Lezione 23

Sommario della Lezione

- Esplorazione di grafi: Visita in profondità

Sommario della Lezione

- Esplorazione di grafi: Visita in profondità
  - Proprietà
  - Applicazioni

Visita in ampiezza (Breadth First) di un grafo G

Visita in ampiezza (Breadth First) di un grafo G

- Esplora il grafo G per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente) s

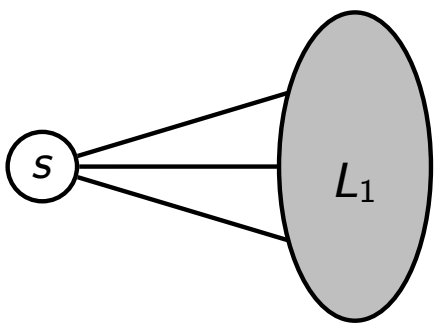
Visita in ampiezza (Breadth First) di un grafo G

- Esplora il grafo G per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente) s
- Calcola la distanza (minimo numero di archi) da s ad ognuno dei vertici raggiungibili da s in G

Visita in ampiezza (Breadth First) di un grafo G

- Esplora il grafo G per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente) s
- Calcola la distanza (minimo numero di archi) da s ad ognuno dei vertici raggiungibili da s in G

Intuizione: Il grafo G viene esplorato scoprendo tutti i vertici a distanza k (livello k) prima di cominciare a scoprire quelli a distanza $k+1$, per $k = 1, 2, \ldots$

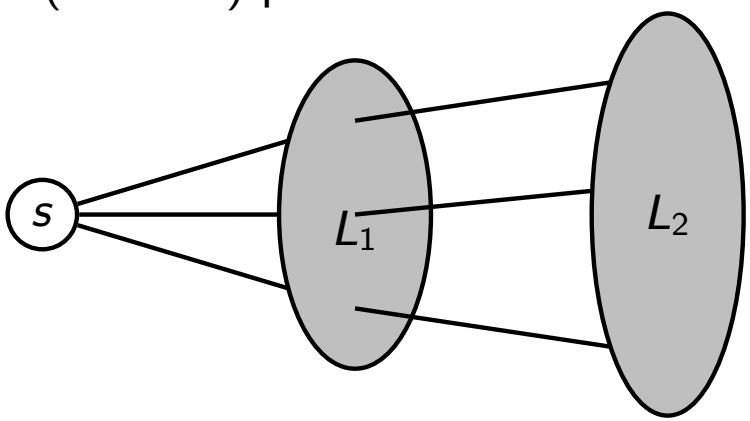


nodi a distanza 1 da s

Visita in ampiezza (Breadth First) di un grafo G

- Esplora il grafo G per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente) s
- Calcola la distanza (minimo numero di archi) da s ad ognuno dei vertici raggiungibili da s in G

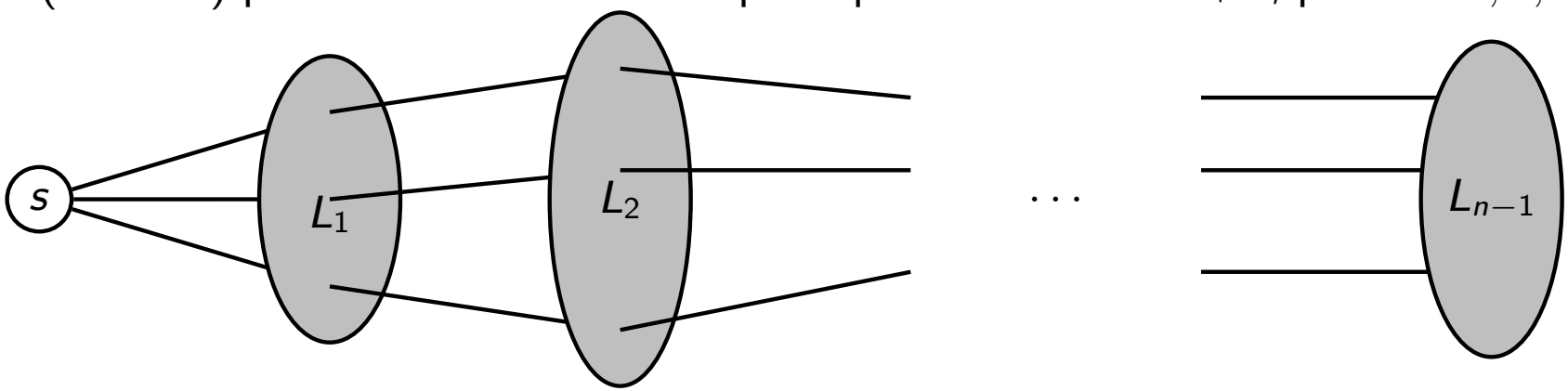
Intuizione: Il grafo G viene esplorato scoprendo tutti i vertici a distanza k (livello k) prima di cominciare a scoprire quelli a distanza $k+1$, per $k = 1, 2, \ldots$



Visita in ampiezza (Breadth First) di un grafo G

- Esplora il grafo G per scoprire tutti i vertici raggiungibili dal vertice di partenza (sorgente) s
- Calcola la distanza (minimo numero di archi) da s ad ognuno dei vertici raggiungibili da s in G

Intuizione: Il grafo G viene esplorato scoprendo tutti i vertici a distanza k (livello k) prima di cominciare a scoprire quelli a distanza $k+1$, per $k = 1, 2, \ldots$



BFS: l'algoritmo

`BFS`(G, s)
1. $L_0 = \{s\}$

BFS: l'algoritmo

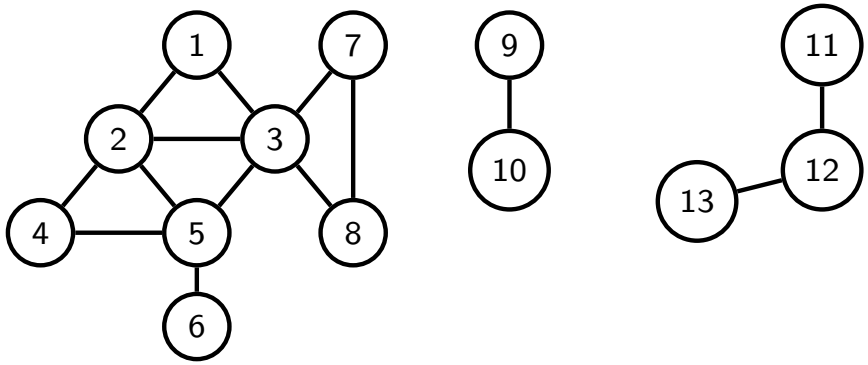
`BFS`(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s

BFS: l'algoritmo

BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i

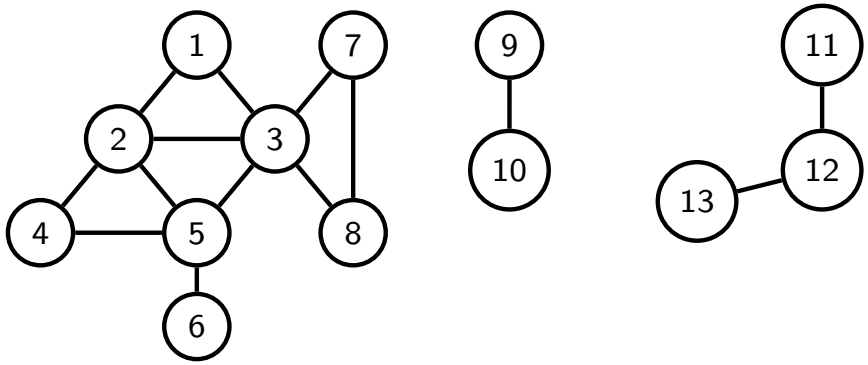
BFS: l'algoritmo

BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i



BFS: l'algoritmo

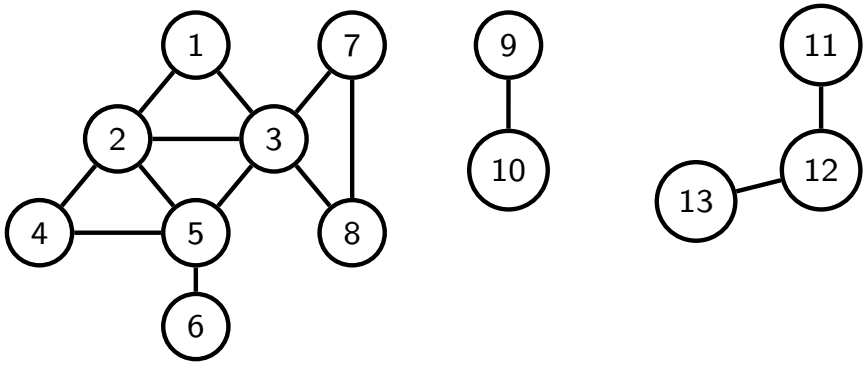
BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i



L_0 1

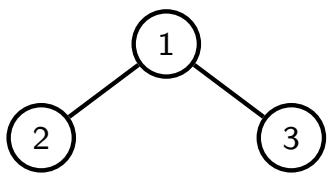
BFS: l'algoritmo

BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i



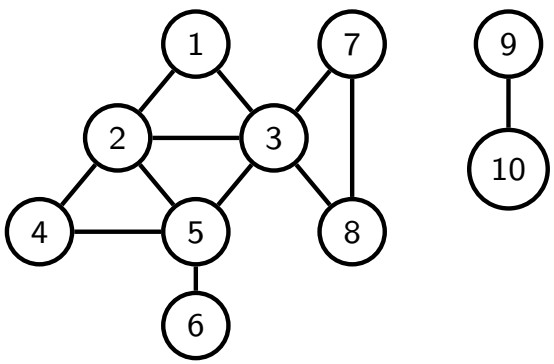
L_0

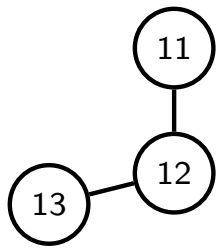
L_1



BFS: l'algoritmo

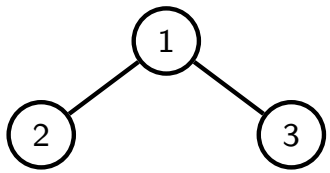
BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i



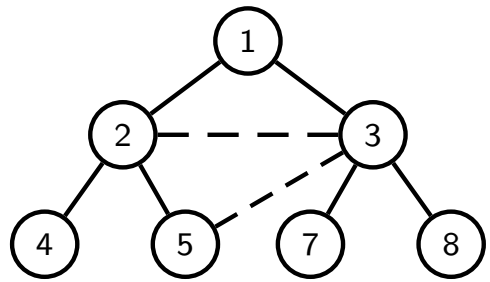


L_0

L_1

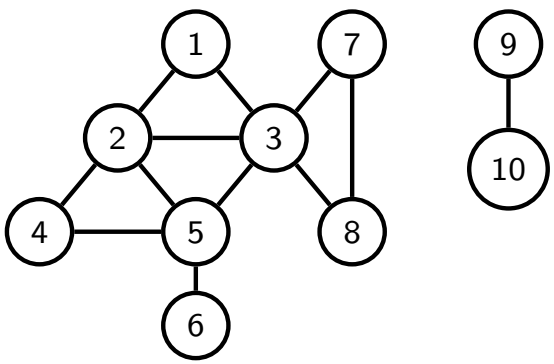


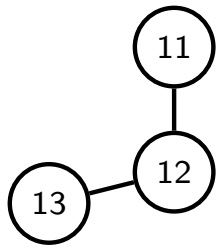
L_2



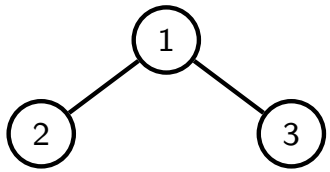
BFS: l'algoritmo

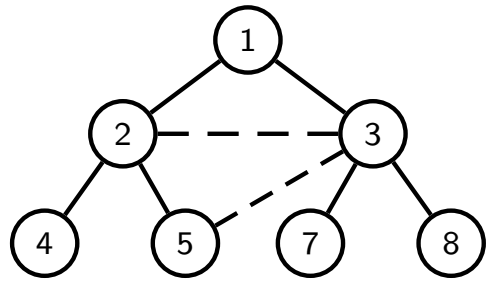
BFS(G, s)
1. $L_0 = \{s\}$
2. $L_1 =$ tutti i vicini di s
3. $\forall i \geq 1$ $L_{i+1} =$tutti i nodi che **non** appartengono ad un livello L_j precedente $(j \leq i)$ e che sono connessi con un arco a qualche nodo in L_i

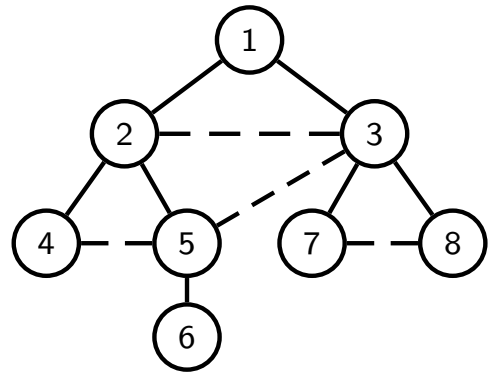




L_0
L_1
L_2
L_3







Proprietà della BFS:

1. L'albero T prodotto dalla $\texttt{BFS}(G, s)$ contiene **tutti e solo i nodi** u del grafo G che sono raggiungibili mediante un cammino dal nodo sorgente s.

Proprietà della BFS:

1. L'albero T prodotto dalla $\texttt{BFS}(G, s)$ contiene **tutti e solo i nodi** u del grafo G che sono raggiungibili mediante un cammino dal nodo sorgente s.
2. Nell'albero T prodotto dalla $\texttt{BFS}(G, s)$, ogni percorso dalla radice s ad un generico nodo u è un percorso di lunghezza **minima** da s ad u nel grafo G.

Proprietà della BFS:

1. L'albero T prodotto dalla $\texttt{BFS}(G, s)$ contiene **tutti e solo i nodi** u del grafo G che sono raggiungibili mediante un cammino dal nodo sorgente s.
2. Nell'albero T prodotto dalla $\texttt{BFS}(G, s)$, ogni percorso dalla radice s ad un generico nodo u è un percorso di lunghezza **minima** da s ad u nel grafo G.
3. La $\texttt{BFS}(G, s)$ permette di scoprire se il grafo G contiene o meno un ciclo, basta verificare se si visita di nuovo un nodo **dopo** averlo scoperto per la prima volta

Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

- $u \in C(u)$

Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

- $u \in C(u)$
- per ogni coppia di vertici $a, b \in C(u)$, esiste in G un cammino che parte in a ed ha come vertice terminale b.

Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

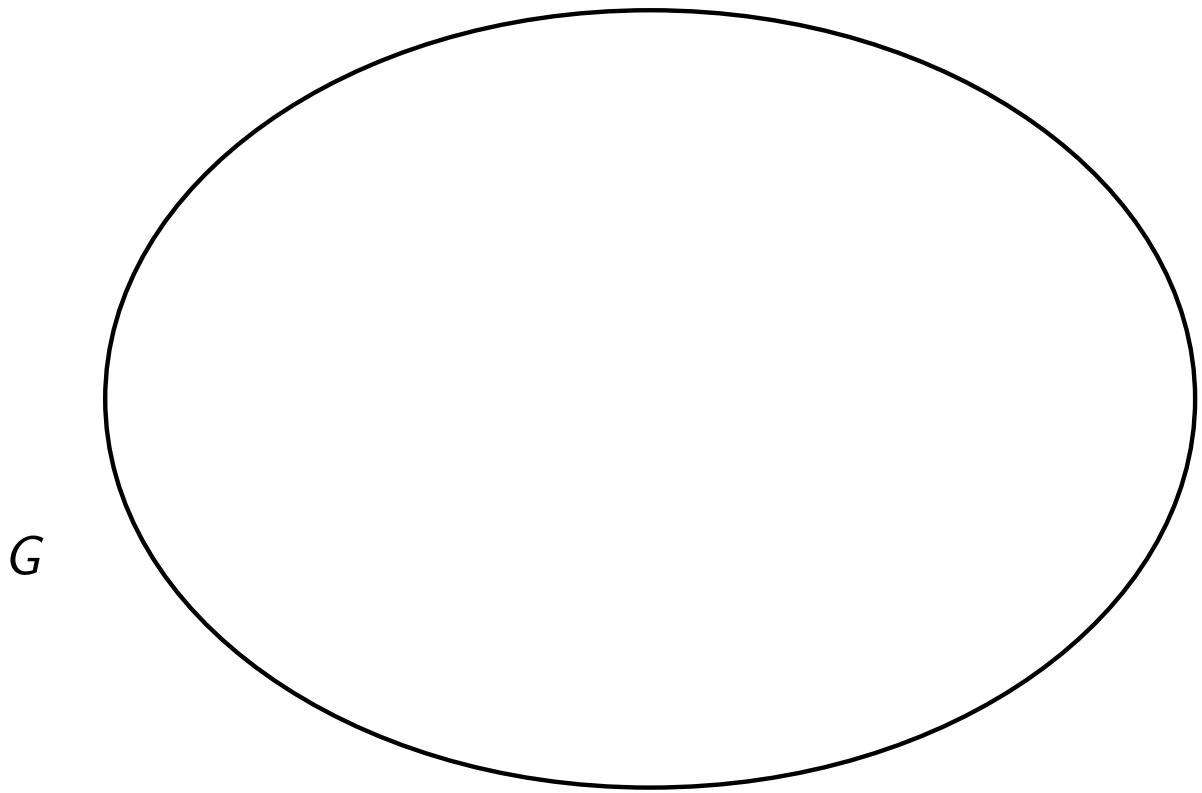
- $u \in C(u)$
- per ogni coppia di vertici $a, b \in C(u)$, esiste in G un cammino che parte in a ed ha come vertice terminale b.

