Università di Pisa

NICOLETTA DE FRANCESCO

# Algoritmi e strutture dati

a.a. 2019/2020

# Grafi

**Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione la maggior parte delle slide utilizzate nella presente lezione**

**GRAFO ORIENTATO = (N, A)**

**N = insieme di nodi**

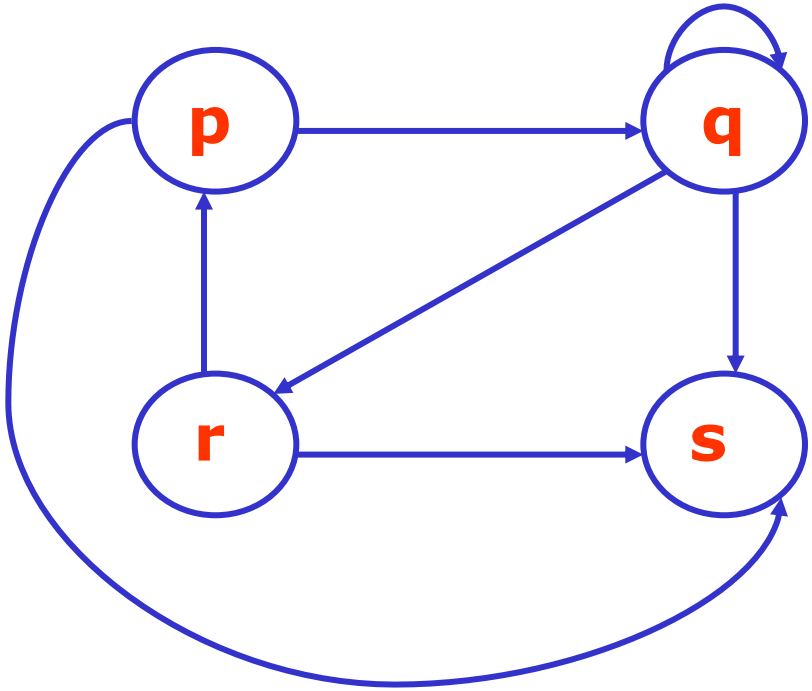
**A $\subseteq$ N x N = insieme di archi**

**(coppie ordinate di nodi)**

- **Se (p,q) $\in$ A, diciamo che p è predecessore di q e q è successore di p.**

**n=|N| numero dei nodi**
**m=|A| numero degli archi.**

**Un grafo orientato con n nodi ha al massimo $n^2$ archi**



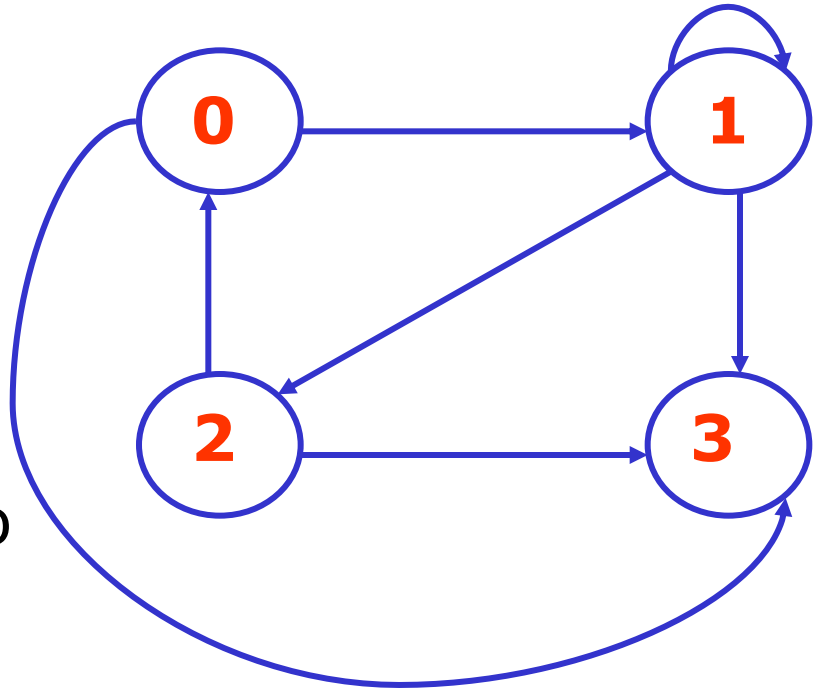
## Cammini e Cicli

Un ***cammino*** è una sequenza di nodi $(n_1, \ldots, n_k)$, $k \geq 1$ tale che esiste un arco da $n_i$ a $n_i+1$ per ogni $1 \leq i < k$.

La lunghezza del cammino è data dal numero degli archi.

