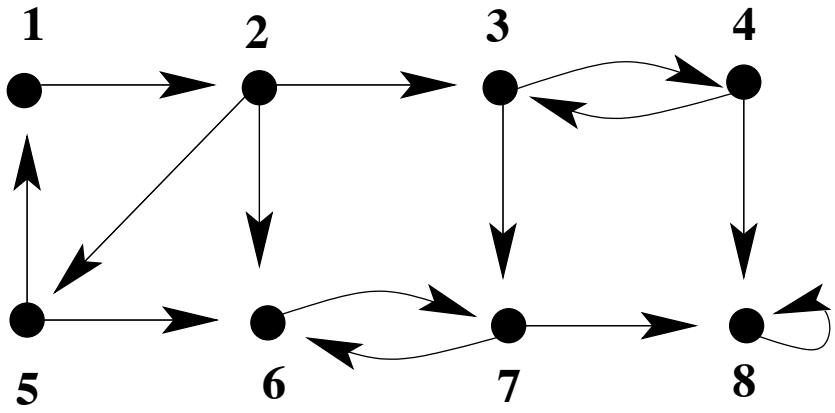
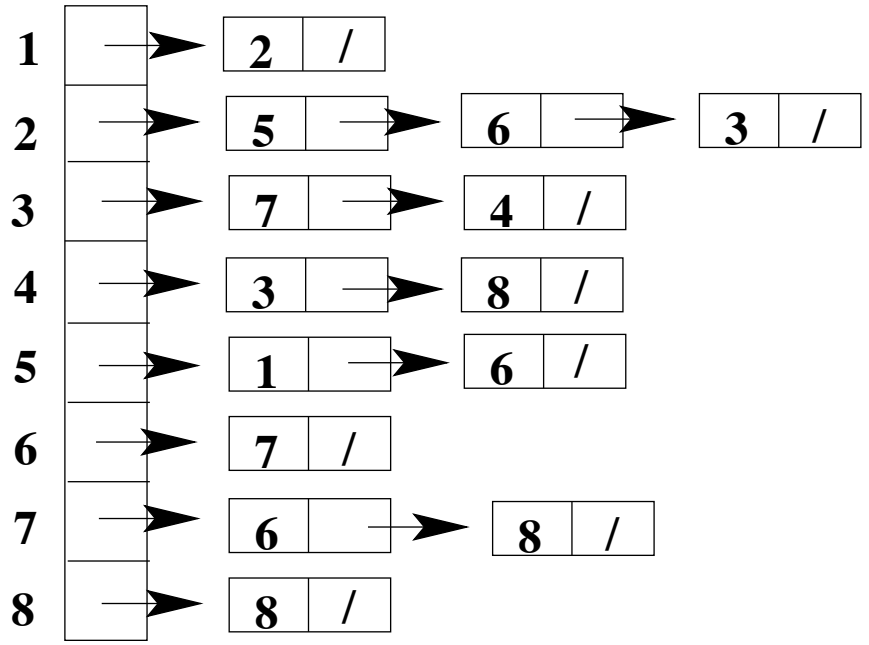
Rappresentazione di grafi

Condideriamo due modi per rappresentare un grafo (diretto o indiretto):

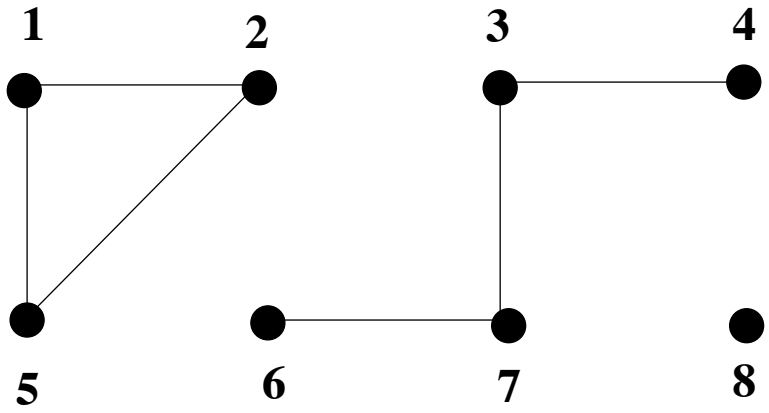
- **liste di adiacenza**: un vettore Adj con indici da 1 a n in cui $Adj[i]$ contiene un puntatore alla lista dei successori di i;
- **matrice di adiacenza**: una matrice quadrata $A = (a_{i,j})$ di dimensione $n \times n$ tale che $a_{i,j} = 1$ se $(i, j) \in E$ e $a_{i,j} = 0$ altrimenti.