Il sottoinsieme dei vertici $C(u)$ viene chiamata *componente connessa* di u.

Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

- $u \in C(u)$
- per ogni coppia di vertici $a, b \in C(u)$, esiste in G un cammino che parte in a ed ha come vertice terminale b.

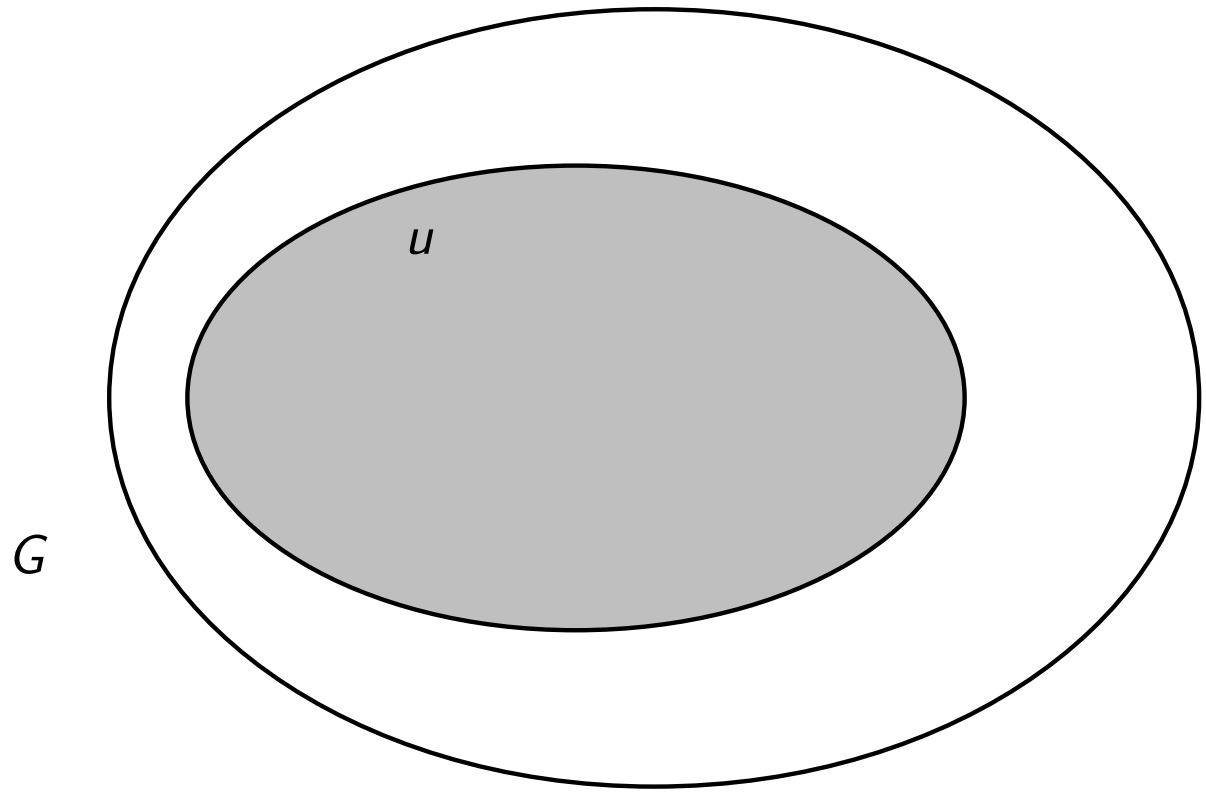
Il sottoinsieme dei vertici $C(u)$ viene chiamata *componente connessa* di u.



Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

- $u \in C(u)$
- per ogni coppia di vertici $a, b \in C(u)$, esiste in G un cammino che parte in a ed ha come vertice terminale b.

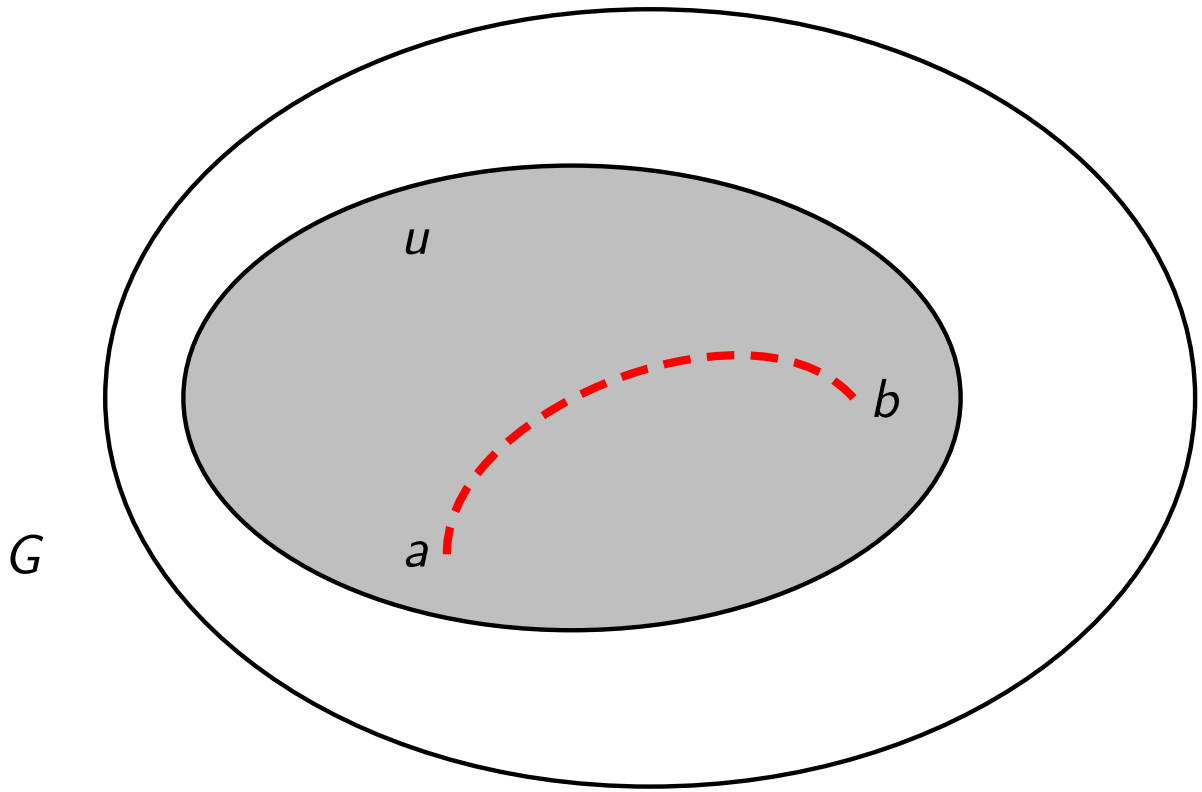
Il sottoinsieme dei vertici $C(u)$ viene chiamata *componente connessa* di u.



Per ogni nodo $u \in V$ del grafo $G = (V, E)$, sia $C(u)$ il **più grande** sottoinsieme dei vertici V tale che

- $u \in C(u)$
- per ogni coppia di vertici $a, b \in C(u)$, esiste in G un cammino che parte in a ed ha come vertice terminale b.

Il sottoinsieme dei vertici $C(u)$ viene chiamata *componente connessa* di u.



L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s e poi dalla radice a b,

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s e poi dalla radice a b, ed eliminare vertici eventualmente ripetuti).

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

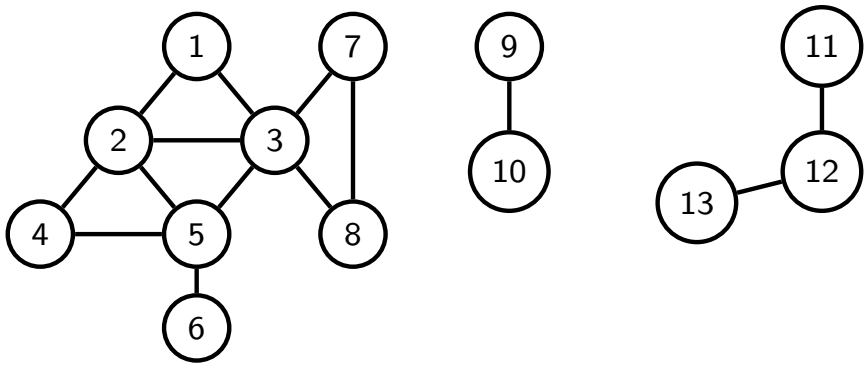
Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s e poi dalla radice a b, ed eliminare vertici eventualmente ripetuti).

Inoltre, è chiaramente il più grande insieme che contiene s con la proprietà di sopra.

L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s e poi dalla radice a b, ed eliminare vertici eventualmente ripetuti).

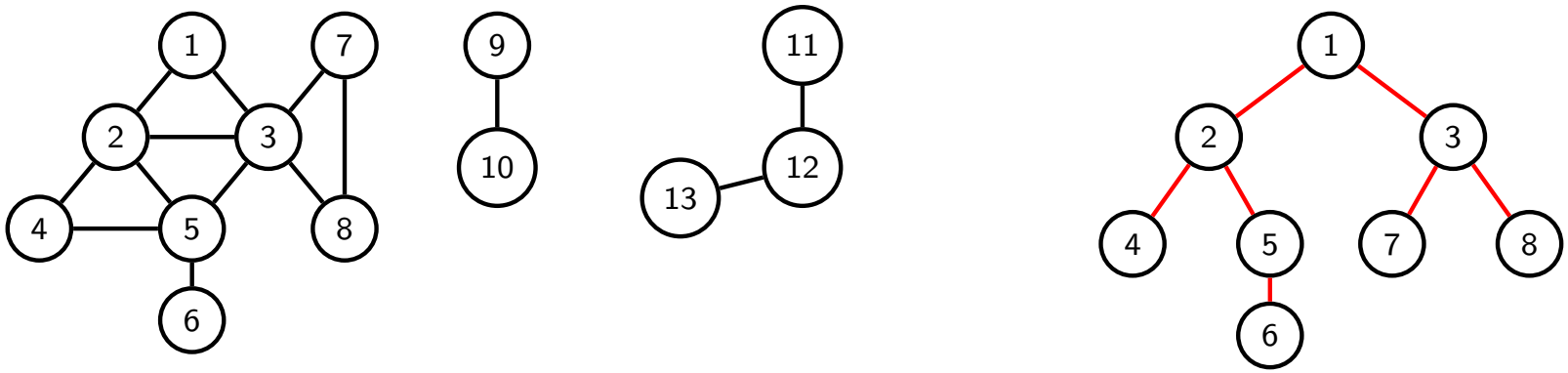
Inoltre, è chiaramente il più grande insieme che contiene s con la proprietà di sopra. Infatti, per costruzione, la BFS termina quanco non ci sono più vertice in G raggiungibili da s.



L'insieme dei vertici dell'albero T prodotto dalla BFS quando essa viene chiamata a partire dal vertice sorgente S è proprio la componente connessa $C(s)$ di s.

Infatti, per ogni coppia di vertici $a, b \in C(s)$ esiste in G un cammino che parte in a ed ha come vertice terminale b (basta andare da a alla radice s e poi dalla radice a b, ed eliminare vertici eventualmente ripetuti).

Inoltre, è chiaramente il più grande insieme che contiene s con la proprietà di sopra. Infatti, per costruzione, la BFS termina quanco non ci sono più vertice in G raggiungibili da s.



Il metodo di visita in profondità (Depth First) procede differentemente.

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato

Il metodo di visita in profondità (Depth First) procede differentemente.

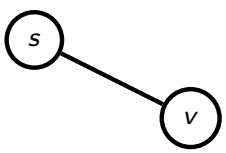
- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...

Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...

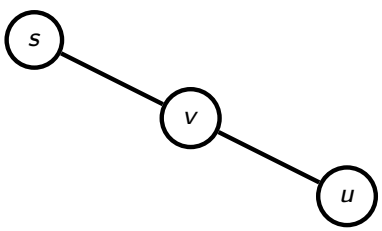
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



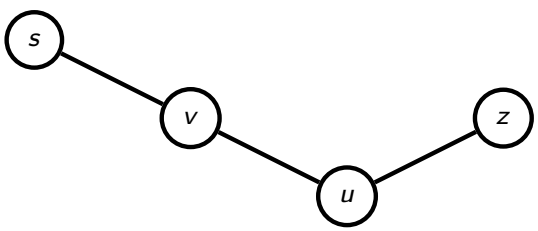
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



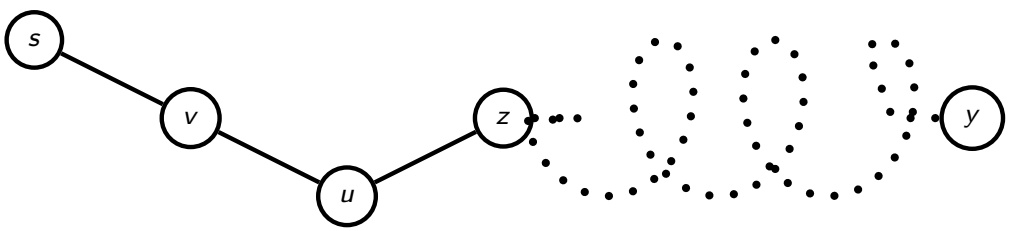
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



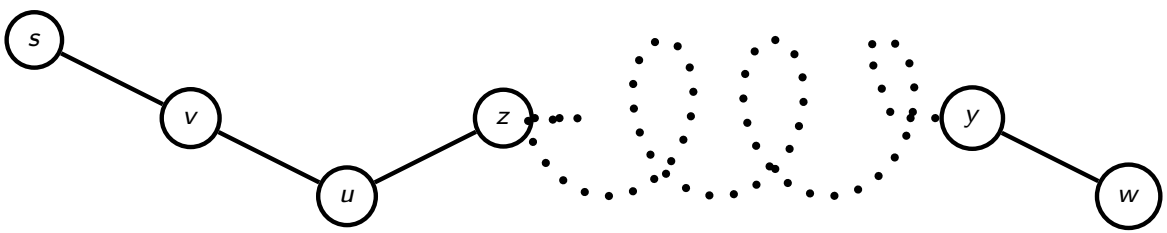
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



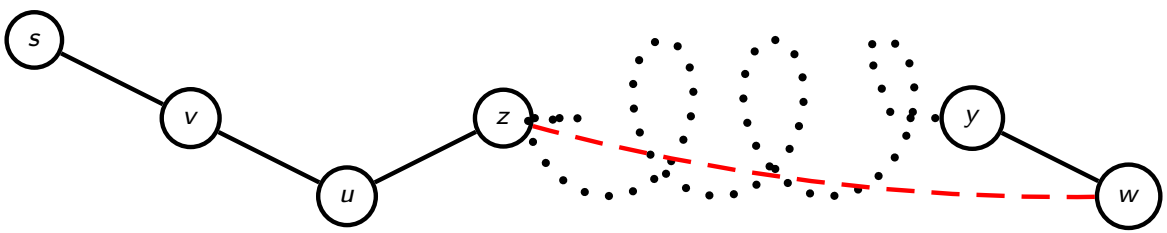
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



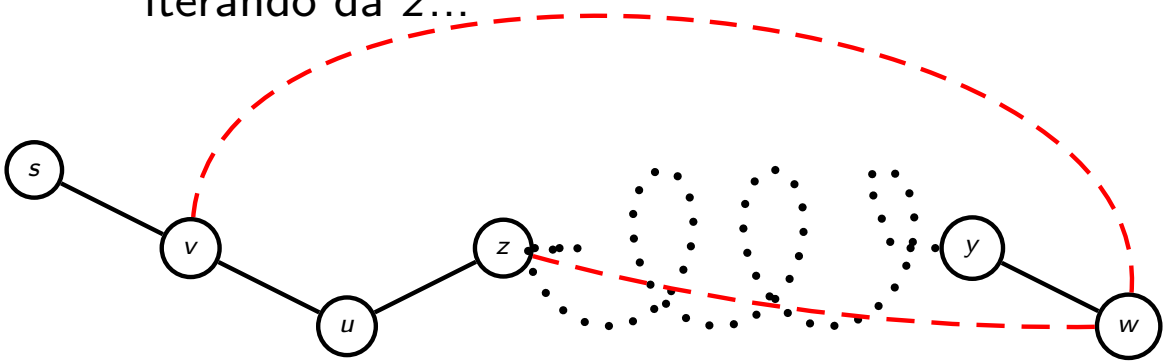
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



