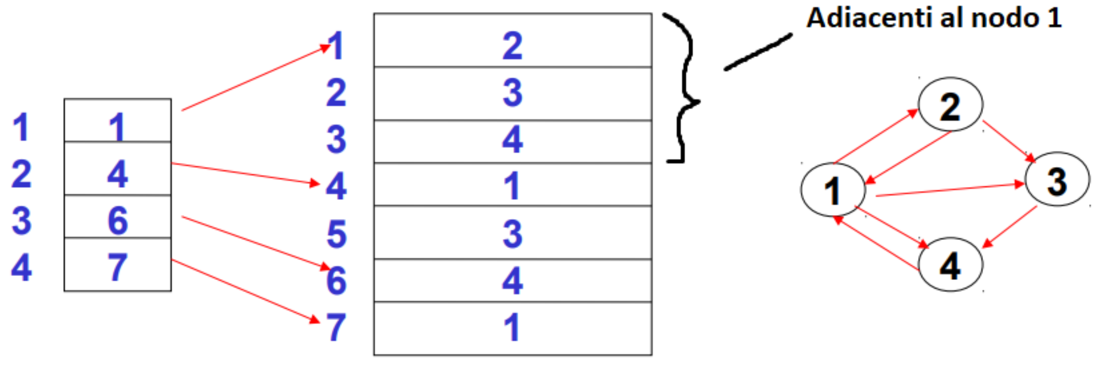
Grafi

Caratteristiche	Il **grafo** è una struttura composta da nodi e archi che rappresenta una relazione binaria su un insieme (cioè una relazione tra due elementi dello stesso insieme). Si utilizzano i **nodi** per rappresentare gli oggetti e gli **archi** per rappresentare le relazioni tra le coppie di oggetti. Verranno considerati solo grafi **orientati (o diretti)**, nei quali gli archi hanno una direzione da un nodo di partenza ad un nodo di arrivo. Formalmente, un **grafo orientato** G è una coppia <N,A> dove: - **N**: insieme finito non vuoto di nodi; - **A⊆NxN**: insieme finito di coppie ordinate di nodi, detti archi. Un **grafo non orientato**, invece, avrà coppie non ordinate: ad esempio, in un grafo orientato, le coppie (Uj, Ui) e (Ui, Uj) indicano due archi distinti (e con direzione e verso distinti), mentre in un grafo non orientato questi indicano un arco che incide sui due nodi. I nodi collegati da un arco si dicono **adiacenti**. Abbiamo anche **archi etichettati** (con etichette o pesi).
Rappresentazione con Matrice di Adiacenza	Quella con **matrice di adiacenza** è la più semplice rappresentazione: si utilizza una matrice $n \times n$ dove gli indici di righe e colonne rappresentano i nodi. Se in corrispondenza di due nodi troviamo il valore 1, allora c'è un arco che va dal nodo sulla riga a quello sulla colonna. Se il grafo è **pesato** non troviamo 0 o 1, ma bensì direttamente i pesi. Questa rappresentazione è utilizzabile anche per i grafi non orientati Se il grafo è **etichettato**, si possono associare altre informazioni al nodo (sempre sulle righe). Ad esempio si possono usare dei flag per indicare se un nodo è stato rimosso, oppure quanti archi uscenti/entranti ci sono.
Rappresentazione con Matrice di Incidenza	Anche nella **matrice di incidenza** avremo i nodi del grafo sulle righe, ma in questo caso avremo gli archi sulle colonne. Se abbiamo un **grafo orientato**, il valore nella matrice sarà +1 se l'arco è entrante, -1 se è uscente, 0 altrimenti. Se invece abbiamo un **grafo non orientato**, se il nodo è toccato da uno specifico arco il valore nella matrice sarà 1, altrimenti sarà 0.
Rappresentazione con Vettore di Adiacenza	La rappresentazione con **vettore di adiacenza** utilizza due vettori: - **vettore dei nodi**, nel quale ogni indice corrisponde ad un nodo, ed il valore corrisponde alla posizione dalla quale inizia l'elenco di nodi adiacenti nel vettore degli archi; - **vettore degli archi** che elenca tutti i nodi adiacenti. Se il grafo è etichettato o pesato, queste informazioni possono essere memorizzate in un altro vettore. Nel caso il grafo fosse non orientato, ogni arco va rappresentato due volte.

<table>
<tr>
<td>Esempio di
Vettore di Adiacenza</td>
<td>



</td>
</tr>
<tr>
<td>Rappresentazione con Liste di Adiacenza</td>
<td>In questa rappresentazione, si utilizza un vettore che rappresenta i nodi, i cui elementi puntano ad una lista di adiacenza, che contiene i nodi adiacenti a quello specifico nodo.</td>
</tr>
<tr>
<td>DFS e BFS: Introduzione</td>
<td>

Il criterio di scelta del nodo stabilisce la modalità di visita negli algoritmi di visita/ricerca.