Un ***ciclo*** è un cammino che comincia e finisce con lo stesso nodo.

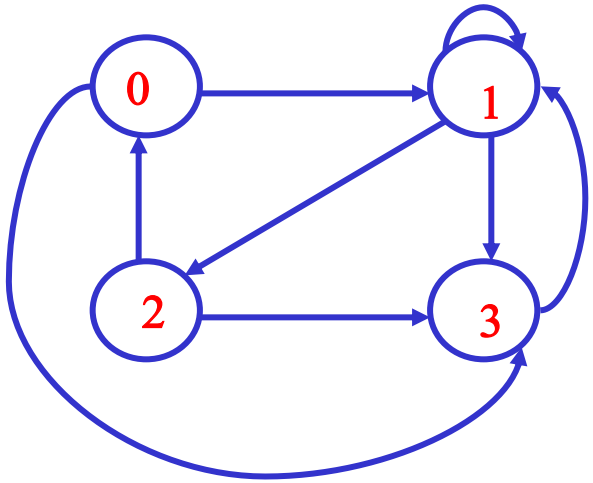
Un grafo è aciclico se non contiene cicli.



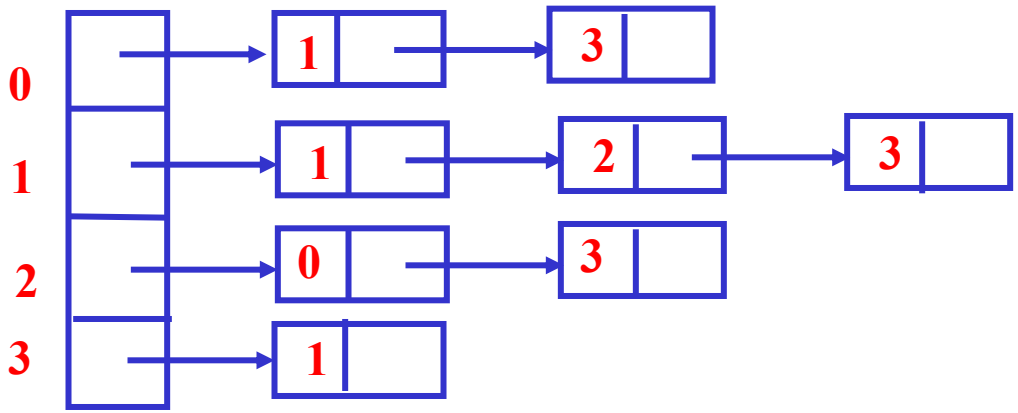
# rappresentazione in memoria dei grafi: liste di adiacenza

```
struct Node{
        int NodeNumber;
        Node * next;
};

Node *graph[N];
```



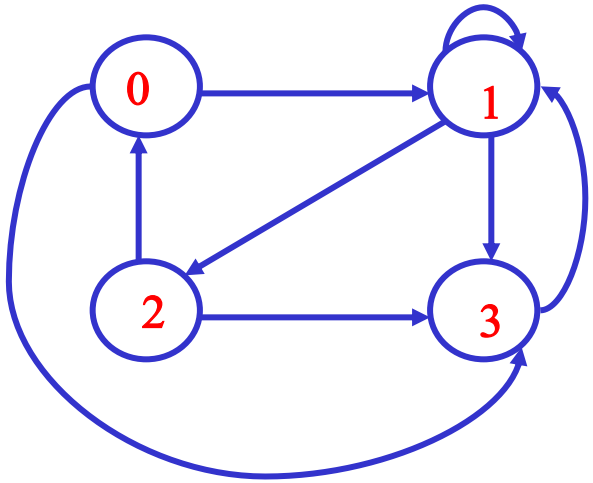

- Viene definito un array con dimensione uguale al numero dei nodi.
- Ogni elemento dell'array rappresenta un nodo con i suoi successori.



# rappresentazione in memoria dei grafi: matrici di adiacenza

**int graph [N][N];**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 |



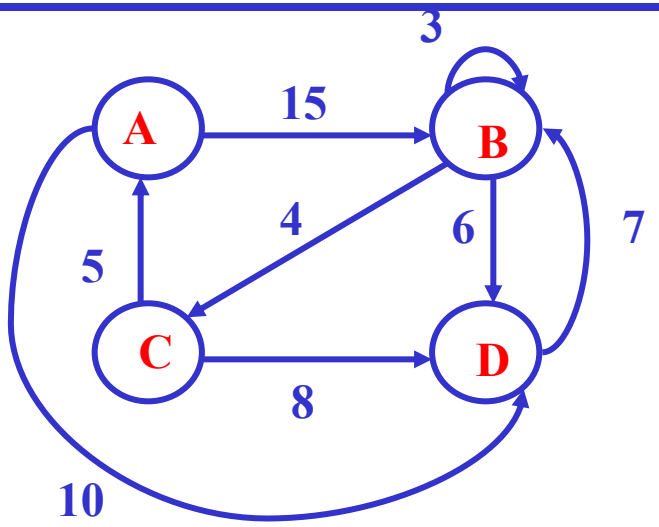

- Nella realizzazione con matrici di adiacenza, il grafo viene rappresentato con una matrice quadrata n × n.
- L'elemento della matrice di indici i, j è 1 se c'è un arco dal nodo i al nodo j e 0 altrimenti.