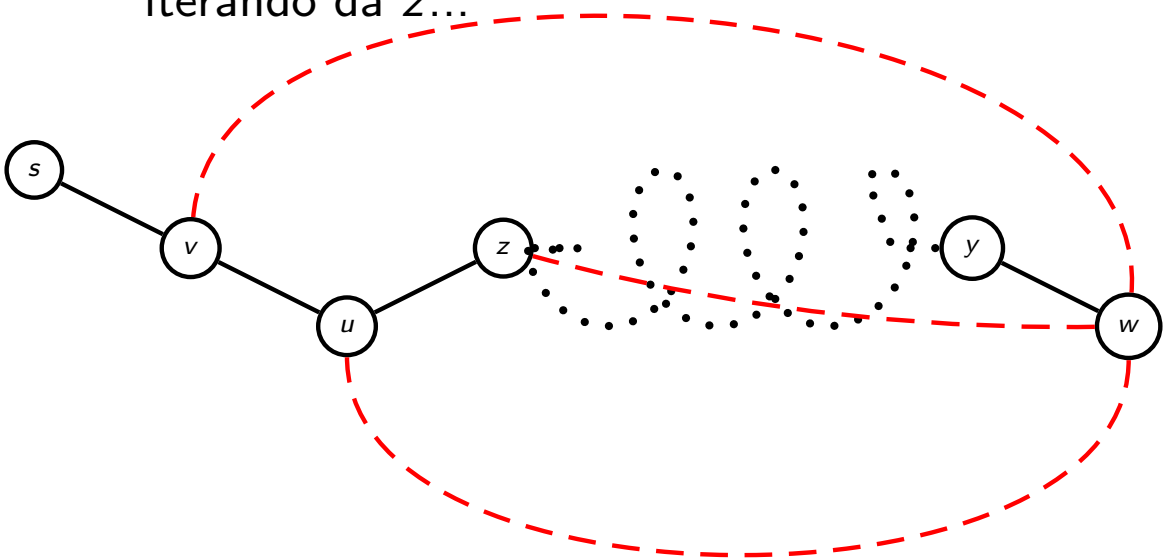
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



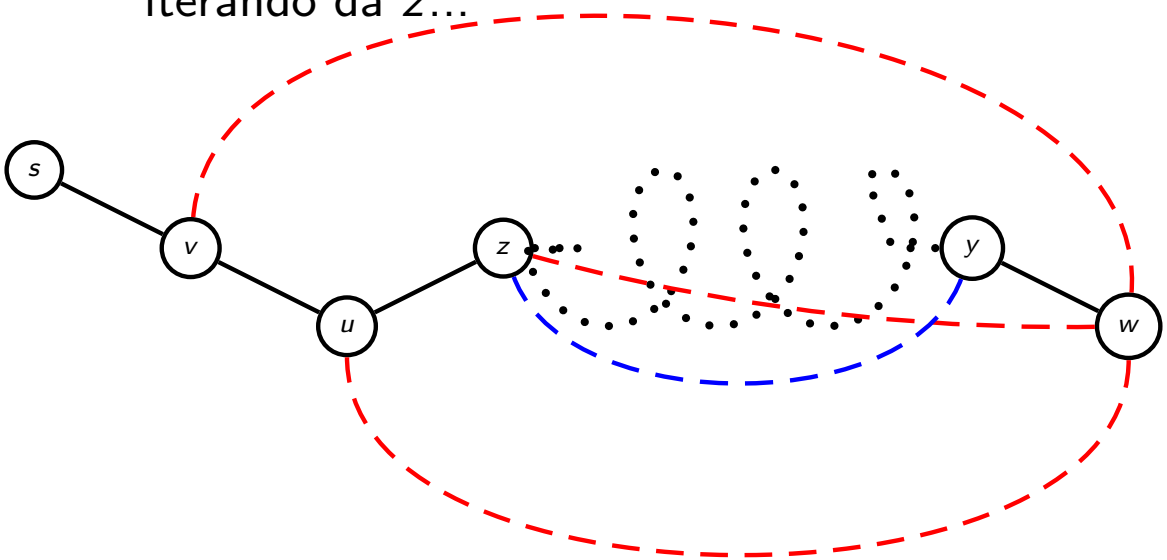
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



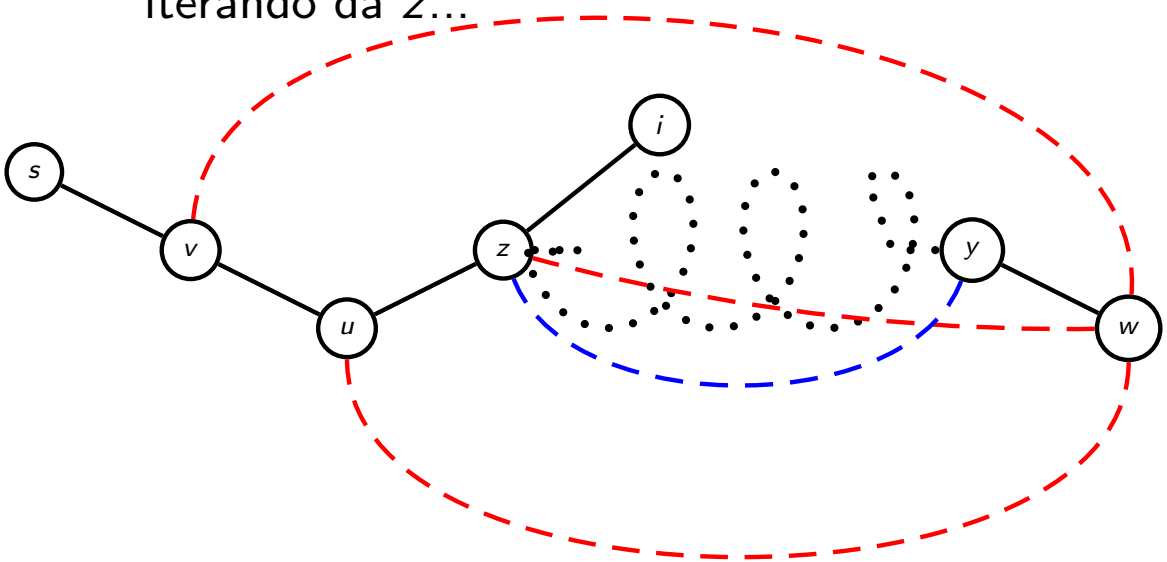
Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



Il metodo di visita in profondità (Depth First) procede differentemente.

- Partendo da un nodo s si considera il primo arco uscente da s, sia esso (s, v) e si raggiunge un nodo v
- Dal nodo v si si considera il primo arco uscente da v per raggiungere un nuovo nodo u e così via, iterando da u...
- Ciò fin quando non si raggiunge un "vicolo cieco", ovvero un nodo w di cui si sono stati esplorati *tutti* i vicini (cosicchè da w non è possibile raggiungere alcun nuovo nodo).
- In tale caso, si torna indietro sul percorso fatto, fin quando non si incontra un qualche nodo z che ha invece almeno un vicino inesplorato e così via, iterando da z...



L'algoritmo `DFS`:

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
```

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
```

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
```

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS)

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T
- ogni qualvolta chiamiamo DFS(v) durante l'esecuzione di DFS(u) (cioè esploriamo v per la prima volta provenendo da u)

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T
- ogni qualvolta chiamiamo DFS(v) durante l'esecuzione di DFS(u) (cioè esploriamo v per la prima volta provenendo da u) allora inseriamo l'arco (u, v) in T

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T
- ogni qualvolta chiamiamo DFS(v) durante l'esecuzione di DFS(u) (cioè esploriamo v per la prima volta provenendo da u) allora inseriamo l'arco (u, v) in T

La complessità di DFS(u) su di un grafo con n vertici e m archi è $\Theta(n + m)$,

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T
- ogni qualvolta chiamiamo DFS(v) durante l'esecuzione di DFS(u) (cioè esploriamo v per la prima volta provenendo da u) allora inseriamo l'arco (u, v) in T

La complessità di DFS(u) su di un grafo con n vertici e m archi è $\Theta(n + m)$, dato che DFS(u) visita ogni nodo al più una volta

L'algoritmo DFS:

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

L'algoritmo DFS produce naturalmente un albero T (che chiameremo albero DFS) con

- s radice di T
- ogni qualvolta chiamiamo DFS(v) durante l'esecuzione di DFS(u) (cioè esploriamo v per la prima volta provenendo da u) allora inseriamo l'arco (u, v) in T

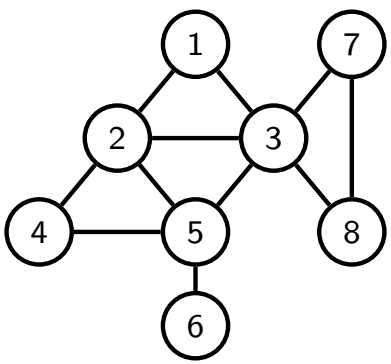
La complessità di DFS(u) su di un grafo con n vertici e m archi è $\Theta(n + m)$, dato che DFS(u) visita ogni nodo al più una volta ed ogni arco al più due volte.

Esempio di esecuzione di DFS

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

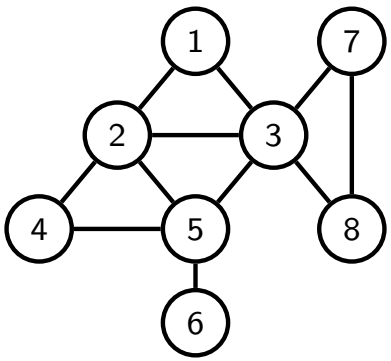
Esempio di esecuzione di DFS

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```



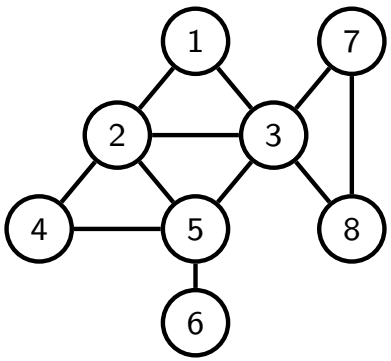
Esempio di esecuzione di DFS

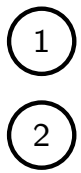
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```



Esempio di esecuzione di DFS

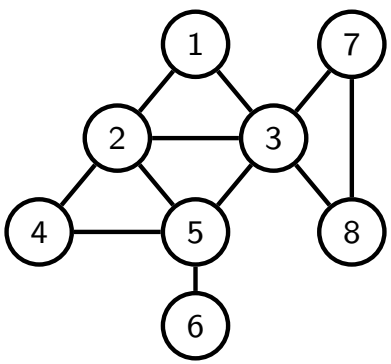
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

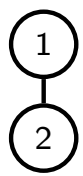




Esempio di esecuzione di DFS

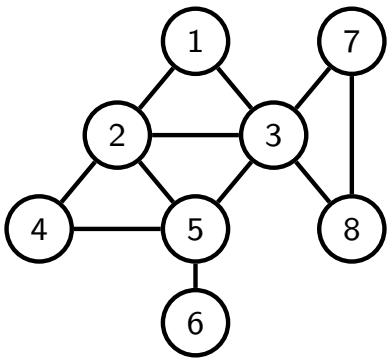
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```





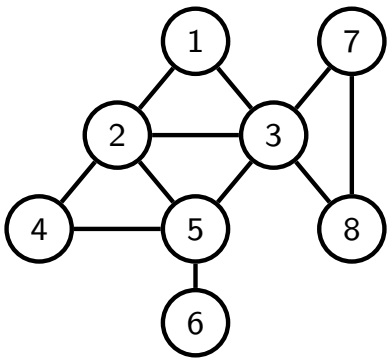
Esempio di esecuzione di DFS

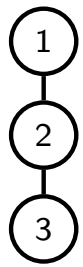
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```



Esempio di esecuzione di DFS

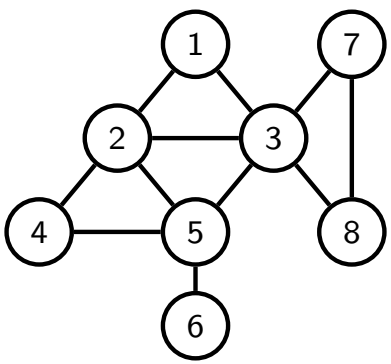
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

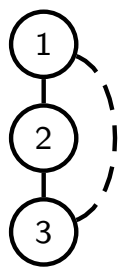




Esempio di esecuzione di DFS

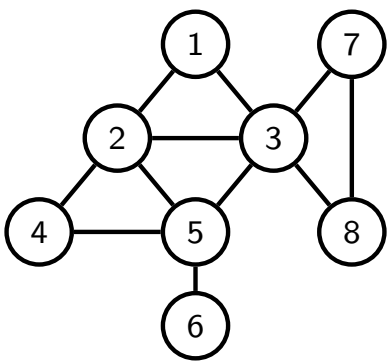
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

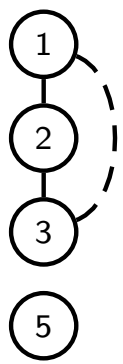




Esempio di esecuzione di DFS

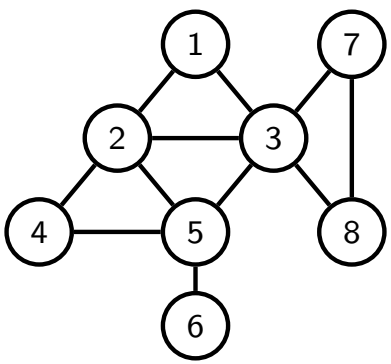
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

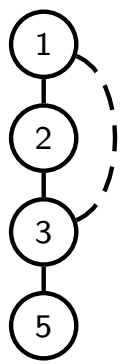




Esempio di esecuzione di DFS

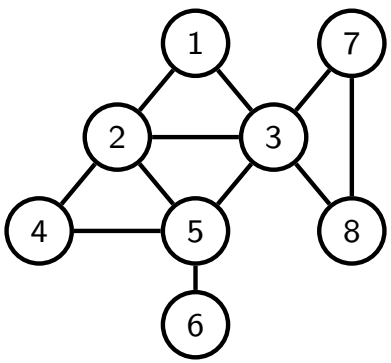
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

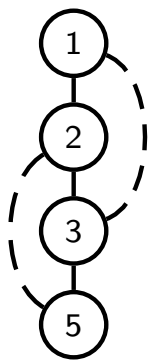




Esempio di esecuzione di DFS

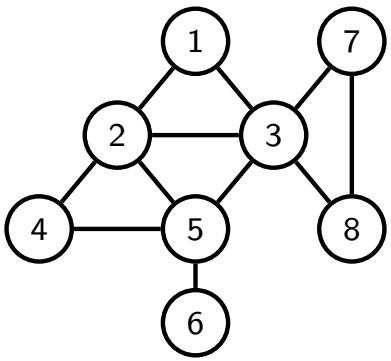
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

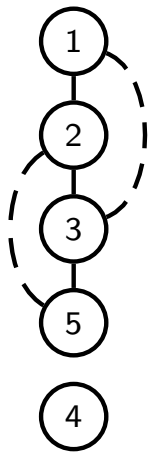




Esempio di esecuzione di DFS

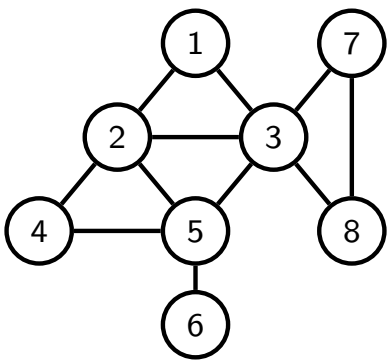
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

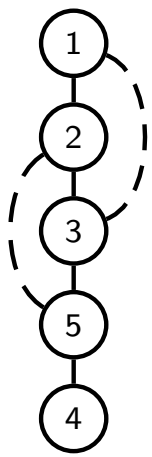




Esempio di esecuzione di DFS

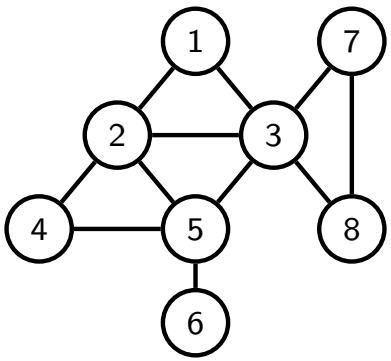
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

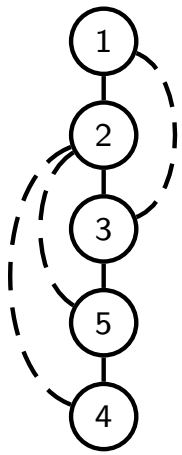




Esempio di esecuzione di DFS

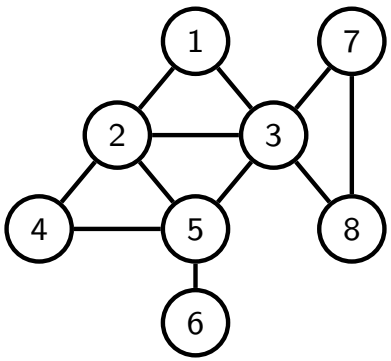
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

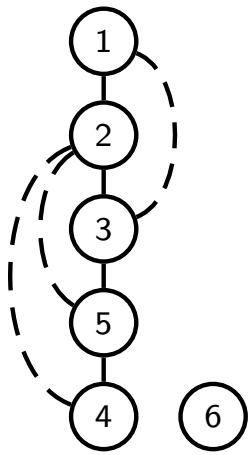




Esempio di esecuzione di DFS

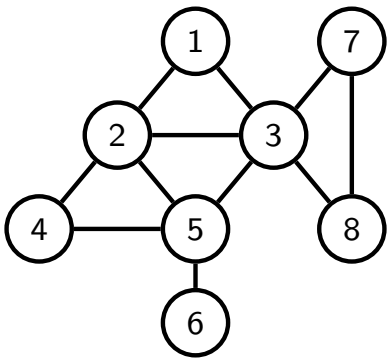
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

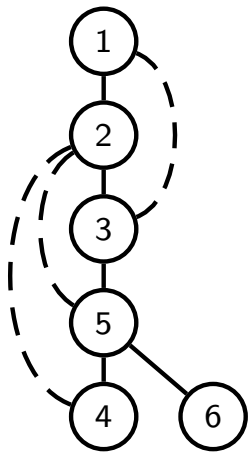




Esempio di esecuzione di DFS

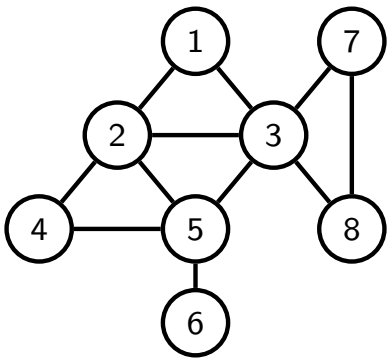
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

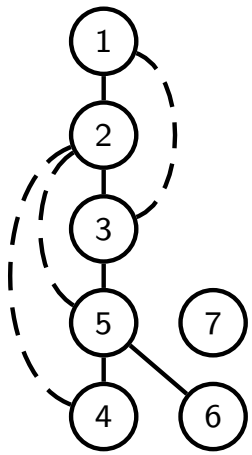




Esempio di esecuzione di DFS

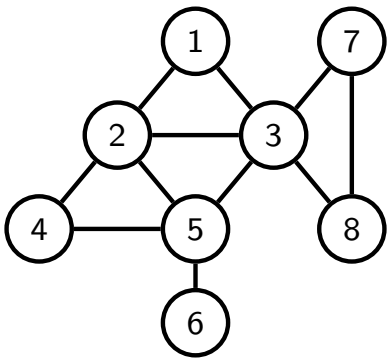
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

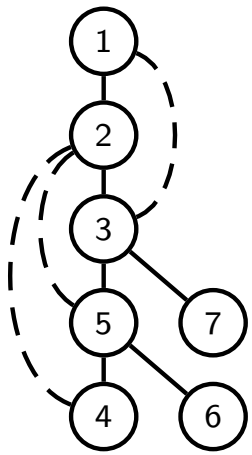




Esempio di esecuzione di DFS

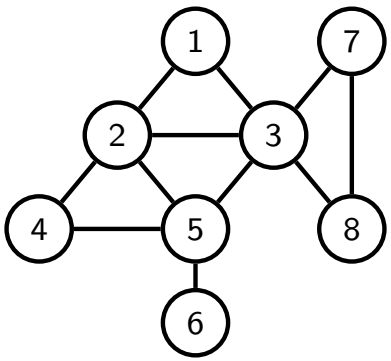
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

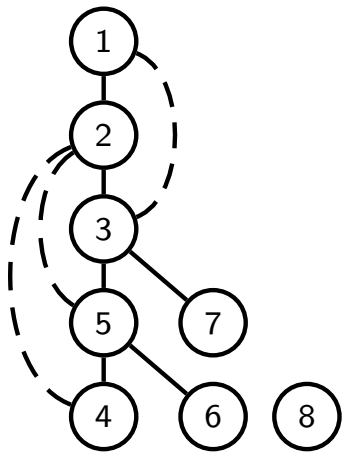




Esempio di esecuzione di DFS

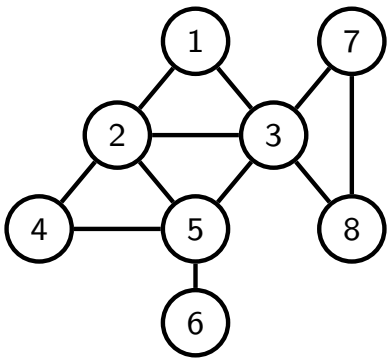
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

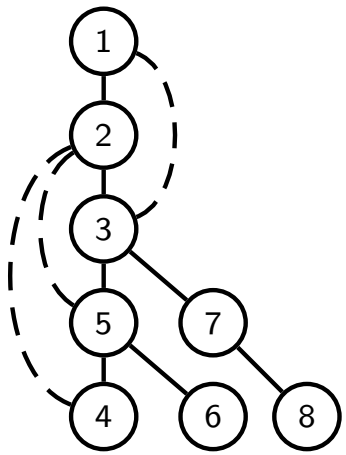




Esempio di esecuzione di DFS

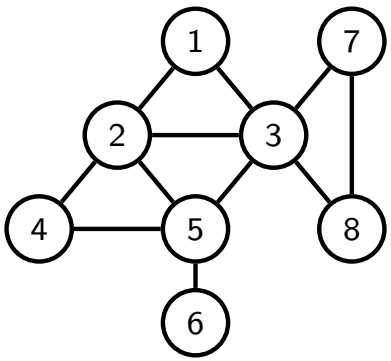
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

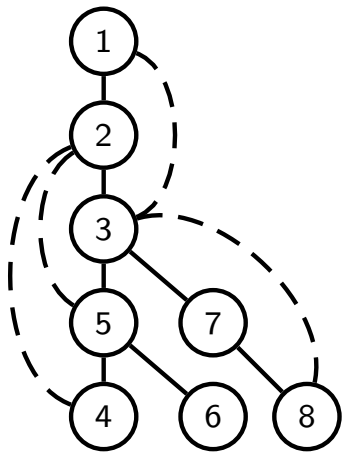




Esempio di esecuzione di DFS

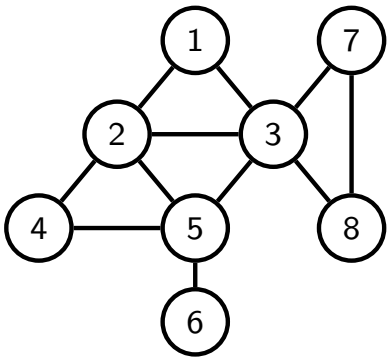
```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ‘‘Esplorato’’
    ricorsivamente chiama DFS(v)
```

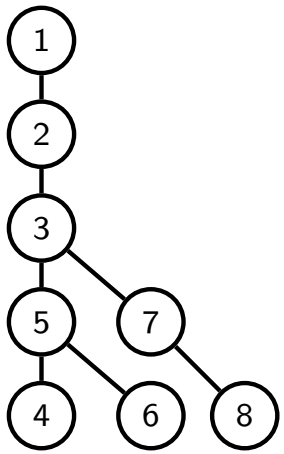




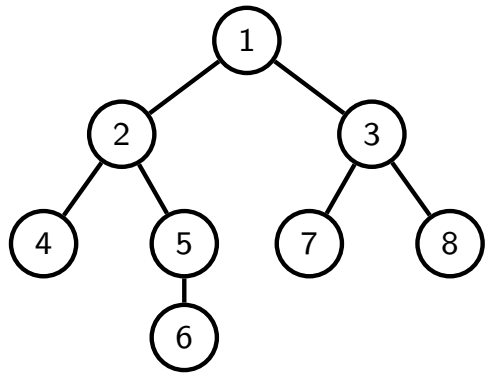
- A differenza dell'albero `BFS` che tende ad essere "corto e largo", l'albero `DFS` tende ad essere "lungo e stretto"

- A differenza dell'albero `BFS` che tende ad essere "corto e largo", l'albero `DFS` tende ad essere "lungo e stretto"





Albero DFS



Albero BFS

Qualche proprietà della visita `DFS`

Qualche proprietà della visita DFS

Fatto 1. Per ogni data chiamata ricorsiva di DFS(u), tutti i nodi che vengono marcati "Esplorato" dall'inizio della chiamata ricorsiva fino alla fine della ricorsione sono discendenti di u nell'albero DFS T.

Qualche proprietà della visita DFS

Fatto 1. Per ogni data chiamata ricorsiva di DFS(u), tutti i nodi che vengono marcati "Esplorato" dall'inizio della chiamata ricorsiva fino alla fine della ricorsione sono discendenti di u nell'albero DFS T.