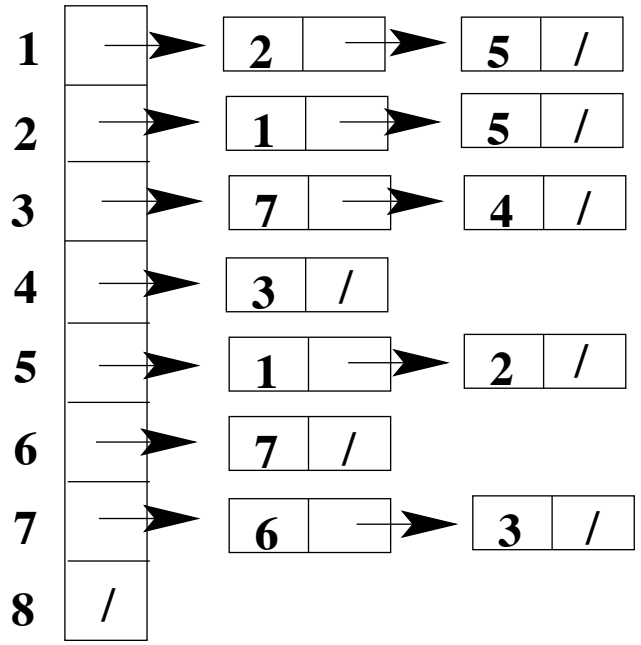
Liste di adiacenza – Grafo diretto



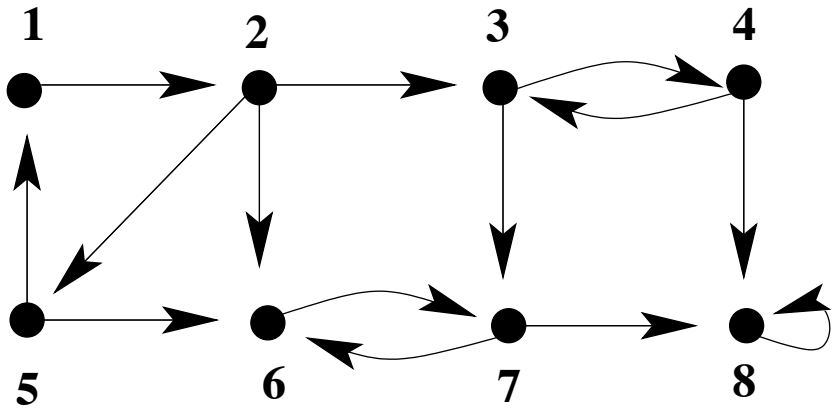


Liste di adiacenza – Grafo indiretto



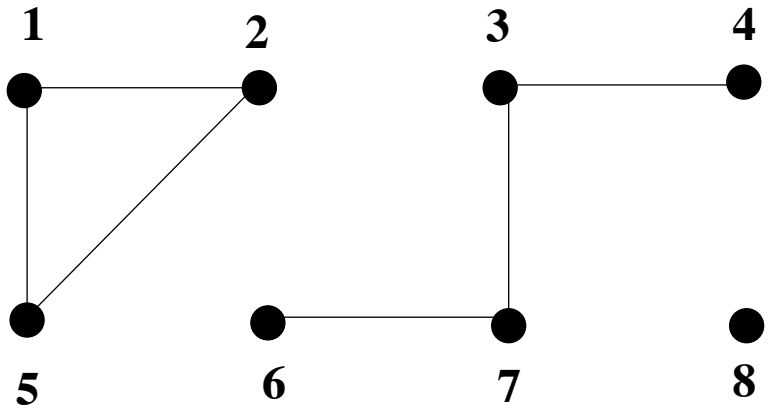


Matrice di adiacenza – Grafo diretto



	1	2	3	4	5	6	7	8
1		1						
2			1		1	1		
3				1			1	
4			1					1
5	1					1		
6							1	
7						1		1
8								1