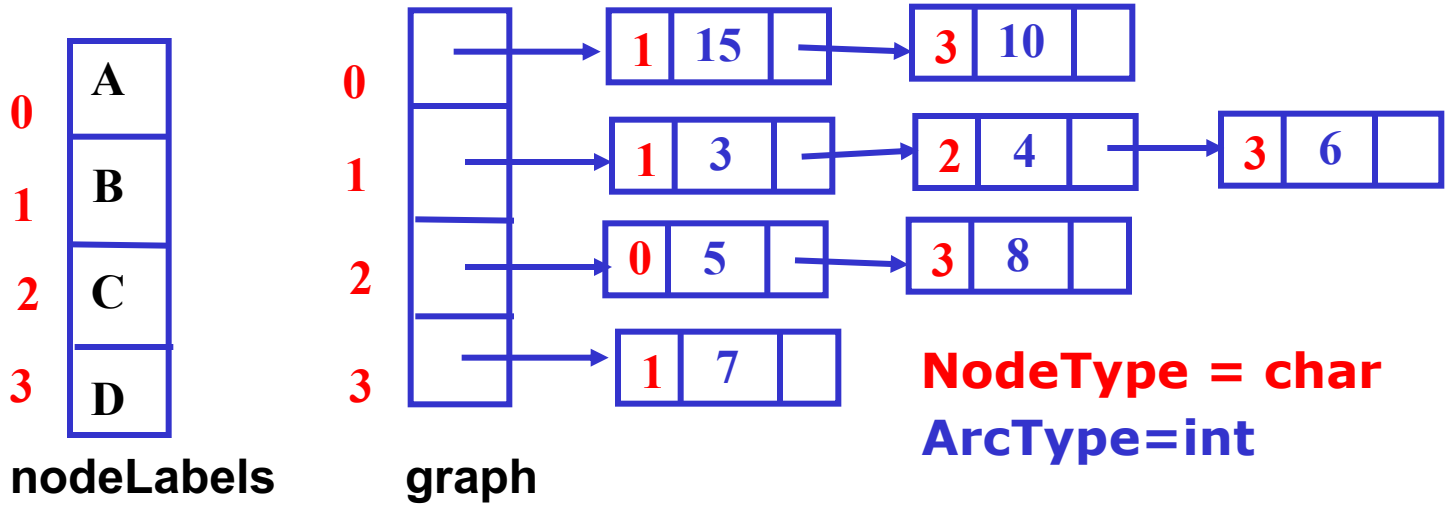
# Grafi con nodi e archi etichettati : Liste di adiacenza

```
struct Node{
        int NodeNumber;
        ArcType arcLabel;
        Node * next;
};

Node * graph[N];

NodeType nodeLabels [N];
```

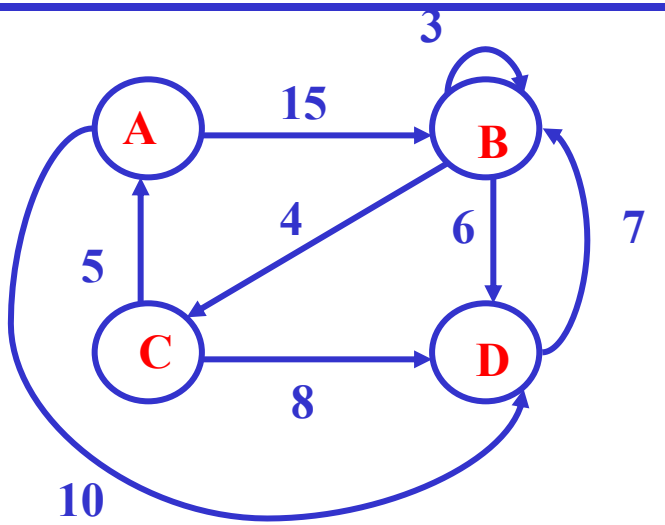


# Con nodi e archi etichettati : matrici di adiacenza

**ArcType** graph [N][N];

**NodeType** nodeLabels [N];




| | nodeLabels |
|---|---|
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | D |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 15 | 0 | 10 |
| 1 | 0 | 3 | 4 | 6 |
| 2 | 5 | 0 | 0 | 8 |
| 3 | 0 | 7 | 0 | 0 |

**NodeType = char**
**ArcType=int**

# visita in profondità

```
void NodeVisit (nodo) {
   esamina il nodo;
   marca il nodo;
   applica NodeVisit ai successori non marcati del nodo;
 }

Void DepthVisit Graph(h) {
          per tutti i nodi:
             se il nodo non è marcato applica nodeVisit;
}
```

Complessità?