```
DFS(u)
Marca il nodo u ``Esplorato'' ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ``Esplorato''
    ricorsivamente chiama DFS(v)
```

Qualche proprietà della visita DFS

Fatto 1. Per ogni data chiamata ricorsiva di DFS(u), tutti i nodi che vengono marcati "Esplorato" dall'inizio della chiamata ricorsiva fino alla fine della ricorsione sono discendenti di u nell'albero DFS T.

```
DFS(u)
Marca il nodo u ``Esplorato'' ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ``Esplorato''
    ricorsivamente chiama DFS(v)
```

Per come funziona DFS(u), il primo vertice v che viene marcato "Esplorato" durante l'esecuzione di DFS(u) è un vicino di u, ed esso diventa figlio di u nell'albero T

Qualche proprietà della visita DFS

Fatto 1. Per ogni data chiamata ricorsiva di DFS(u), tutti i nodi che vengono marcati "Esplorato" dall'inizio della chiamata ricorsiva fino alla fine della ricorsione sono discendenti di u nell'albero DFS T.

```
DFS(u)
Marca il nodo u ''Esplorato'' ed inseriscilo in T
For ogni arco (u, v) incidente su u
  If v non è marcato ''Esplorato''
    ricorsivamente chiama DFS(v)
```

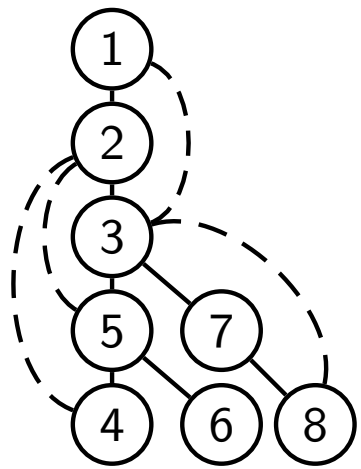
Per come funziona DFS(u), il primo vertice v che viene marcato "Esplorato" durante l'esecuzione di DFS(u) è un vicino di u, ed esso diventa figlio di u nell'albero T

Successivamente viene chiamato DFS(v) che introduce (eventualmente) nell'alberto T un nuovo nodo w che sarà figlio di v (ovvero discendente di u) e così via per tutti gli eventuali nuovi nodi inseriti in T (ovvero marcati "Esplorato")

Fatto 2. Sia T un albero DFS e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

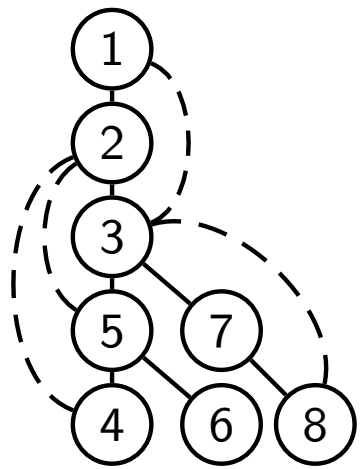
Fatto 2. Sia T un albero DFS e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla DFS.



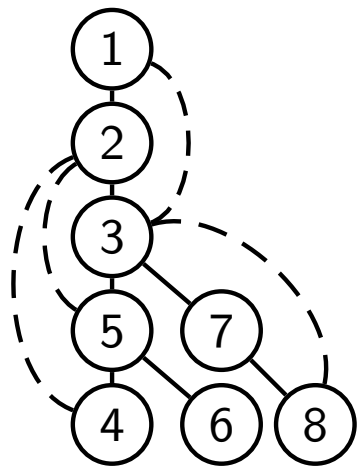
Fatto 2. Sia T un albero `DFS` e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla `DFS`. Per ipotesi, quando l'arco (x, y) è esaminato durante l'esecuzione di `DFS`(x) esso non viene aggiunto a T in quanto y è marcato "`Esplorato`".



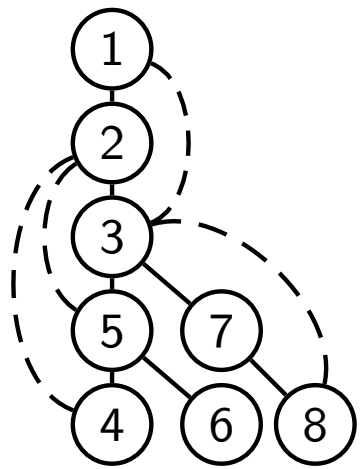
Fatto 2. Sia T un albero DFS e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla DFS. Per ipotesi, quando l'arco (x, y) è esaminato durante l'esecuzione di DFS(x) esso non viene aggiunto a T in quanto y è marcato "Esplorato". Poichè x è stato raggiunto prima di y dalla DFS, al momento della chiamata ricorsiva DFS(x) il nodo y *non* è marcato "Esplorato".



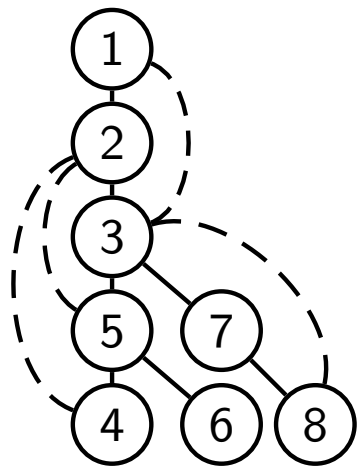
Fatto 2. Sia T un albero DFS e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla DFS. Per ipotesi, quando l'arco (x, y) è esaminato durante l'esecuzione di DFS(x) esso non viene aggiunto a T in quanto y è marcato "Esplorato". Poichè x è stato raggiunto prima di y dalla DFS, al momento della chiamata ricorsiva DFS(x) il nodo y *non* è marcato "Esplorato". Ciò vuol dire che y è stato scoperto in qualche momento successivo alla chiamata ricorsiva DFS(x)



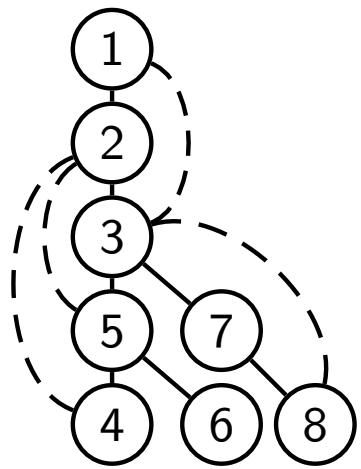
Fatto 2. Sia T un albero `DFS` e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla `DFS`. Per ipotesi, quando l'arco (x, y) è esaminato durante l'esecuzione di `DFS`(x) esso non viene aggiunto a T in quanto y è marcato "`Esplorato`". Poichè x è stato raggiunto prima di y dalla `DFS`, al momento della chiamata ricorsiva `DFS`(x) il nodo y *non* è marcato "`Esplorato`". Ciò vuol dire che y è stato scoperto in qualche momento successivo alla chiamata ricorsiva `DFS`(x) e prima del completamento della ricorsione in `DFS`(x).



Fatto 2. Sia T un albero `DFS` e siano x e y due nodi in T. Sia (x, y) un arco del grafo G che non è un arco di T. Allora o x è un ancestore di y o y è un ancestore di x.

Sia (x, y) un tale arco e supponiamo che il nodo x sia raggiunto per primo dalla `DFS`. Per ipotesi, quando l'arco (x, y) è esaminato durante l'esecuzione di `DFS`(x) esso non viene aggiunto a T in quanto y è marcato "`Esplorato`". Poichè x è stato raggiunto prima di y dalla `DFS`, al momento della chiamata ricorsiva `DFS`(x) il nodo y *non* è marcato "`Esplorato`". Ciò vuol dire che y è stato scoperto in qualche momento successivo alla chiamata ricorsiva `DFS`(x) e prima del completamento della ricorsione in `DFS`(x). Dal Fatto 1 segue che y è un discendente di x nell'albero T.



