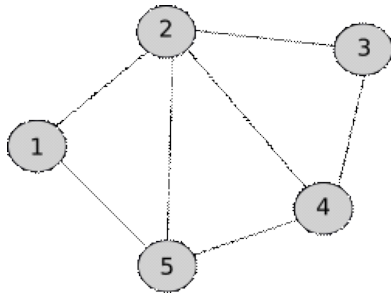


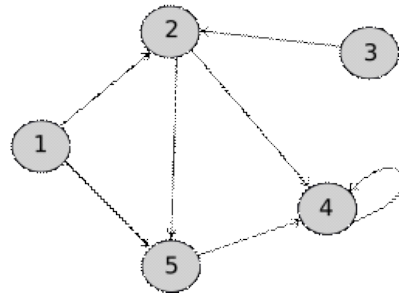
4-Grafi

- Un **grafo** è una struttura relazionale formata da un numero finito V di **vertici** (o nodi) e un numero finito E di **archi** (segmenti o spigoli) che collegano ogni nodo agli altri.

Grafo $G = (V, E)$



Grafo NON orientato

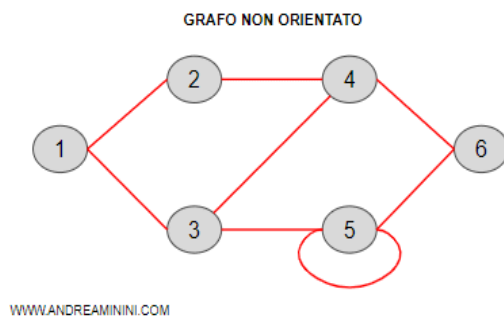


Grafo orientato

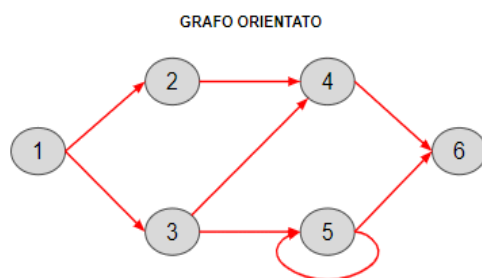
Assumiamo che ogni vertice in V sia univocamente identificato da un identificativo nel range $[1, |V|]$ abbiamo 2 metodi standard per la rappresentazione:

- liste di adiacenza;
- matrici di adiacenza;