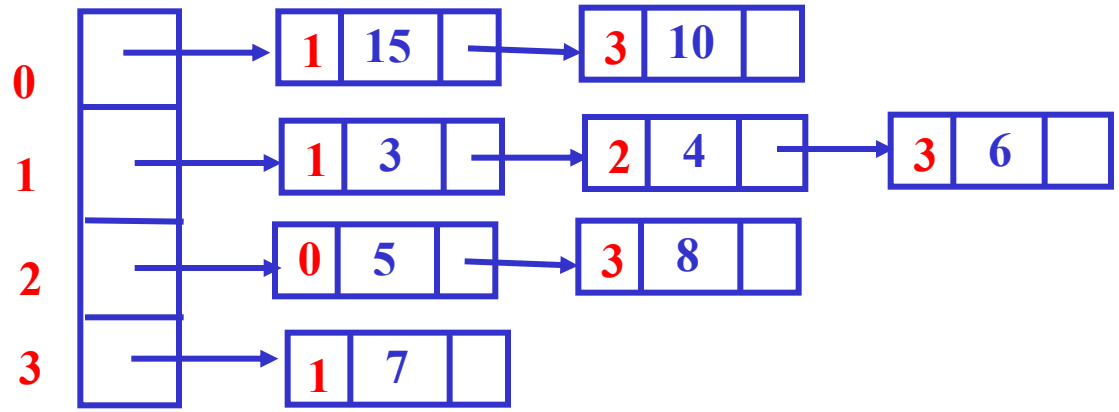
**n=|N| numero dei nodi**
**m=|A| numero degli archi**

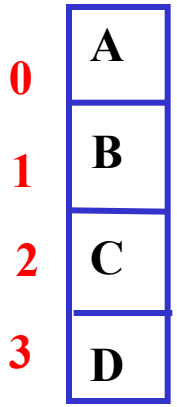
# Una classe per i grafi

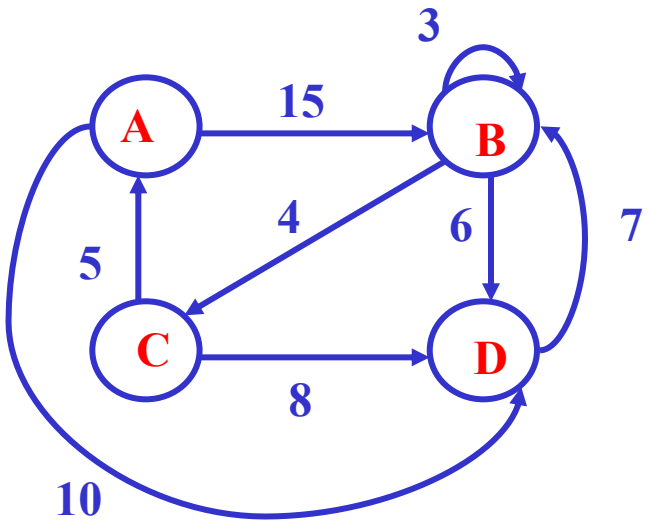
```
class Graph{
 struct Node {
    int nodeNumber;
    Node* next;
 };
 Node* graph [N];
 NodeType nodeLabels [N];
 int mark[N];
 void nodeVisit( int i) {
    mark[i]=1;
    <esamina nodeLabels[i]>;
    Node* g; int j;
    for (g=graph[i]; g; g=g->next){
          j=g->nodeNumber;
          if (!mark[j]) nodeVisit(j);
    }
 }
```

```
public:
 void depthVisit() {
    for (int i=0; i<N; i++)
          mark[i]=0;
    for (i=0; i<N; i++)
          if (! mark[i])
                nodeVisit (i);
}
..
};
```

# visita in profondità: esempio




Visita?