Riassumendo:

- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u,

Riassumendo:

- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene

Riassumendo:

- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla `BFS`).

Riassumendo:

- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla `BFS`).
- Alla fine dell'esecuzione di `DFS`(u), gli archi della parte di grafo G che costituisce la componente connessa di u sono suddivisi in due insiemi A e B disgiunti:

Riassumendo:

- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla `BFS`).
- Alla fine dell'esecuzione di `DFS`(u), gli archi della parte di grafo G che costituisce la componente connessa di u sono suddivisi in due insiemi A e B disgiunti: **1.** $A =$archi dell'albero T,

Riassumendo:

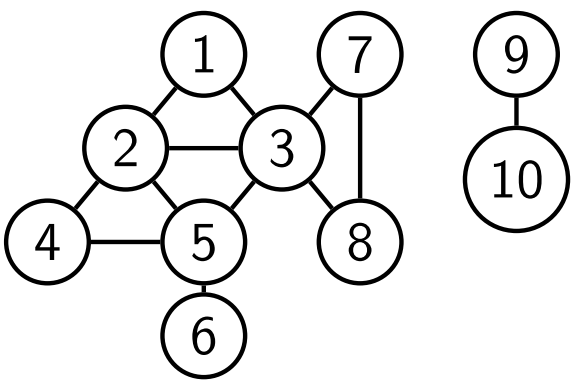
- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla `BFS`).
- Alla fine dell'esecuzione di `DFS`(u), gli archi della parte di grafo G che costituisce la componente connessa di u sono suddivisi in due insiemi A e B disgiunti: **1.** A =archi dell'albero T, **2.** B=archi di G che connettono ancestori-discendenti di T

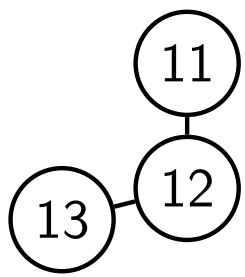
Riassumendo:

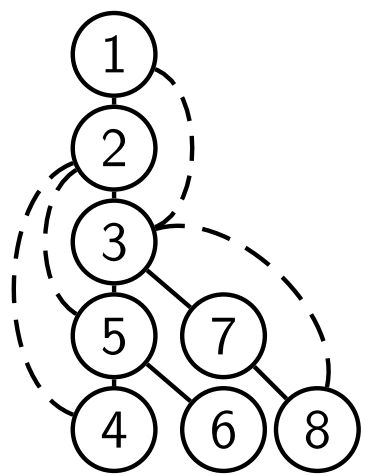
- Alla fine dell'esecuzione di `DFS`(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla `BFS`).
- Alla fine dell'esecuzione di `DFS`(u), gli archi della parte di grafo G che costituisce la componente connessa di u sono suddivisi in due insiemi A e B disgiunti: **1.** A =archi dell'albero T, **2.** B=archi di G che connettono ancestori-discendenti di T ($\equiv$tali archi sono delle "scorciatoie in T)

Riassumendo:

- Alla fine dell'esecuzione di DFS(u) l'albero T generato contiene tutti i nodi raggiungibili da u, ovvero la componente connessa cui u appartiene (analoga proprietà valeva per l'albero generato dalla BFS).
- Alla fine dell'esecuzione di DFS(u), gli archi della parte di grafo G che costituisce la componente connessa di u sono suddivisi in due insiemi A e B disgiunti: **1.** A =archi dell'albero T, **2.** B=archi di G che connettono ancestori-discendenti di T ($\equiv$tali archi sono delle "scorciatoie in T)







Albero risultante da DFS(1)

Quindi anche `DFS` può essere usata per determinare le componenti connesse di G, nel senso che per ogni nodo u del grafo G, l'albero prodotto da `DFS`(u) (o `BFS`(u)) consiste della componente connessa contenente u (ovverosia l'albero consiste di tutti e soli i nodi raggiungibili da u)

Quindi anche `DFS` può essere usata per determinare le componenti connesse di G, nel senso che per ogni nodo u del grafo G, l'albero prodotto da `DFS`(u) (o `BFS`(u)) consiste della componente connessa contenente u (ovverosia l'albero consiste di tutti e soli i nodi raggiungibili da u)
Domanda: *dati due nodi u e v, che relazione sussiste tra la componente connessa di u e quella di v?*

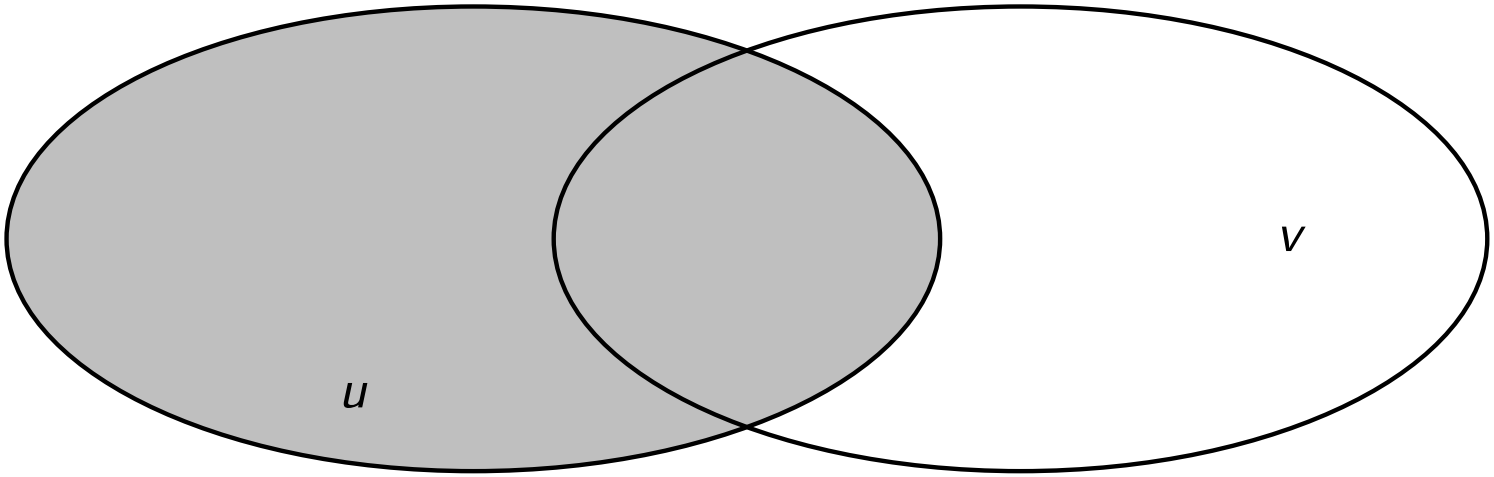
Quindi anche `DFS` può essere usata per determinare le componenti connesse di G, nel senso che per ogni nodo u del grafo G, l'albero prodotto da `DFS`(u) (o `BFS`(u)) consiste della componente connessa contenente u (ovverosia l'albero consiste di tutti e soli i nodi raggiungibili da u)
Domanda: *dati due nodi u e v, che relazione sussiste tra la componente connessa di u e quella di v?*

Fatto 3. Per ogni coppia di nodi u e v nel grafo, le loro componenti connesse o sono uguali o sono disgiunte (cioè non hanno alcun nodo in comune)

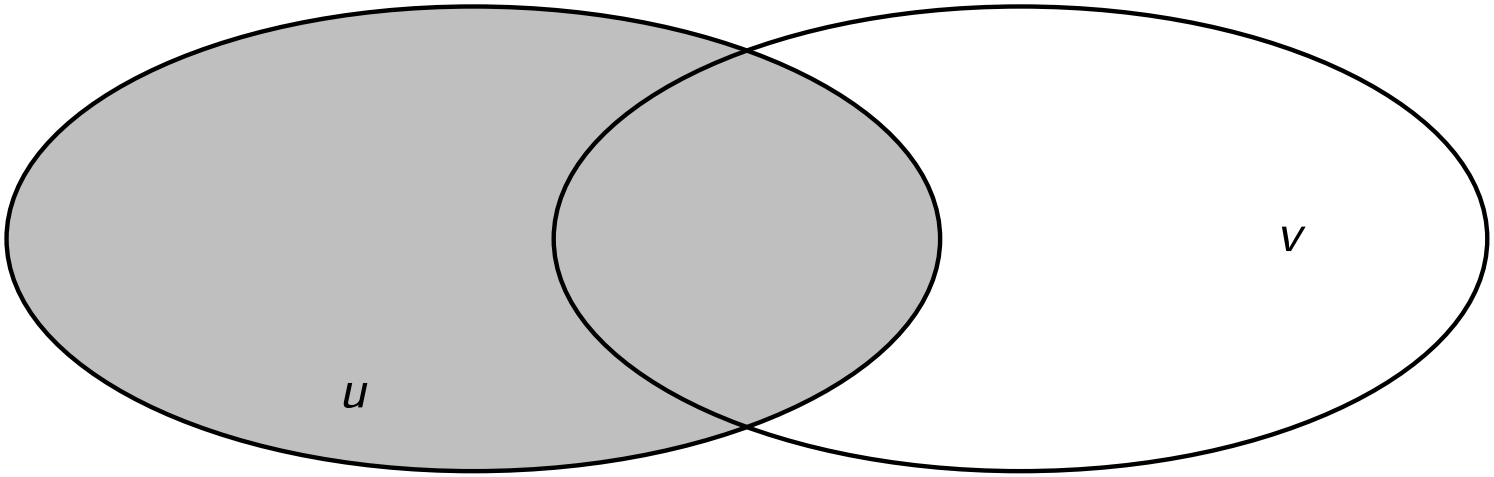
Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte.

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:

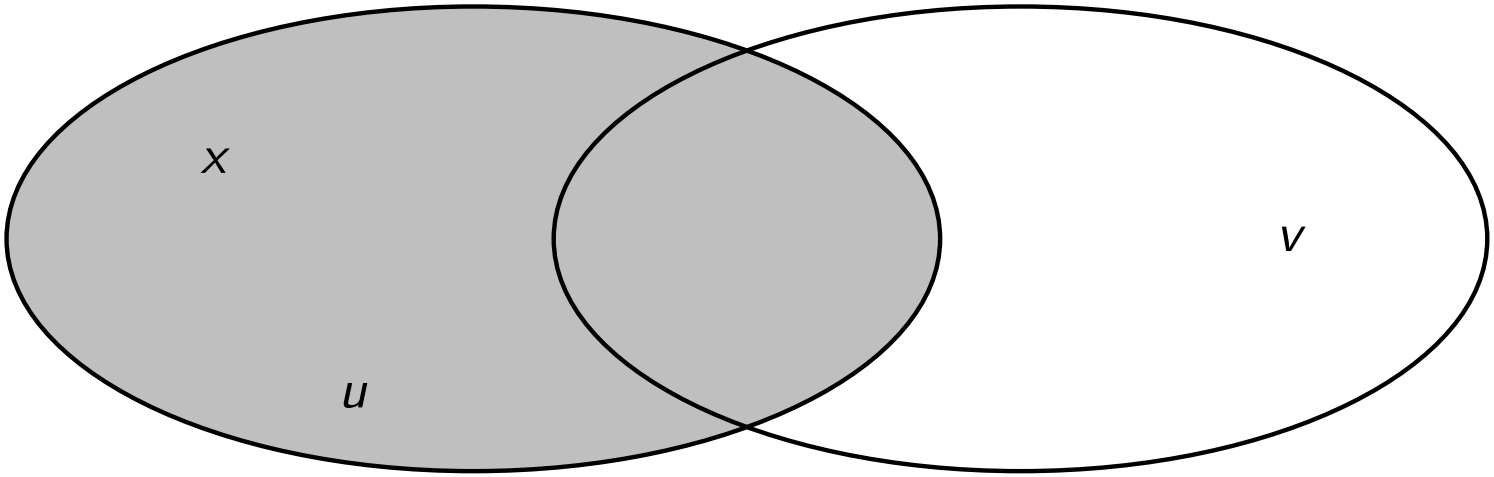


Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



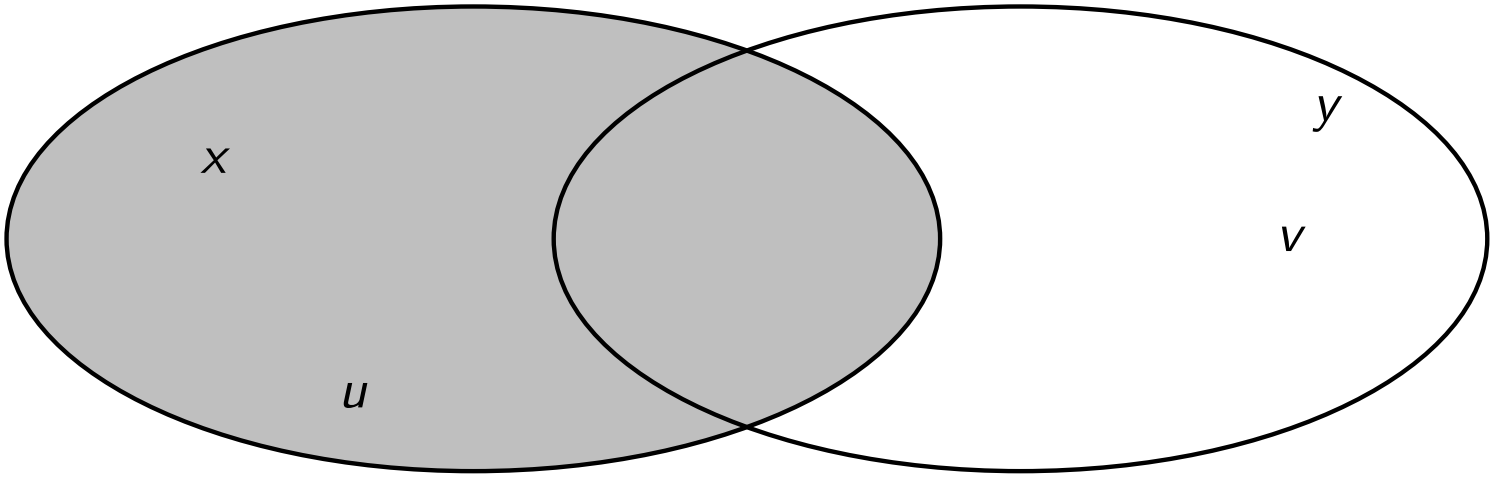
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi)

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



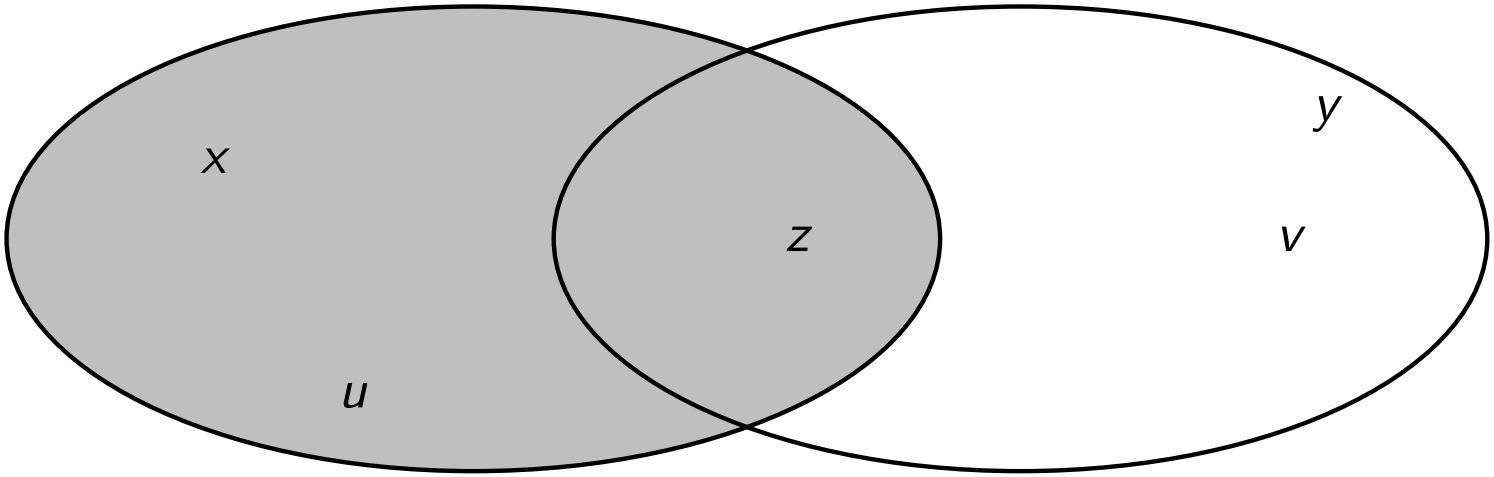
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



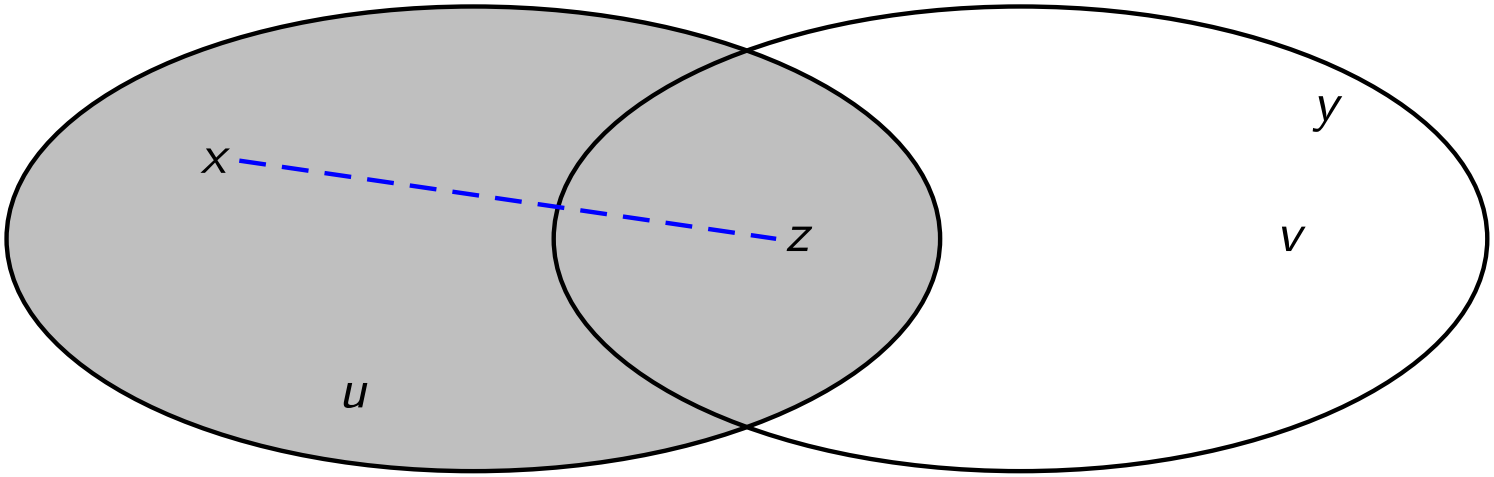
