sottoinsieme (proprio o improprio): applicata a due liste `set1` e `set2` determina se `set1` rappresenta un sottoinsieme di `set2`.

3. Definire le funzioni:

```
explode: string -> char list
implode: char list -> string
```

   che trasformano, rispettivamente, una stringa in una lista di caratteri (explode) e viceversa (implode). Ad esempio `explode "CIAO" = ['C'; 'I'; 'A'; 'O']` e `implode ['C'; 'I'; 'A'; 'O'] = "CIAO"`.

4. Scrivere una funzione `intpairs: int -> (int*int) list` che, applicata a un intero positivo `n`, riporti una lista di tutte le coppie di interi `(x,y)` con `x` e `y` compresi tra `1` e `n`.

   Ad esempio, `intpairs 3` riporterà la lista `[(1, 1); (1, 2); (1, 3); (2, 1); (2, 2); (2, 3); (3, 1); (3, 2); (3, 3)]` (o una sua permutazione).

   Suggerimento: risolvere il seguente sottoproblema: dato un elemento `y` e una lista `[x1;x2;...;xn]` (che poi sarà grave la lista di tutti i numeri compresi tra `1` e `n`), costruire la lista `[(y,x1);(y,x2);....;(y,xn)]`.

5. Definire una funzione `trips: 'a list -> ('a * 'a * 'a) list` che, applicata a una lista `lst`, riporti la lista di tutte le triple adiacenti di elementi di `lst` (la lista vuota se `lst` ha meno di 3 elementi).

   Ad esempio `trips [1;2;3;4;5] = [(1, 2, 3); (2, 3, 4); (3, 4, 5)]` (o una sua permutazione).

6. Per "sottolista" di una lista `L` si intende una "sottosequenza" di `L`, cioè una lista contenente elementi che occorrono consecutivi nella lista `L`, nello stesso ordine. Ad esempio `[1;2;3]` è una sottolista di `[0;1;2;3;4]` ma non di `[0;1;2;5;4;3]`. Formalmente, una lista `L1` è una sottolista di `L` se esistono liste `PRIMA` e `DOPO` (eventualmente vuote) tali che `L = PRIMA @ L1 @ DOPO`.

   Definire una funzione `choose: int -> 'a list -> 'a list list` che, applicata a un intero positivo k e una lista L, riporti una lista contenente tutte le sottoliste di L di lunghezza k.

   Ad esempio, `choose 3 [1;2;3;4;5] = [[1; 2; 3]; [2; 3; 4]; [3; 4; 5]]` (o una sua permutazione).

   Suggerimento: utilizzare la funzione `take` definita a lezione.

7. Definire una funzione `strike_ball: 'a list -> 'a list -> (int * int)` che, applicata a due liste, `test` e `guess`, che si assumono della stessa lunghezza, riporti una coppia `(strike,ball)` dove `strike` è il numero di elementi di `test` che occorrono anche in `guess`, ma in diversa posizione, e `ball` è il numero di elementi di `test` che occorrono in `guess` nella stessa posizione in cui sono in `test`.

   Ad esempio, `strike_ball [1;2;3;4;5;6] [5;6;3;4;2;10] = (3, 2)`: ci sono 3 elementi "fuori posto" (2,5,6) e 2 nella stessa posizione (3 e 4).

   Suggerimento: scandire contemporaneamente le due liste.

8. Implementare gli algoritmi di ordinamento veloce (quick sort) e ordinamento per inserimento (insert sort).