# Grafi non orientati

**grafo non orientato = (N, A),**

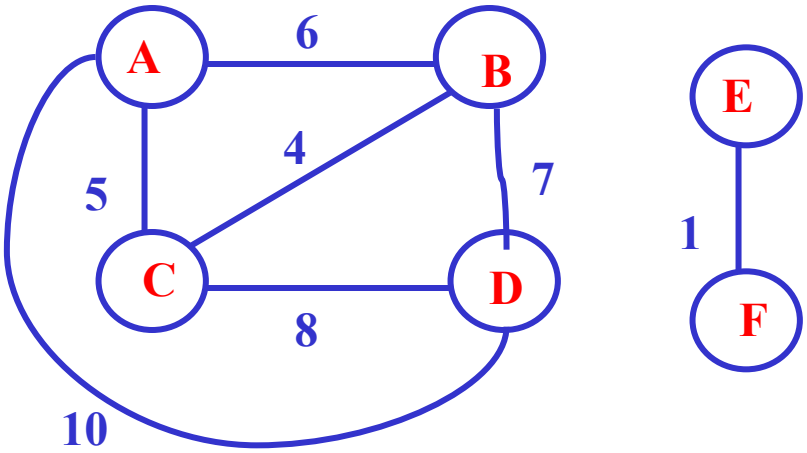
**N = insieme di nodi**

**A = insieme di coppie non ordinate di nodi**

- **Se (p,q) ∈ A, p è diverso da q, diciamo che è adiacente a q e viceversa.**

**Un grafo non orientato con n nodi ha al massimo n(n-1)/2 archi**

# Esempio di grafo non orientato




- Un cammino in un grafo non orientato è una sequenza di nodi $(n_1, \ldots, n_k)$, $k \geq 1$ tale che $n_i$ è adiacente a $n_i+1$ per ogni i.
- Un ciclo è un cammino che inizia e termina con lo stesso nodo e non ha ripetizioni, eccettuato l'ultimo nodo.
- Un grafo non orientato è connesso se esiste un cammino fra due nodi qualsiasi del grafo.

## Rappresentazione in memoria dei grafi non orientati

Un grafo non orientato può essere visto come un grafo orientato tale che, per ogni arco da un nodo p a un nodo q, ne esiste uno da q a p.

La rappresentazione in memoria dei grafi non orientati può essere fatta con le matrici o con le liste di adiacenza tenendo conto di questa equivalenza.

Naturalmente ogni arco del grafo non orientato sarà rappresentato due volte (la matrice di adiacenza è sempre simmetrica).

# Multi-grafi orientati

**Multi-grafo non orientato = (N, A),**

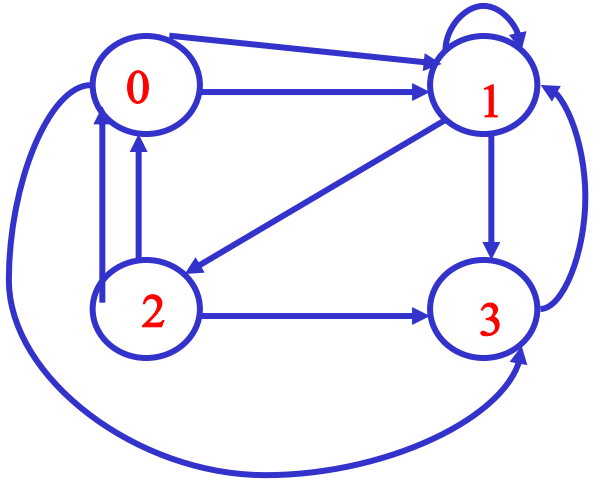
**N = insieme di nodi**
**A = multi-insieme di coppie non ordinate di nodi**

**Non c'è relazione fra il numero di nodi e il numero di archi**

**Analogamente si definiscono i multi-grafi non orientati**

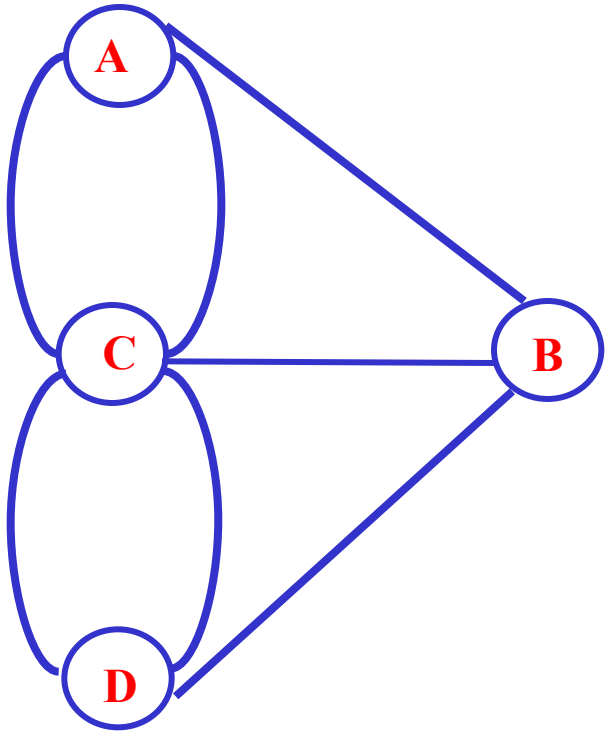
# Multi-grafi orientati : matrice di adiacenza