La ricerca **DFS** è un’estensione della visita in preordine di un albero; in questa modalità il **nodo scelto** per proseguire la ricerca è sempre **quello visitato più di recente**.

Nella ricerca **BFS**, invece, i **nodi sono visitati** in ordine di **distanza crescente dal nodo di partenza** (dove la distanza è data dal numero di archi tra il nodo di partenza ed un altro nodo).
La **ricerca in ampiezza**, allora, fornisce sempre un **cammino “ottimo”**.

DFS e BFS possono essere anche usate per risolvere problemi come:

- verificare se un grafo non orientato è connesso;
- trovare tutti i sottografi connessi in un grafo.

Inoltre, entrambi gli algoritmi generano un albero (in base all’ordine in cui visitano i nodi), che avrà come radice il primo nodo generato.
Questa caratteristica risulta interessante perché può essere sfruttata per trovare il cammino minimo tra nodi.

</td>
</tr>
<tr>
<td>DFS</td>
<td>

L'**algoritmo DFS (Depth First Search)**, nel caso del grafo prevede che ogni vertice venga marcato come “visitato” quando viene visitato, per poi spostarsi su un vertice adiacente non marcato.
Se non dovessero esserci più vertici adiacenti si indietreggia lungo i vertici già visitati finchè non si trova uno non visitato, continuando così fino a quando tutti i vertici saranno stati visitati.

L’algoritmo sarà:

```
DFS(G: Grafo, u: Nodo)

ESAMINA IL NODO u E MARCALO "VISITATO"
    for(TUTTI I VERTICI v CHE APPARTENGONO AD A(U))
        ESAMINA L'ARCO (u,v)
        if(v NON E "VISITATO")
            DFS(G,v)
```

</td>
</tr>
</table>

BFS	L'algoritmo **BFS (Breadth First Search)** visita ogni nodo adiacente al vertice corrente prima di passare ad un altro vertice.
	I nodi sono visitati in ordine di distanza dal vertice corrente, dove la distanza sarà il numero di archi percorsi in un cammino da *u* a *v*.
	
	Conviene utilizzare una coda per memorizzare i vertici visitati ma non completamente esaminati, in modo tale che, quando si è pronti a spostarsi su un vertice adiacente al corrente, ci si sposta sul vecchio vertice corrente.
	
	`BFS(G:Grafo, U:Nodo)`
	`    creaCoda(Q)`
	`    inCoda(u,Q)`
	`    while(codaVuota(Q) == false)`
	`        u = leggiCoda(Q)`
	`        fuoriCoda(Q)`
	`        ESAMINA IL NODO u E MARCALO 'visitato'`
	`        for(TUTTI I NODI v ADIACENTI A u)`
	`            ESAMINA ARCO u,v`
	`            if(v NON E MARCATO 'visitato' && v NON E NELLA CODA Q)`
	`                inCoda(v,Q)`

Specifica Semantica

TIPI:
- **Grafo**: insieme G = (N,A) con N sottoinsieme finito di elementi di tipo nodo, A sottoinsieme di NxN
- **Nodo**: insieme finito qualsiasi
- **Lista**: lista di elementi di tipo nodo
- **boolean**: insieme dei valori di verità

OPERATORI:

crea() = G

POST: $G = (N,A)$ con $N = \emptyset$, $A = \emptyset$

vuoto(G) = b

POST: b = true se $N = \emptyset$, $A = \emptyset$

b = false altrimenti

inserisciNodo(u,G) = G'

PRE: $G = (N,A)$ $u \notin N$

POST: $G' = (N',A)$ $N' = N \cup \{u\}$

inserisciArco(u,v,G) = G'

PRE: $G = (N,A)$ $u \in N$, $v \in N$, $(n,v) \notin A$

POST: $G' = (N,A')$ $A' = A \cup (u,v)$

cancellaNodo(u,G) = G'

PRE: $G = (N,A)$ $u \in N$

e non esiste $v \in N$ tale che $(u,v) \in A$

oppure $(v,u) \in A$

POST: $G' = (N',A)$ $N' = N \setminus \{u\}$

cancellaArco(u,v,G) = G'

PRE: $G = (N,A)$ $u \in N$, $v \in N$, $(n,v) \in A$

POST: $G' = (N, A')$ $A' = A \setminus (u,v)$

adiacenti(u,G) = L

PRE: $G = (N,A)$ $u \in N$

POST: L è una lista che contiene una e una sola volta gli elementi di $A(u) = \{v \in N \mid (u,v) \in A)$