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v,

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



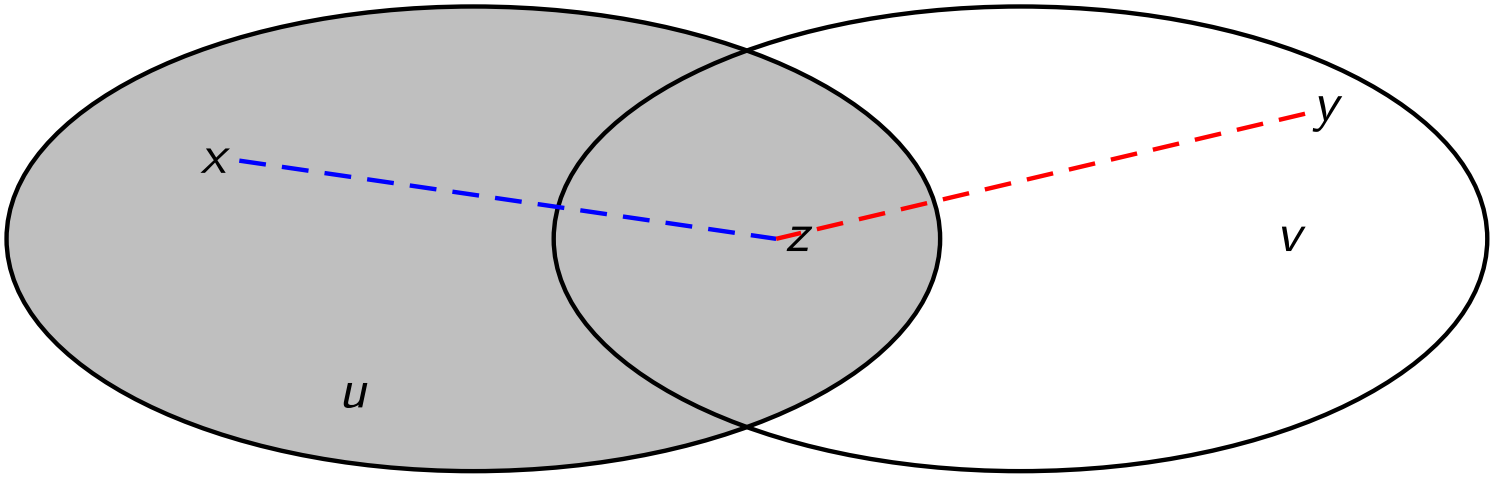
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti.

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



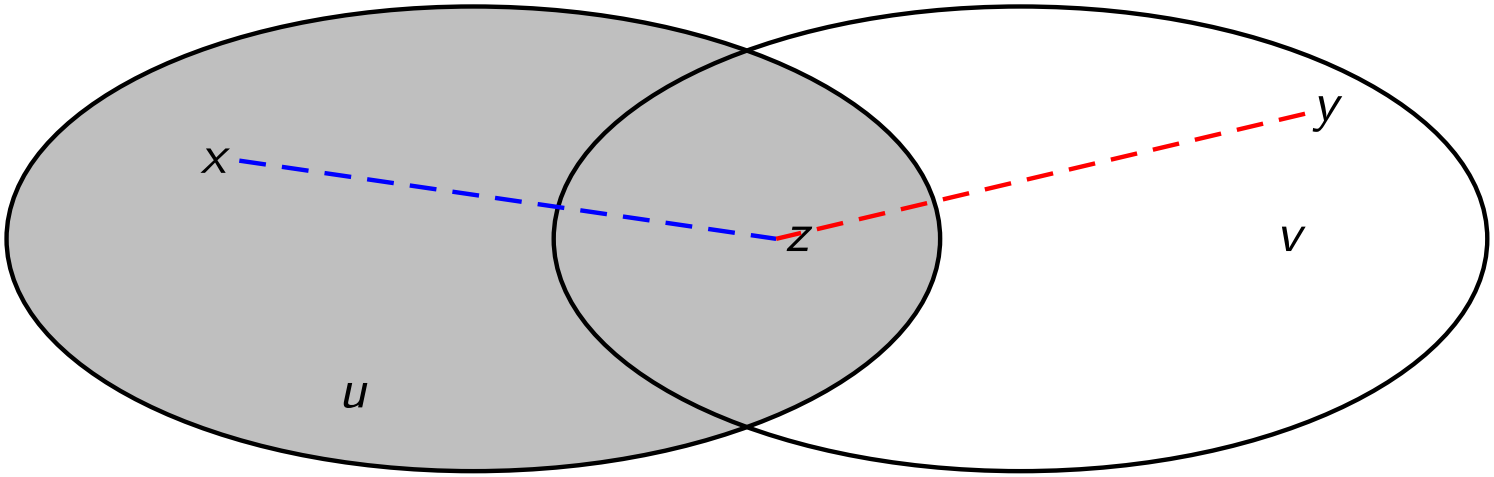
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti. Per ipotesi, esiste un cammino da x a z,

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



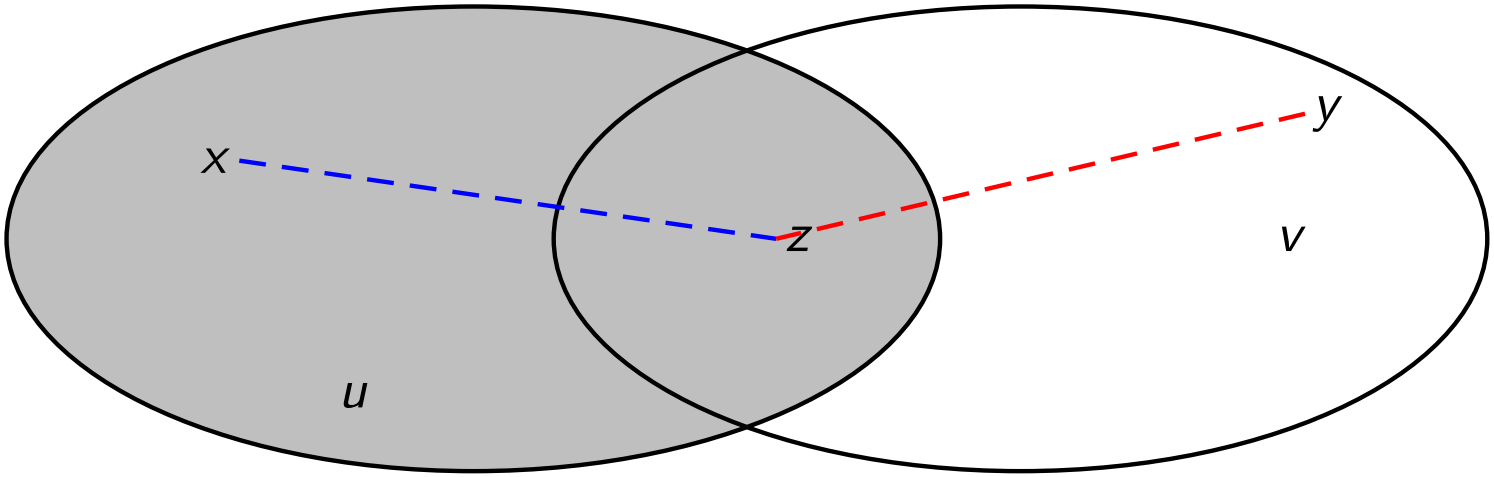
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti. Per ipotesi, esiste un cammino da x a z, ed un cammino da z a y,

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



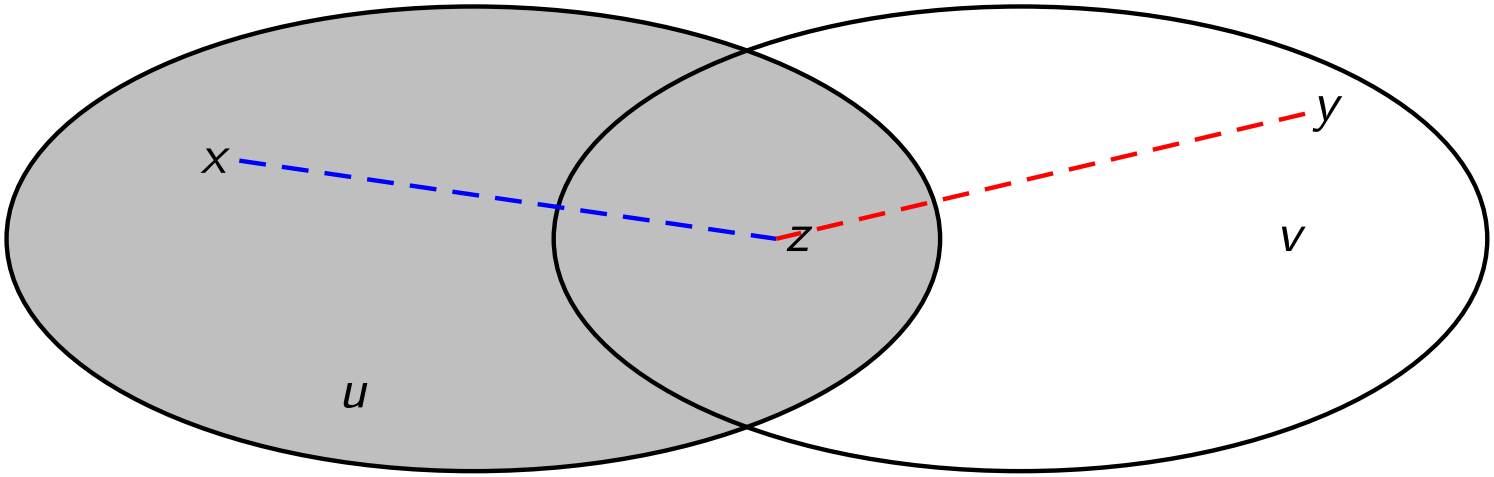
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti. Per ipotesi, esiste un cammino da x a z, ed un cammino da z a y, *ergo* un cammino da x a y.

Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti. Per ipotesi, esiste un cammino da x a z, ed un cammino da z a y, *ergo* un cammino da x a y. Ne segue che x e y sono in una stessa componente connessa

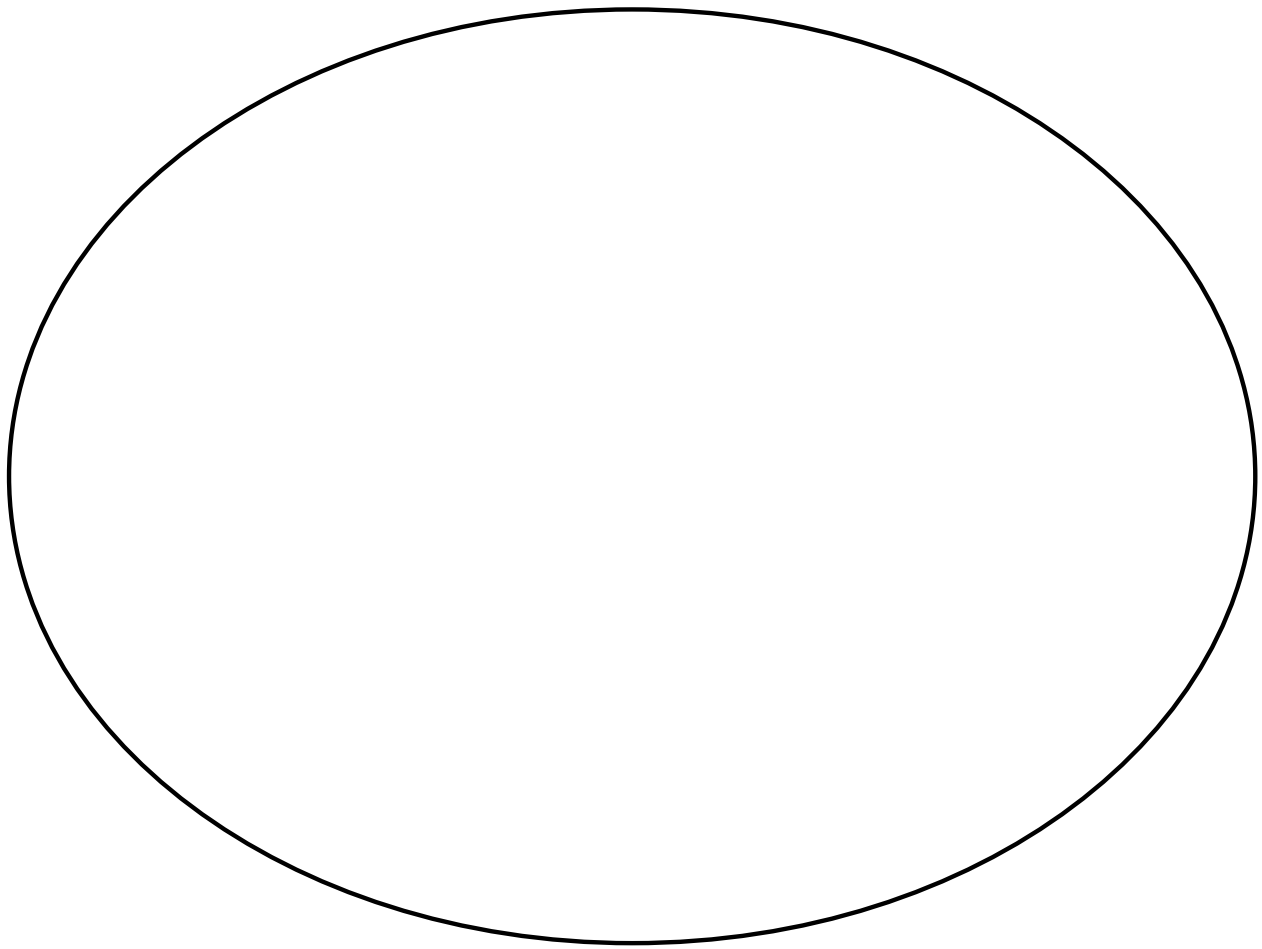
Supponiamo per assurdo che le componenti connesse non siano nè uguali nè disgiunte. Allora le componenti connesse di u e v saranno fatte così:



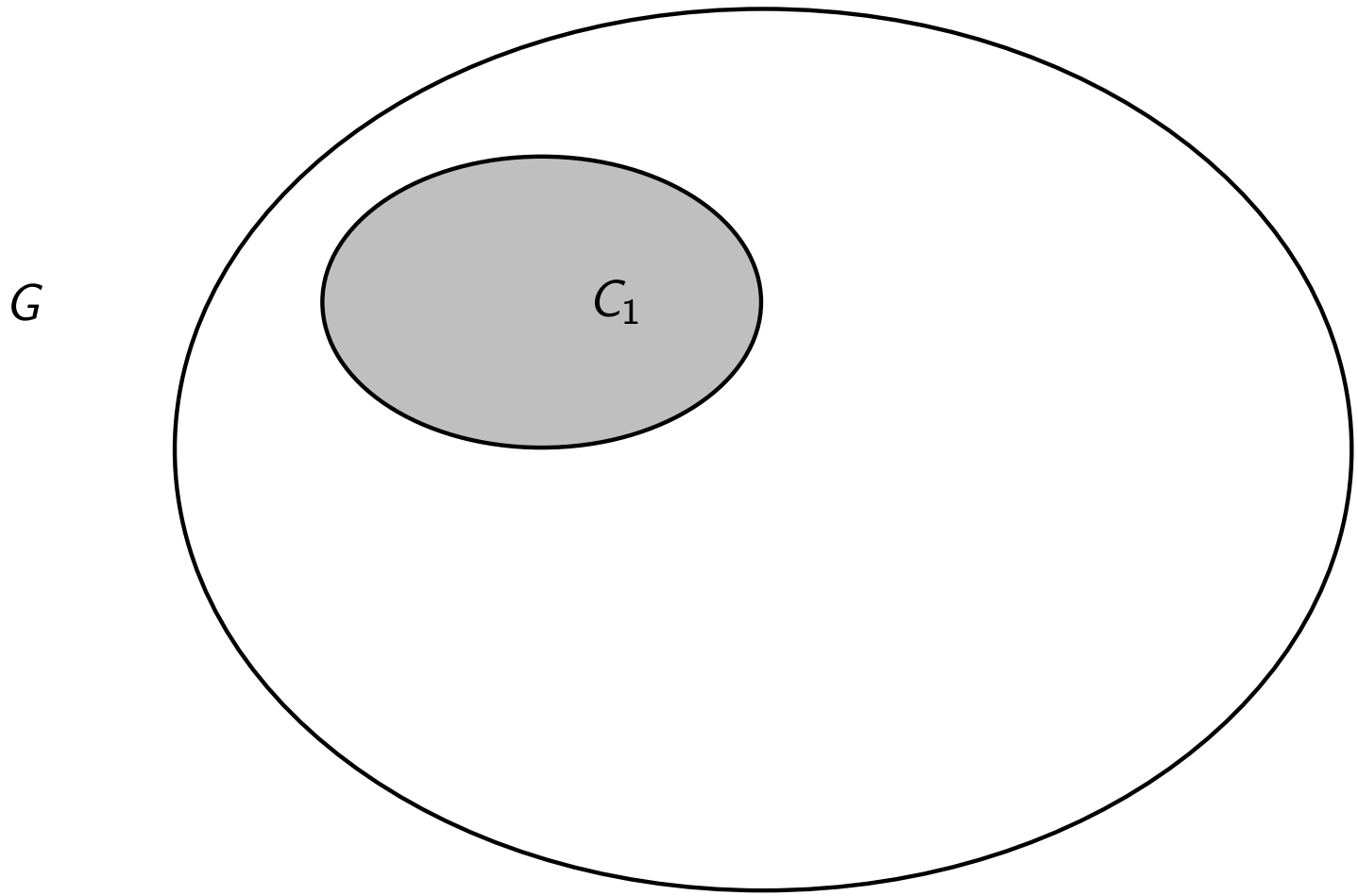
(ricordiamo due nodi sono in una stessa componente connessa se e solo se esiste un cammino tra di essi) Consideriamo due nodi arbitrari x e y, con x nella componente connessa di u e y nella componente connessa di v, e z un nodo nella intersezione delle componenti. Per ipotesi, esiste un cammino da x a z, ed un cammino da z a y, *ergo* un cammino da x a y. Ne segue che x e y sono in una stessa componente connessa e quindi la componente connessa di u =componente connessa di v.

Una conseguenza di questo risultato è che l'insieme dei vertici di un generico grafo $G = (V, E)$ si può partizionare in componenti connesse $C_1, \ldots, C_k$ (ovvero tali che $C_1 \cup C_2 \cup \ldots \cup C_k = V$ e $C_i \cap C_j = \emptyset$, per $i \neq j$), dove $\forall i = 1, \ldots, k$, $\forall a, b \in C_i$ esiste un cammino da a a b.

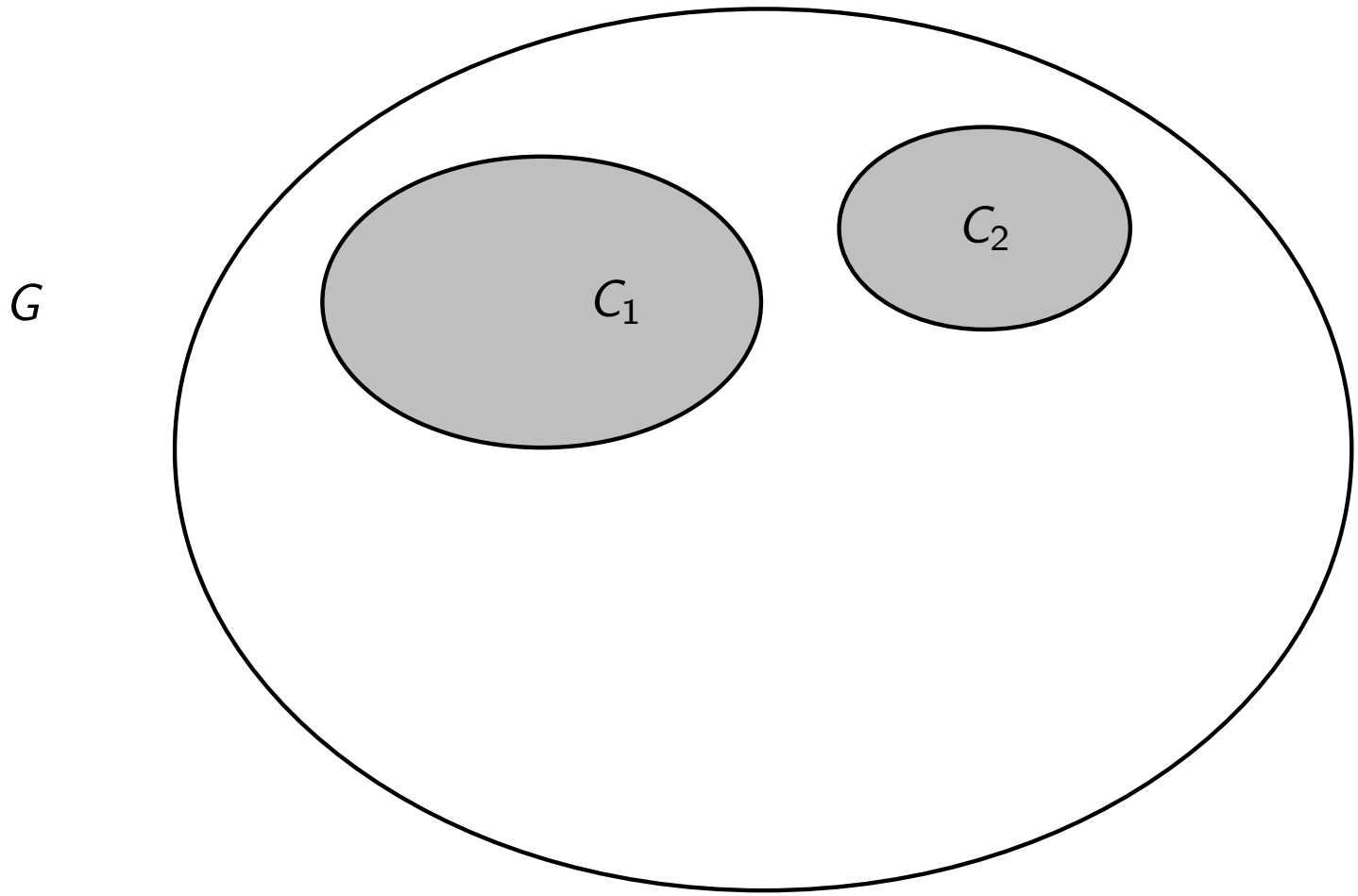
G



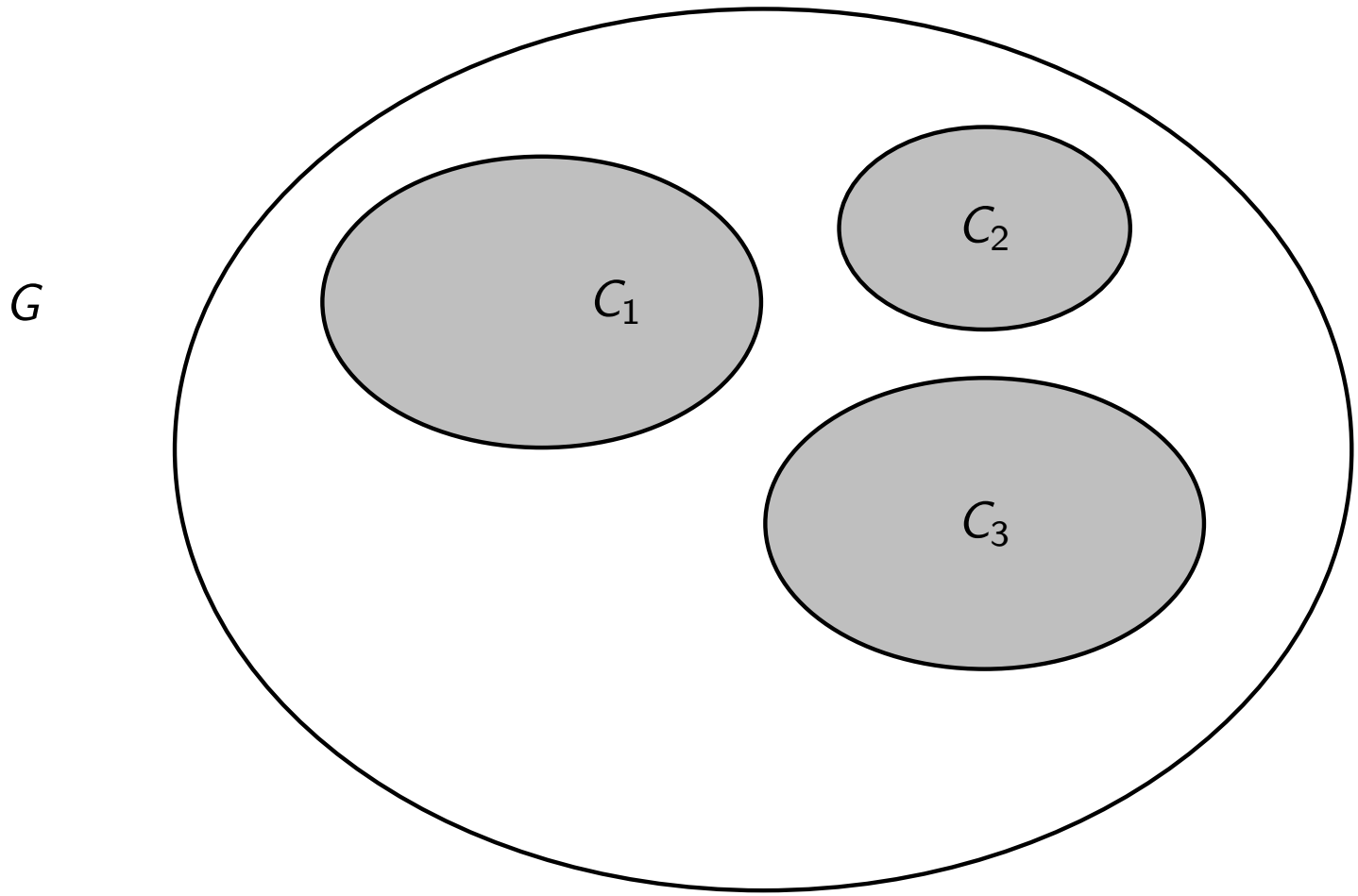
Una conseguenza di questo risultato è che l'insieme dei vertici di un generico grafo $G = (V, E)$ si può partizionare in componenti connesse $C_1, \ldots, C_k$ (ovvero tali che $C_1 \cup C_2 \cup \ldots \cup C_k = V$ e $C_i \cap C_j = \emptyset$, per $i \neq j$), dove $\forall i = 1, \ldots, k$, $\forall a, b \in C_i$ esiste un cammino da a a b.



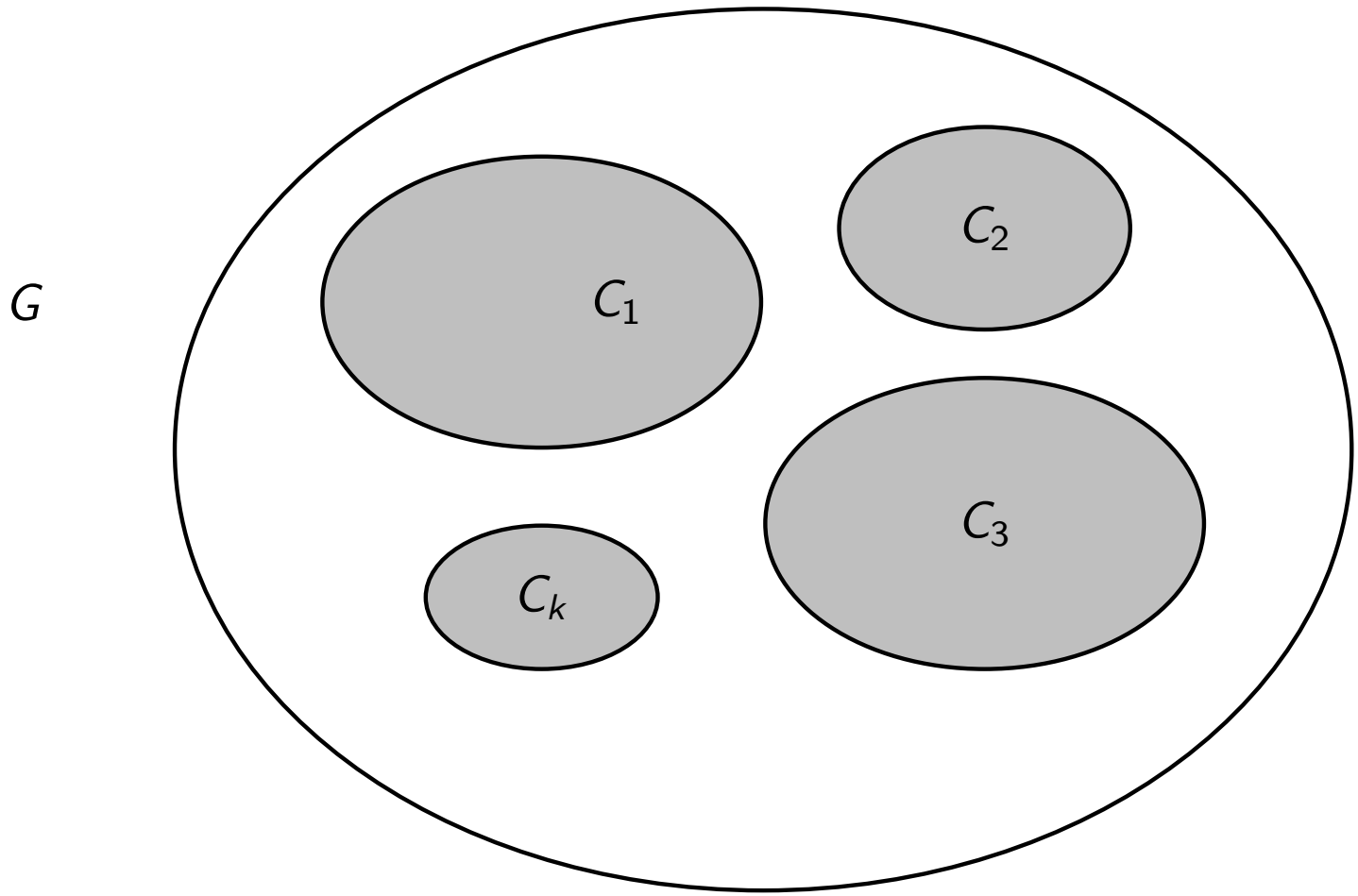
Una conseguenza di questo risultato è che l'insieme dei vertici di un generico grafo $G = (V, E)$ si può partizionare in componenti connesse $C_1, \ldots, C_k$ (ovvero tali che $C_1 \cup C_2 \cup \ldots \cup C_k = V$ e $C_i \cap C_j = \emptyset$, per $i \neq j$), dove $\forall i = 1, \ldots, k$, $\forall a, b \in C_i$ esiste un cammino da a a b.



Una conseguenza di questo risultato è che l'insieme dei vertici di un generico grafo $G = (V, E)$ si può partizionare in componenti connesse $C_1, \ldots, C_k$ (ovvero tali che $C_1 \cup C_2 \cup \ldots \cup C_k = V$ e $C_i \cap C_j = \emptyset$, per $i \neq j$), dove $\forall i = 1, \ldots, k$, $\forall a, b \in C_i$ esiste un cammino da a a b.



Una conseguenza di questo risultato è che l'insieme dei vertici di un generico grafo $G = (V, E)$ si può partizionare in componenti connesse $C_1, \ldots, C_k$ (ovvero tali che $C_1 \cup C_2 \cup \ldots \cup C_k = V$ e $C_i \cap C_j = \emptyset$, per $i \neq j$), dove $\forall i = 1, \ldots, k$, $\forall a, b \in C_i$ esiste un cammino da a a b.



Come si possono calcolare tutte le componenti connesse di un grafo?

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente `DFS` (o `BFS`) su nodi non ancora visitati, fin quando ve ne sono.

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente `DFS` (o `BFS`) su nodi non ancora visitati, fin quando ve ne sono.

`DFS`(G)

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente `DFS` (o `BFS`) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
```