**int graph [N][N];**

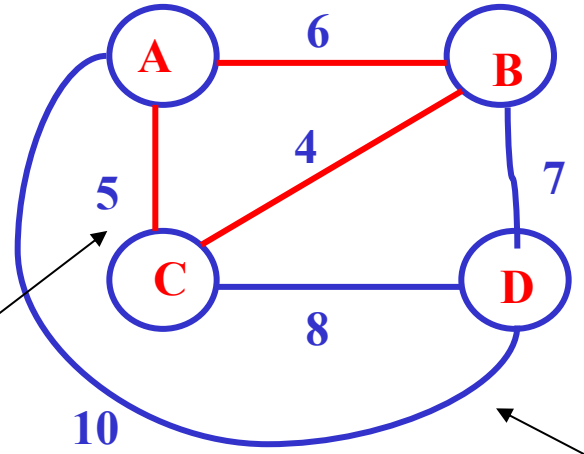


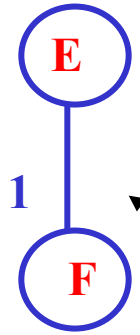

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 2 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 2 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 |

# Esempio di multi-grafo non orientato



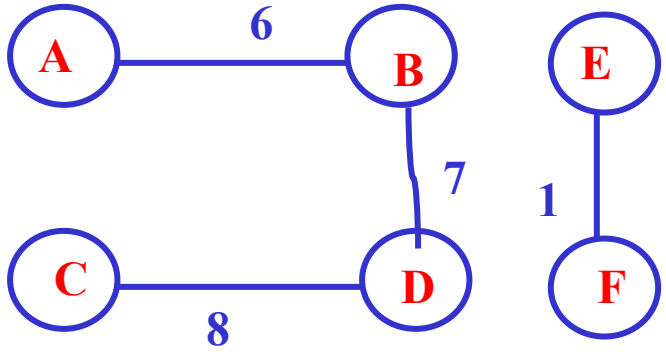
# Minimo albero di copertura



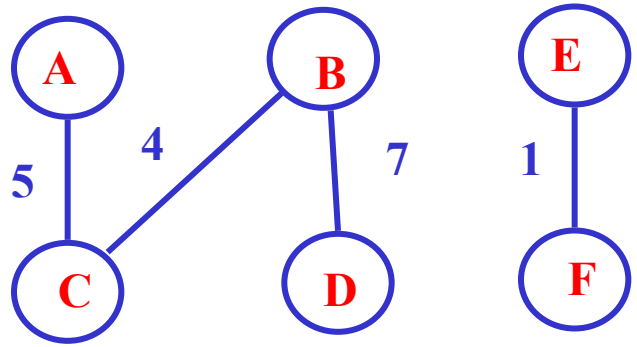




Componente connessa

Componenti connesse massimali




Albero di copertura




Minimo albero di copertura

# Minimo albero di copertura

- **Un grafo non orientato è connesso se esiste un cammino fra due nodi qualsiasi del grafo**

- **Componente connessa: sottografo connesso**

- **Componente connessa massimale: nessun nodo è connesso ad un'altra componente connessa**

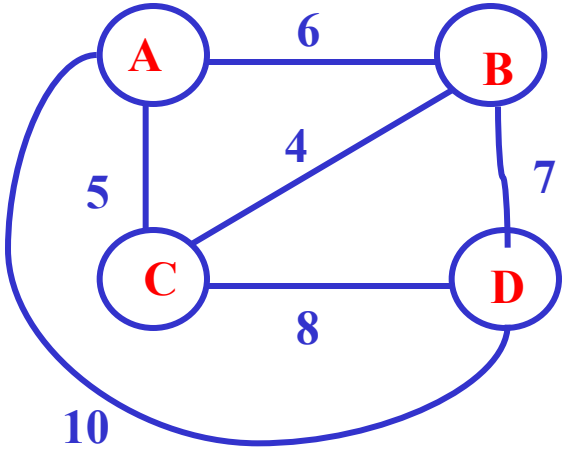
- **Albero di copertura: insieme di componenti connesse massimali acicliche**

