

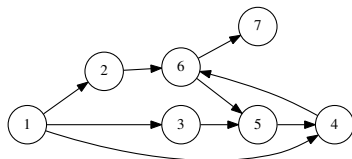
Rappresentazione mediante liste di successori

```
type 'a graph = ('a * 'a list) list
let grafo = [(1, [2;3;4]); (2, [6]); (3, [5]); (4, [6]);
            (5, [4]); (6, [5;7]); (7, [])]
```

Problema di base nei grafi: dato un nodo e un grafo, trovare i successori del nodo dato. Se il nodo non è nel grafo, sollevare un'eccezione

```
exception Nodo_inesistente
(* successori: 'a -> 'a graph -> 'a list *)
let successori x grafo =
  try List.assoc x grafo
  with Not_found -> raise Nodo_inesistente
```

Rappresentazione mediante lista di archi



```
type 'a graph = ('a * 'a) list
let grafo = [(1,2); (1,3); (1,4); (2,6); (3,5); (4,6);
            (5,4); (6,5); (6,7)]
```

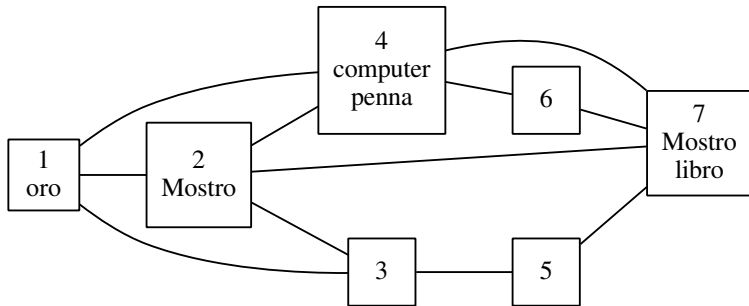
La rappresentazione è la stessa per grafi orientati e non orientati.

La differenza tra i due tipi di grafo sarà nella definizione della funzione successori (o vicini)

Vai al codice

Attraversamento di un labirinto

Problema: attraversamento di un labirinto, da una casella di entrata a una casella di uscita, senza passare per caselle che contengono un mostro, e raccogliendo tutti gli oggetti che si trovano nelle altre caselle (se ce ne sono)



Vai al codice

Una città è costituita da un insieme di posti, collegati tra loro, alcuni dei quali sono negozi che possono vendere degli oggetti.

Rappresentiamo una città mediante il tipo **city** così definito:

```
type shops = (int * string list) list
type city = (int * int) list * shops
```

(gli oggetti sono rappresentati da stringhe).

Scrivere un programma con una funzione

compra: string list -> city -> int -> int list

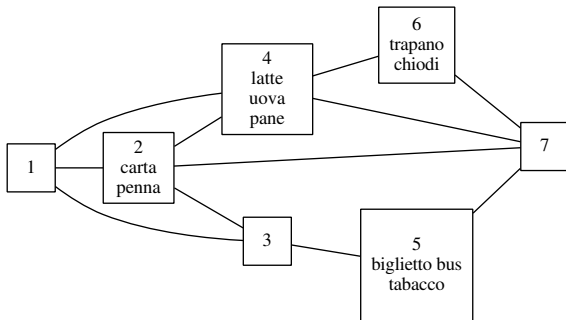
che, data una lista *senza ripetizioni* di stringhe **tobuy**, una città **c** e un intero **start**, riporti, se esiste, un cammino senza cicli in **c** che a partire da **start** contenga negozi nei quali è possibile comprare tutti gli oggetti contenuti in **tobuy**.

Esempio

city=(grafo,shops), dove:

```
let grafo = [(1,2); (1,3); (1,4); (2,3); (2,4); (2,7);  
            (3,5); (4,6); (4,7); (5,7); (6,7)];  
let negozi = [(2, ["carta"; "penna"]);  
              (4, ["latte"; "uova"; "pane"]);  
              (5, ["biglietto bus"; "tabacco"]);  
              (6, ["trapano"; "chiodi"])]
```

(i luoghi identificati da 1, 3 e 7 non sono negozi).



Esempio (II)

Ad esempio, si può avere:

```
# compra ["chiodi"; "tabacco"; "uova"; "penna"; "pane"]  
      (grafo, negozi) 1;;  
- : int list = [1; 2; 3; 5; 7; 4; 6]
```

(dove **grafo** e **negozi** sono definiti come sopra).

Potrebbe essere utile utilizzare la funzione **diff** così definita:

```
let diff lst1 lst2 =  
  List.filter  
    (function x -> not (List.mem x lst1)) lst2
```

per la quale era richiesto in un punto precedente dello stesso compito di determinare il tipo e dare una specifica dichiarativa.

[Vai al codice](#)