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente DFS (o BFS) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
   marca u ‘‘non Esplorato’’
```

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente DFS (o BFS) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
   marca u ‘‘non Esplorato’’
For ciascun vertice u ∈ V
```

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente DFS (o BFS) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
   marca u ‘‘non Esplorato’’
For ciascun vertice u ∈ V
   If (u ‘‘non Esplorato’’)
```

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente DFS (o BFS) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
   marca u ‘‘non Esplorato’’
For ciascun vertice u ∈ V
   If (u ‘‘non Esplorato’’)
    then DFS(G, u)
```

Come si possono calcolare tutte le componenti connesse di un grafo?

Visto che le componenti connesse di un grafo $G = (V, E)$ sono disgiunte tra di loro, le possiamo generare tutte, una alla volta, chiamando iterativamente DFS (o BFS) su nodi non ancora visitati, fin quando ve ne sono.

```
DFS(G)
For ciascun vertice u ∈ V
   marca u ‘‘non Esplorato’’
For ciascun vertice u ∈ V
   If (u ‘‘non Esplorato’’)
    then DFS(G, u)
```

Complessità: $\Theta(n + m)$, dove $n = |V|, m = |E|$

Abbiamo presentato la `DFS` nella sua versione ricorsiva.

Abbiamo presentato la `DFS` nella sua versione ricorsiva. Usando uno stack (*first-in*, *last-out*) possiamo darne una versione esplicita:

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in*, *last-out*) possiamo darne una versione esplicita:

```
DFS(s)
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in*, *last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
    Estrai un nodo u da S
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
    Estrai un nodo u da S
    If Explored[u] =false then
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
    Estrai un nodo u da S
    If Explored[u] =false then
      Explored[u] ← true
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
    Estrai un nodo u da S
    If Explored[u] =false then
      Explored[u] ← true
      For ogni arco (u, v) uscente da u
```

Abbiamo presentato la DFS nella sua versione ricorsiva. Usando uno stack (*first-in, last-out*) possiamo darne una versione esplicita:

```
DFS(s)
For ciascun vertice u ∈ V
    Explored[u] ←false
Inserisci s nello stack S
While S ≠ ∅
    Estrai un nodo u da S
    If Explored[u] =false then
      Explored[u] ← true
      For ogni arco (u, v) uscente da u
       Aggiungi v allo stack S
```

Questo algoritmo è essenzialmente lo stesso algoritmo ricorsivo prima presentato.