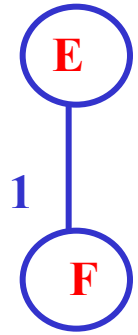
- **Minimo albero di copertura: la somma dei pesi degli archi è minima**

## algoritmo di Kruskal per trovare il minimo albero di coperura

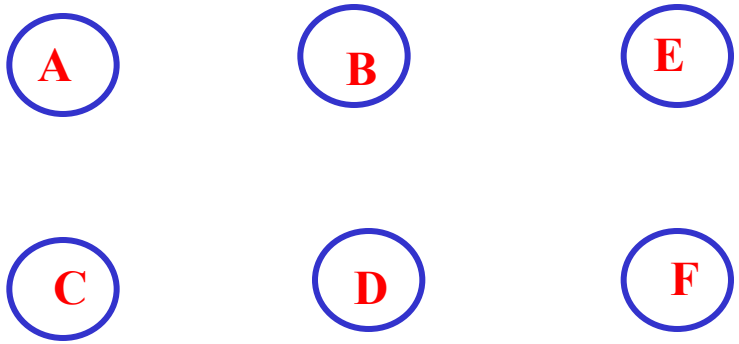
1. **Ordina** gli archi del grafo in ordine crescente

2. **Scorri** l'elenco ordinato degli archi:
   per ogni arco a
   ```
   if (a connette due componenti non connesse) {
      scegli a;
      unifica le componenti;
   }
   ```

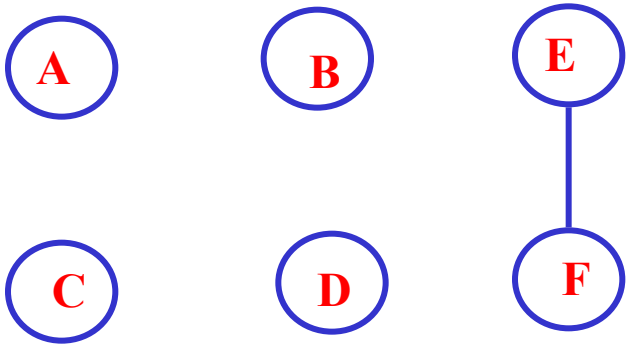



(E,F) (B,C) (A,C) (A,B) (D,B) (C,D) (A,D)

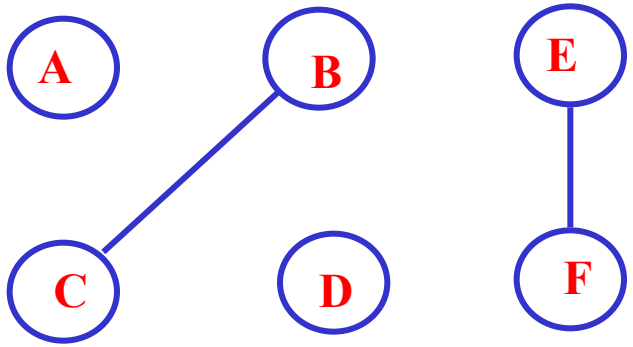



{A} {B} {C} {D} {E} {F}



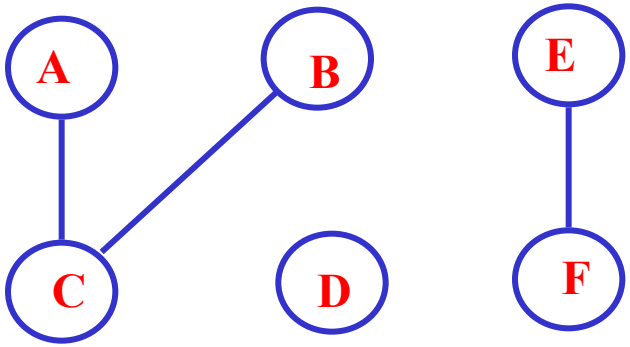

(E,F) (B,C) (A,C) (A,B) (D,B) (C,D) (A,D)

{A} {B} {C} {D} {E, F}



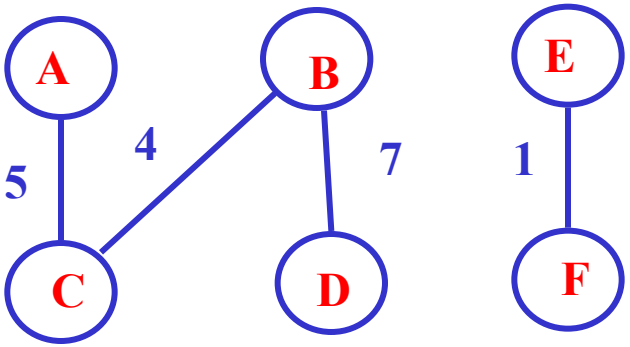

(E,F) (B,C) (A,C) (A,B) (D,B) (C,D) (A,D)