Matrice di adiacenza – Grafo indiretto



	1	2	3	4	5	6	7	8
1		1			1			
2	1				1			
3				1			1	
4			1					
5	1	1						
6							1	
7			1			1		
8								

Rappresentazione di grafi - Spazio usato

- **Liste di adiacenza**: il vettore Adj ha dimensione n, la somma delle lunghezze delle liste di adiacenza è m per grafi diretti e $2m$ per grafi indiretti, quindi lo spazio utilizzato è $\Theta(n+m)$;
- **Matrice di adiacenza**: lo spazio usato è sempre $\Theta(n^2)$.

Un grafo si dice **denso** se $m \sim n^2$. Un grafo è **sparso** se $m \prec n^2$.

Per grafi sparsi, la dimensione della rappresentazione a liste è asintoticamente inferiore di quella della rappresentazione a matrice, quindi è da preferire.

Un vantaggio della rappresentazione a matrice è che posso verificare in tempo costante se un arco è presente nel grafo o meno.

Esercizio 1 *Si scriva una procedura che dato un grafo rappresentato con liste di adiacenza e due nodi u e v, determini se (u, v) è un arco del grafo.*

$EdgeFind(G, u, v)$

```
1: if ListSearch(Adj[u], v) = NIL then
2:     return FALSE
3: else
4:     return TRUE
5: end if
```

Qual è la **complessità** nel caso pessimo?

Esercizio 2 *Sia G un grafo diretto e u un suo vertice. Si scriva una procedura che calcola il* ***grado uscente*** *e una procedura che calcola il* ***grado entrante*** *di u.*

$ListOutDegree(G, u)$

```
1: return ListLength(Adj[u])
```

$ListInDegree(G, u)$

```
1: indegree ← 0
2: for all v ∈ V[G] do
3:     if ListSearch(Adj[v], u) ≠ NIL then
4:         indegree ← indegree + 1
5:     end if
6: end for
7: return indegree
```

$MatrixOutDegree(G, u)$

```
1: outdegree ← 0
2: for i ← 1 to n do
3:     if A[u, i] = 1 then
4:         outdegree ← outdegree + 1
5:     end if
6: end for
7: return outdegree
```

$MatrixInDegree(G, u)$

```
1: indegree ← 0
2: for i ← 1 to n do
3:     indegree ← indegree + A[i, u]
4: end for
5: return indegree
```

Esercizio 3 *Dato un grafo diretto $G = (V, E)$, il* ***grafo inverso*** *di G è il grafo $G^{-1} = (V, E^{-1})$ ove $E^{-1} = \{(v, u) \ : \ (u, v) \in E\}$. Si scriva una procedura che calcola il grafo inverso di un grafo dato.*

$ListGraphInverse(G, G^{-1})$

```
1: for each u ∈ V[G] do
2:     for each v ∈ Adj[u] do
3:         ListInsert(Adj^{-1}[v], u)
4:     end for
5: end for
```

$MatrixGraphInverse(G, G^{-1})$

```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         A^{-1}[i, j] ← A[j, i]
4:     end for
5: end for
```

Esercizio 4 *Si consideri la seguente procedura che inverte* in-place *un grafo orientato rappresentato a matrice:*

$MatrixGraphInverse(G)$

```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         if A[i, j] ≠ A[j, i] then
4:             Exchange(A, i, j, j, i)
5:         end if
6:     end for
7: end for
```

E' corretta? Qual è l'effetto della procedura?

Questa è corretta:

$MatrixGraphInverse(G)$

```
1: for i ← 2 to n do
2:     for j ← 1 to i − 1 do
3:         if A[i, j] ≠ A[j, i] then
4:             Exchange(A, i, j, j, i)
5:         end if
6:     end for
7: end for
```

Quanto costa?