Liste di adiacenza



NODI	LISTE DI ADIACENZA
1	2, 3
2	1, 4
3	1, 4, 5
4	2, 3, 6
5	3, 5, 6
6	4, 5

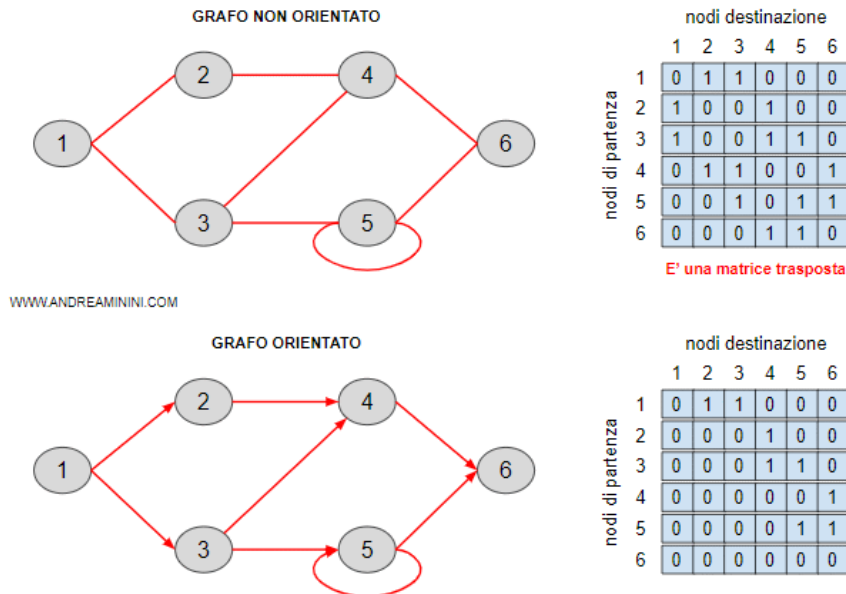


NODI	LISTE DI ADIACENZA
1	2, 3
2	4
3	4, 5
4	6
5	5, 6
6	-

Grafo $G = (V, E)$

- Array Adj di $|V|$ liste, una per ogni **vertice** in V
- Per ogni **vertice** u in V , Adj[u] contiene tutti i vertici v in V tali che esista un **arco** (u, v) in E (tutti i vertici adiacenti a u in G , memorizzati in ordine arbitrario).
- A livello implementativo, una soluzione è che Adj[u] contenga un **puntatore alla testa della lista** tali vertici.
- Nei grafi orientati e pesati il **peso** dell'arco (u, v) è memorizzato col vertice v nella lista di u .

Matrici di adiacenza



Grafo $G = (V, E)$

- Matrice $A = (a_{ij})$ di dimensione $|V| \times |V|$

$$a_{ij} = \begin{cases} 1 & \text{se } (i, j) \text{ appartiene ad } E \\ 0 & \text{altrimenti} \end{cases}$$

- Per archi pesati **memorizzo il peso** anziché il valore 1

Primitive Grafo (liste di adiacenza)	Nodo Grafo	Array di liste (struttura del grafo)
	<pre>struct nodo_ad{ int nodo; float peso; nodo_ad* next; }; typedef nodo_ad* adj_lista; //struttura nodo lista adiac.</pre>	<pre>typedef struct{ adj_lista* nodi; int dimensione; } grafo; //struttura del Grafo adj_lista* Adj = new adj_lista[n]; //allocazione lista adiacenza</pre>
<p>1-Restituisce la rappresentazione di un grafo di n vertici, identificate univocamente da 1 a n attraverso n liste di adiacenza</p> <p>2-Aggiunge l'arco orientato s,d con peso w alla lista d'adiacenza del nodo s</p>	<p>1-NEW_GRAPH</p> <pre>grafo new_graph(int n){ grafo G; G.dimensione=n; G.nodi= new adj_list[n]; for (int i=0; i<n; i++) { G.nodi[i] = NULL; } return G; }</pre>	<p>2-ADD_ARC</p> <pre>void add_arc(grafo& G, int u, int v, float w) { nodo_ad* t = new nodo_ad; t->nodo = v-1; t->peso = w; t->next = G.nodi[u-1]; G.nodi[u-1] = t; }</pre>

<p>3–Aggiunge l’arco non orientato (s,d) con peso w alla lista di adiacenza del nodo s e del nodo d</p> <p>4–Restituisce il numero dei nodi</p>	<p>3–ADD_EDGE</p> <pre>void add_edge(grafo& g, int u, int v, float w){ add_arc(g,u,v,w); add_arc(g,v,u,w); }</pre>	<p>4–GET_DIM</p> <pre>int get_dim(grafo g){ return g.dimensione; }</pre>
<p>5–Restituisce la testa della lista di adiacenza del nodo indicato</p> <p>6–Restituisce l’identificativo del nodo contenuto nell’elemento della lista di adiacenza</p>	<p>5–GET_ADJLIST</p> <pre>adj_list get_adjlist(grafo g,int u){ return g.nodi[u-1]; }</pre>	<p>6–GET_ADJNODE</p> <pre>int get_adjnode(nodo_ad* l){ return (l->nodo+1); }</pre>
<p>7–Restituisce il prossimo elemento della lista di adiacenza</p>	<p>7–GET_NEXTADJ</p> <pre>adj_list get_nextadj(adj_list l){ return l->next; }</pre>	