{A} {B, C} {D} {E, F}




(E,F) (B,C) (A,C) (A,B) (D,B) (C,D) (A,D)

{A, B, C} {D} {E, F}




(E,F) (B,C) (A,C) (A,B) (D,B) (C,D) (A,D)

{A, B, C, D} {E, F}

**Lunghezza: 17**

# Bibliografia

Demetrescu:

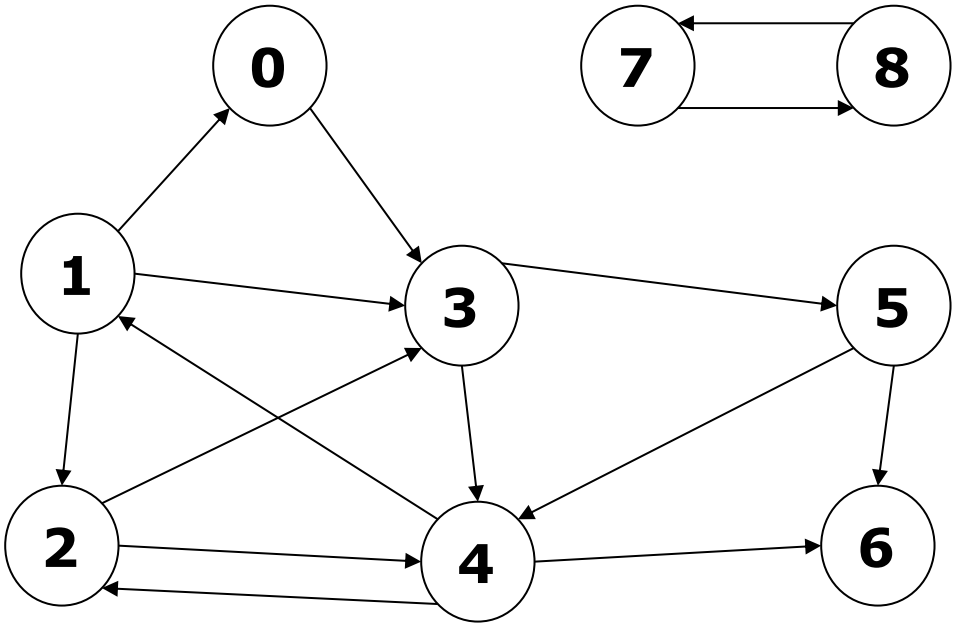
Paragrafo
Cormen:

Capitolo

# Esercizio 1

**Indicare la sequenza di nodi ottenuta visitando in profondita' il grafo orientato seguente memorizzato con liste di adiacenza.**

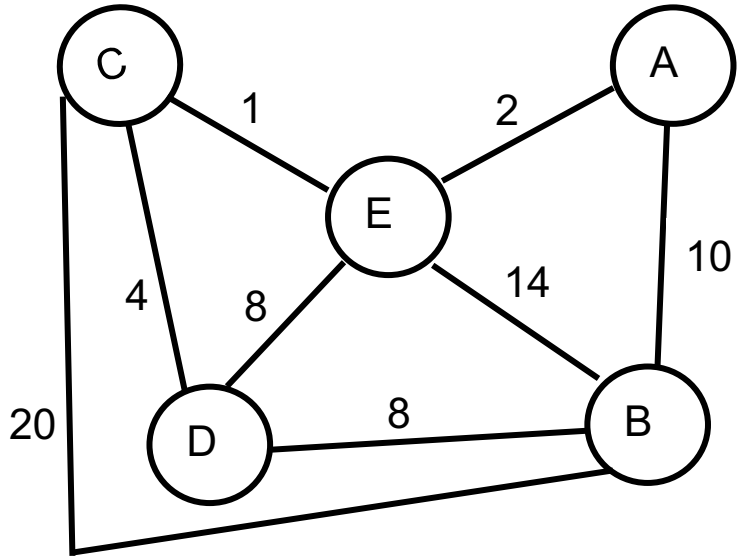
**Supporre che nelle liste di adiacenza i nodi compaiano in ordine crescente.**




**0: 3**
**1: 0 -> 2-> 3**
**2: 3->4**
**3: 4->5**
**4: 1-> 2-> 6**
**5: 4->6**
**6:**
**7: 8**
**8: 7**

# Esercizio 2

**Trovare il minimo albero di copertura del grafo seguente con l'algoritmo di Kruskal, indicando le component connesse ad ogni passaggio**




| Arco | Componenti connesse |
|---|---|
| | |
| | |
| | |
| | |
| | |

**Dimensione: ?**