Programmazione Funzionale

Esercitazione 16 – Operazione sui grafi

In questi esercizi assumiamo di aver definito i grafi come lista di archi;

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

In questo contesto un grafo e anche un lista associativa.

Esercizio 1. Il collegamento ordinato di due grafi (V_1, E_1) e (V_2, E_2) e definito come

$$G_1 \nabla G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(x_1, x_2) | x_1 \in V_1, x_2 \in V_2\}).$$

Vogliamo implementare l'operazione di collegamento;

1. Definire una funzione `gen_edges : 'a -> 'a list -> 'a graph` che prende un elemento `x` e una lista `lst` e restituisce la lista dei archi `(x,a)` per ogni elemento `a` della lista `lst`.

   Si puo usare `List.map` e una funzione `f:'a -> 'a -> ('a * 'a)`.

2. Definire una funzione `joinSingle : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e un grafo `grafo` e restituisce `grafo` in cui sono stati aggiunti archi della forma `(x,a)` per ogni nodo `a` del grafo.

   Si puo usare la funzione `nodes 'a graph -> 'a list` e una funzione ausiliaria.

3. Definire una funzione `flatten : 'a list list -> 'a list` che prende una lista di liste `[l1;...;ln]` e restituisce la lista `l1 @ ... @ ln`.

4. Definire una versione generalizzata di `gen_edges`. Definire `generate: 'a list -> 'a list -> 'a graph` che prende una lista `[x1;...;xn]` e una lista `[a1;...;ak]` e restituisce la lista dei archi della forma `(xi,aj)`.

   Si puo usare `gen_edges` e `List.map` e poi concatenare tutte le liste ottenute.

5. Definire una funzione `joinOrd : 'a graph -> 'a graph -> 'a graph` che prende due grafi `g1` e `g2` e restituisce il collegamento ordinato di `g1` con `g2`.

6. Il collegamento (non ordinato) di due grafi (V_1, E_1) e (V_2, E_2) e definito come

   $$G_1 \nabla G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup (x_1, x_2) | x_1 \in V_1, x_2 \in V_2 \cup (x_2, x_1) | x_1 \in V_1, x_2 \in V_2).$$

   Usando le domande e funzioni precedenti implementare la funzione `join: 'a graph -> 'a graph -> 'a graph` che prende due grafi `g1` e `g2` e restituisce il collegamento (non ordinato) di `g1` con `g2`.

Esercizio 2. Il *prodotto* di due grafi $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ e definito come:

$$G_1 \times G_2 = (V_1 \times V_2, \{((x_1, y_1), (x_2, y_2)) \mid (x_1, x_2) \in E_1 \text{ and } (y_1, y_2) \in E_2\})$$

Vogliamo implementare il prodotto di due graffi:

1. Definire una funzione `product : 'a list -> 'b list -> ('a * 'b) list` che prende due liste `la` e `lb` e restituisce la lista delle coppie `(a,b)` dove `a` appartiene a `la` e `b` appartiene a `lb`.
2. Definire una funzione `swap : ('a * 'a) -> ('b * 'b) -> ('a * 'b) * ('a * 'b)` che prende due coppie `(a1,a2)` e `(b1,b2)` e restituisce `((a1,b1),(a2,b2))`.
3. Mediante le funzione `product` e `swap`, definire `graphprod: 'a graph -> 'b graph -> ('a * 'b) graph` che prende due grafi e restituisce il loro grafo prodotto.

   Si può usare una funzione `list_to_set : 'a list -> 'a list` che toglie le ripetizioni della lista in entrata.

Esercizio 3. La contrazione di un nodo in un grafo $G = (V, E)$ e definita come

$$ctr(G, x) = (V \setminus \{x\}, (E \setminus \{(a, b) \mid a = x \text{ or } b = x\}) \cup \{(a, b) \mid (a, x) \in E \text{ and } (x, b) \in E\})$$

In altre parole tutti i archi che partevano o arrivavano in x sono tolti e quando un cammino della forma (a, x, b) occorreva in G e diventato un arco a, b nel grafo $ctr(G, x)$.

1. Definire una funzione `remove : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e toglie da un grafo tutti i archi che contengono l'elemento `x`.
2. Definire una funzione `edges : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e un grafo `grafo` e restituisce la lista dei archi della forma `(a,x)` o `(x,a)` che occorrono in `grafo`.
3. Mediante la funzione `edges` definire una funzione `bridge : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e restituisce la lista dei archi della forma `(a,b)` quando i archi `(a,x)` e `(x,b)` occorrono nel grafo.

4. Mediante `edges`,`remove` e `bridges` definire la funzione `contract : 'a -> 'a graph -> 'a graph` che prende un nodo `x` e un grafo `grafo` e restituisce il grafo che corrisponde alla contrazione di `x` in `grafo`.

Esercizio 4. La contrazione di un arco (x, y) in un grafo $G = (V, E)$ e definita come;

$$ctr(G, (x, y)) = (V \setminus \{y\}, (E \setminus \{(a, b) \mid y = a \text{ or } b = y\}) \cup \{(a, x) \mid (a, y) \in E\}) \cup \{(x, a) \mid (y, a) \in E\}$$

Vogliamo implementare quest'operazione in OCAML.

1. Definire una funzione `remove : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e toglie da un grafo tutti i archi che contengono l'elemento `x`.
2. Definire una funzione `edges : 'a -> 'a graph -> 'a graph` che prende un elemento `x` e un grafo `grafo` e restituisce la lista dei archi della forma `(a,x)` o `(x,a)` che occorrono in `grafo`.
3. Definire una funzione `substi : 'a -> 'a -> 'a graph -> 'a graph` che prende un elemento `x` un elemento `new` e un grafo `grafo` e sostituisce in ogni archo del grafo l'elemento `x` con l'elemento `new`.
4. Mediante le funzione `remove`,`edges` e `substi` definire la funzione `contract : ('a * 'a) -> 'a graph -> 'a graph` che prende un arco `(x,y)` e un grafo `grafo` e restituisce la contrazione dell'arco `(x,y)` nel grafo `grafo`.