È interessante notare che la DFS e la BFS sono formalmente lo "stesso" algoritmo, la DFS usa uno stack mentre la BFS usa una coda (ignorando ovviamente il computo delle distanze dei nodi dalla sorgente s ed altre questioni accessorie).

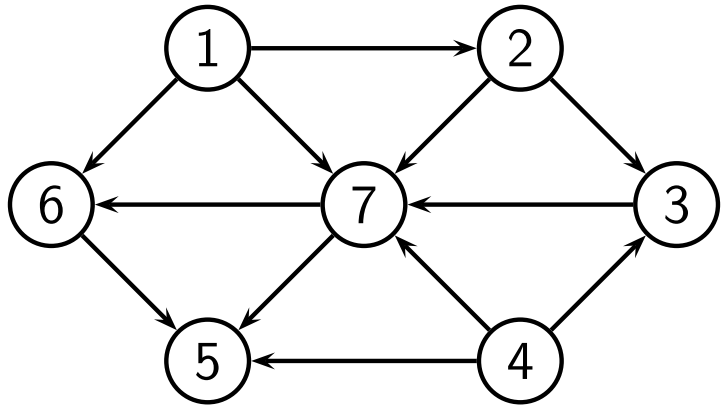
È interessante notare che la DFS e la BFS sono formalmente lo "stesso" algoritmo, la DFS usa uno stack mentre la BFS usa una coda (ignorando ovviamente il computo delle distanze dei nodi dalla sorgente s ed altre questioni accessorie).

DFS(s)
For ciascun vertice $u \in V$
  Explored[u] $\leftarrow$false
Inserisci s nello stack S
While $S \neq \emptyset$
  Estrai un nodo u da S
  If Explored[u] =`false` **then**
    Explored[u] $\leftarrow$ `true`
    $\forall$ arco (u, v) uscente da u
      Aggiungi v allo stack S

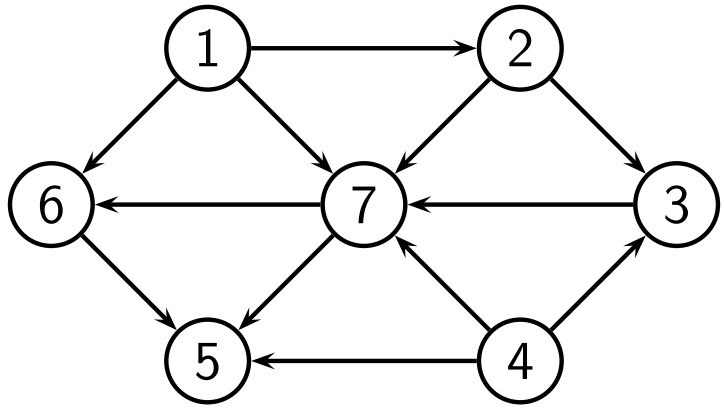
BFS(G, s)
For ciascun vertice $u \in V$
  Scoperto[v]=`false`
`Scoperto`[s] =`true`
Inserisci s nella coda Q;
While $Q \neq \emptyset$
  Estrai il nodo u dalla testa della lista Q
    $\forall(u, v)$ incidente su u
    If Scoperto[v]=`false` **then**
      Scoperto[v] $\leftarrow$`true`
    Aggiungi v alla fine della coda Q

Parliamo ora di grafi diretti.
Grafo diretto $G = (V, E)$, in cui ogni arco $(u, v) \in E$ ha una direzione "dal nodo u al nodo v".

Parliamo ora di grafi diretti.
Grafo diretto $G = (V, E)$, in cui ogni arco $(u, v) \in E$ ha una direzione "dal nodo u al nodo v". Esempio:

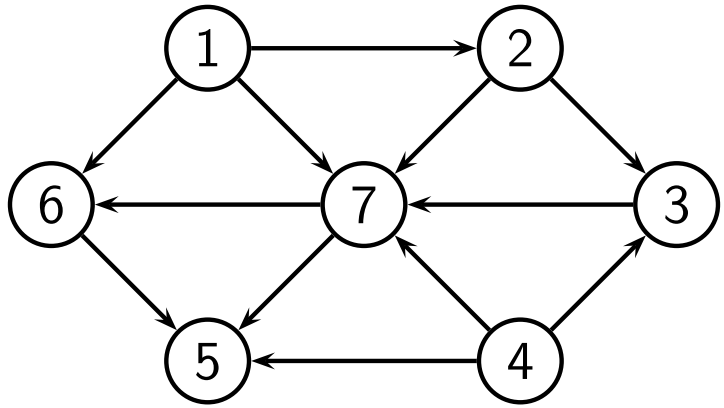


Parliamo ora di grafi diretti.
Grafo diretto $G = (V, E)$, in cui ogni arco $(u, v) \in E$ ha una direzione "dal nodo u al nodo v". Esempio:



Es.: Grafo del Web – ogni hyperlink punta da una pagina web all'altra

Parliamo ora di grafi diretti.
Grafo diretto $G = (V, E)$, in cui ogni arco $(u, v) \in E$ ha una direzione "dal nodo u al nodo v". Esempio:



Es.: Grafo del Web – ogni hyperlink punta da una pagina web all'altra
Il grafo che rappresenta il Web (nodi≡pagine, archi≡hyperlink) è quindi naturalmente un grafo diretto, e la sua "direzionalità" è cruciale per i moderni motori di ricerca che classificano l'importanza di una pagina web sulla base della struttura del grafo risultante.

La visita BFS funziona anche in grafi diretti:

BFS(G, s)

1. Poni Scoperto$[s]$ =true e Scoperto$[v]$ =false $\forall v \neq s$

La visita BFS funziona anche in grafi diretti:

BFS(G, s)
1. Poni Scoperto$[s]$ =true e Scoperto$[v]$ =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$

La visita BFS funziona anche in grafi diretti:

BFS(G, s)

1. Poni Scoperto$[s]$ =true e Scoperto$[v]$ =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$
3. Inizializza l'albero BFS T a $\emptyset$

La visita BFS funziona anche in grafi diretti:

BFS(G, s)
1. Poni Scoperto[s] =true e Scoperto[v] =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$
3. Inizializza l'albero BFS T a $\emptyset$
4. While Q non è vuota

La visita BFS funziona anche in grafi diretti:

BFS(G, s)
1. Poni Scoperto[s] =true e Scoperto[v] =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$
3. Inizializza l'albero BFS T a $\emptyset$
4. While Q non è vuota
5. Estrai il nodo u dalla testa della lista Q

La visita BFS funziona anche in grafi diretti:

BFS(G, s)
1. Poni Scoperto[s] =true e Scoperto[v] =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$
3. Inizializza l'albero BFS T a $\emptyset$
4. While Q non è vuota
5. Estrai il nodo u dalla testa della lista Q
6. Considera ciascun arco (u, v) ''uscente'' da u

La visita BFS funziona anche in grafi diretti:

BFS(G, s)
1. Poni Scoperto[s] =true e Scoperto[v] =false $\forall v \neq s$
2. Inizializza Q in modo che contenga s; $d[s] \leftarrow 0$
3. Inizializza l'albero BFS T a $\emptyset$
4. While Q non è vuota
5. Estrai il nodo u dalla testa della lista Q
6. Considera ciascun arco (u, v) ''uscente'' da u
7. If Scoperto[v]=false then

La visita BFS funziona anche in grafi diretti:

```
BFS(G, s)
1.  Poni Scoperto[s] =true e Scoperto[v] =false ∀v ≠ s
2.  Inizializza Q in modo che contenga s; d[s] ← 0
3.  Inizializza l'albero BFS T a ∅
4.  While Q non è vuota
5.  Estrai il nodo u dalla testa della lista Q
6.      Considera ciascun arco (u, v) ''uscente'' da u
7.      If Scoperto[v]=false then
8.        Poni Scoperto[v]=true
```

La visita BFS funziona anche in grafi diretti:

```
BFS(G, s)
1.  Poni Scoperto[s] =true e Scoperto[v] =false ∀v ≠ s
2.  Inizializza Q in modo che contenga s; d[s] ← 0
3.  Inizializza l'albero BFS T a ∅
4.  While Q non è vuota
5.  Estrai il nodo u dalla testa della lista Q
6.      Considera ciascun arco (u, v) ''uscente'' da u
7.      If Scoperto[v]=false then
8.        Poni Scoperto[v]=true
9.        Aggiungi l'arco (u, v) all'albero T
```

La visita BFS funziona anche in grafi diretti:

```
BFS(G, s)
1.  Poni Scoperto[s] =true e Scoperto[v] =false ∀v ≠ s
2.  Inizializza Q in modo che contenga s; d[s] ← 0
3.  Inizializza l'albero BFS T a ∅
4.  While Q non è vuota
5.  Estrai il nodo u dalla testa della lista Q
6.      Considera ciascun arco (u, v) ''uscente'' da u
7.      If Scoperto[v]=false then
8.        Poni Scoperto[v]=true
9.        Aggiungi l'arco (u, v) all'albero T
10.       Aggiungi v alla fine della coda Q;
            d[v] ← d[u] + 1
```

Così come per la BFS in grafi non diretti, anche nel caso di grafi diretti la BFS produce un alberto T.

Così come per la BFS in grafi non diretti, anche nel caso di grafi diretti la BFS produce un alberto T. Tale albero contiene tutti i nodi dei vari livelli L_i, con un arco da $x \in L_i$ a $y \in L_{i+1}$ se e solo se il nodo y è stato scoperto da BFS *per la prima volta* partendo dal nodo x.

Così come per la BFS in grafi non diretti, anche nel caso di grafi diretti la BFS produce un alberto T. Tale albero contiene tutti i nodi dei vari livelli L_i, con un arco da $x \in L_i$ a $y \in L_{i+1}$ se e solo se il nodo y è stato scoperto da BFS *per la prima volta* partendo dal nodo x.
L'albero T ha la proprietà che esso contiene tutti e solo i nodi x per cui nel grafo esiste un cammino *diretto*

Così come per la BFS in grafi non diretti, anche nel caso di grafi diretti la BFS produce un alberto T. Tale albero contiene tutti i nodi dei vari livelli L_i, con un arco da $x \in L_i$ a $y \in L_{i+1}$ se e solo se il nodo y è stato scoperto da BFS *per la prima volta* partendo dal nodo x.
L'albero T ha la proprietà che esso contiene tutti e solo i nodi x per cui nel grafo esiste un cammino *diretto* (cioè un cammino in cui tutti gli archi sono nella direzione "giusta") dal nodo sorgente s della BFS ad x.

Così come per la BFS in grafi non diretti, anche nel caso di grafi diretti la BFS produce un alberto T. Tale albero contiene tutti i nodi dei vari livelli L_i, con un arco da $x \in L_i$ a $y \in L_{i+1}$ se e solo se il nodo y è stato scoperto da BFS *per la prima volta* partendo dal nodo x.
L'albero T ha la proprietà che esso contiene tutti e solo i nodi x per cui nel grafo esiste un cammino *diretto* (cioè un cammino in cui tutti gli archi sono nella direzione "giusta") dal nodo sorgente s della BFS ad x.
Tali nodi x possono (*o possono anche non*) avere un cammino diretto da essi a s

Ed anche la DFS funziona in grafi diretti.

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in R
For ogni arco (u, v) ‘‘uscente’’ da u
  If v non è marcato ‘‘Esplorato’’
    inserisci l’arco (u, v) in T e ricorsivamente chiama
DFS(v)
```

Ed anche la DFS funziona in grafi diretti.

```
DFS(u)
Marca il nodo u ‘‘Esplorato’’ ed inseriscilo in R
For ogni arco (u, v) ‘‘uscente’’ da u
  If v non è marcato ‘‘Esplorato’’
    inserisci l’arco (u, v) in T e ricorsivamente chiama
DFS(v)
```

Anche in questo caso l'albero T prodotto da DFS(u) ha la proprietà che esso contiene tutti i e solo i nodi x per cui nel grafo esiste un cammino *diretto* (cioè un cammino in cui tutti gli archi sono nella direzione "giusta") dal nodo u al nodo x.

Connettività forte in grafi diretti

I nodi u e v di un grafo G sono mutualmnente raggiungibili se e solo se esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Connettività forte in grafi diretti

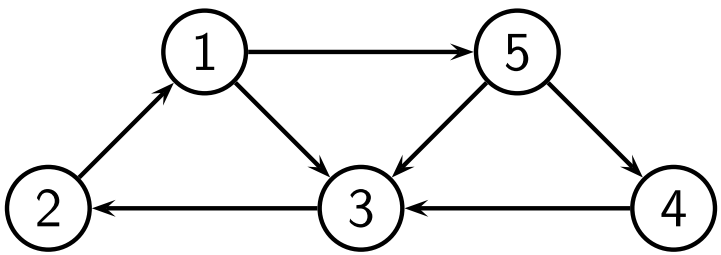
I nodi u e v di un grafo G sono mutualmnente raggiungibili se e solo se esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile

Connettività forte in grafi diretti

I nodi u e v di un grafo G sono mutualmnente raggiungibili se e solo se esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile

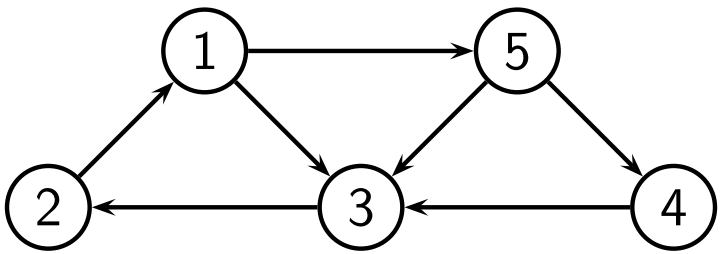


Grafo fortemente connesso

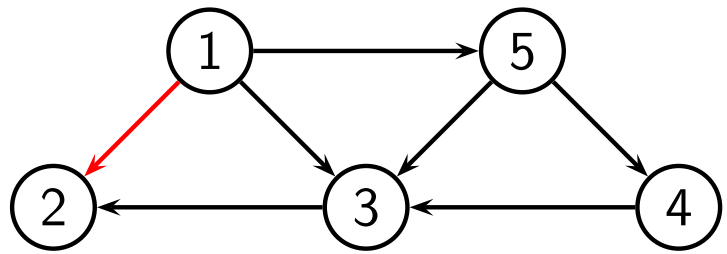
Connettività forte in grafi diretti

I nodi u e v di un grafo G sono mutualmnente raggiungibili se e solo se esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile

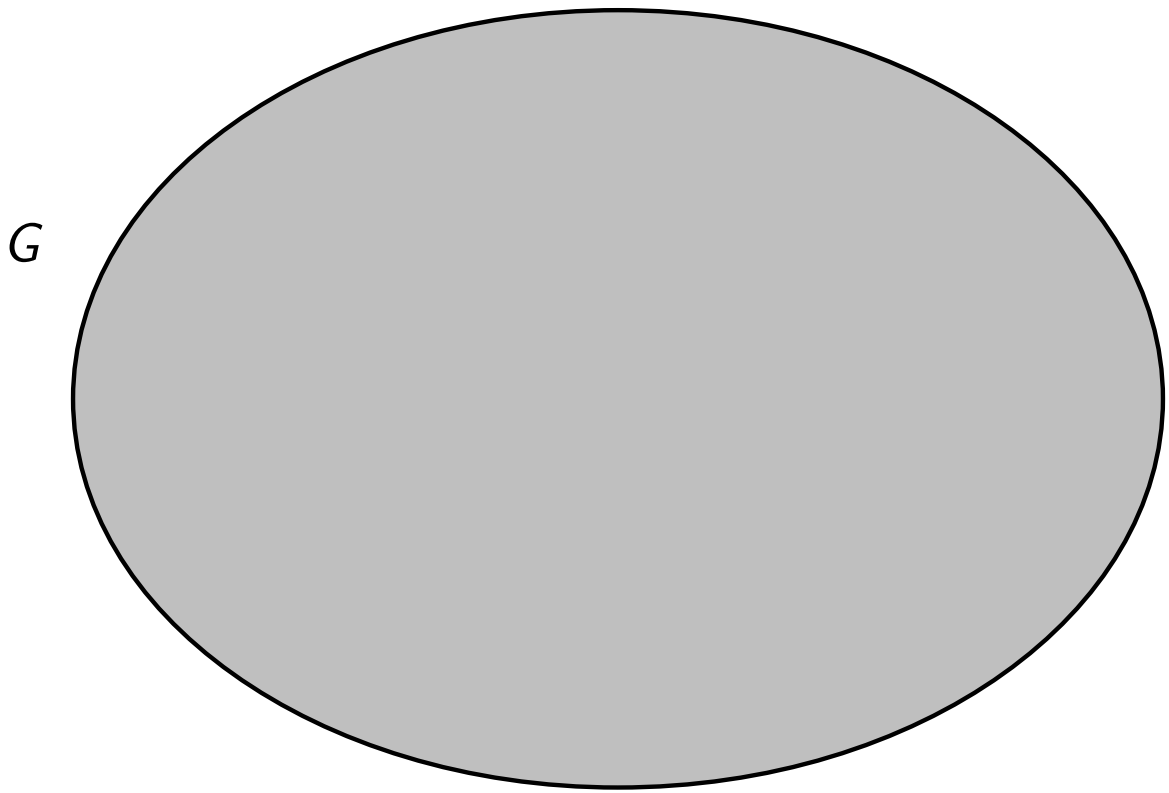


Grafo fortemente connesso

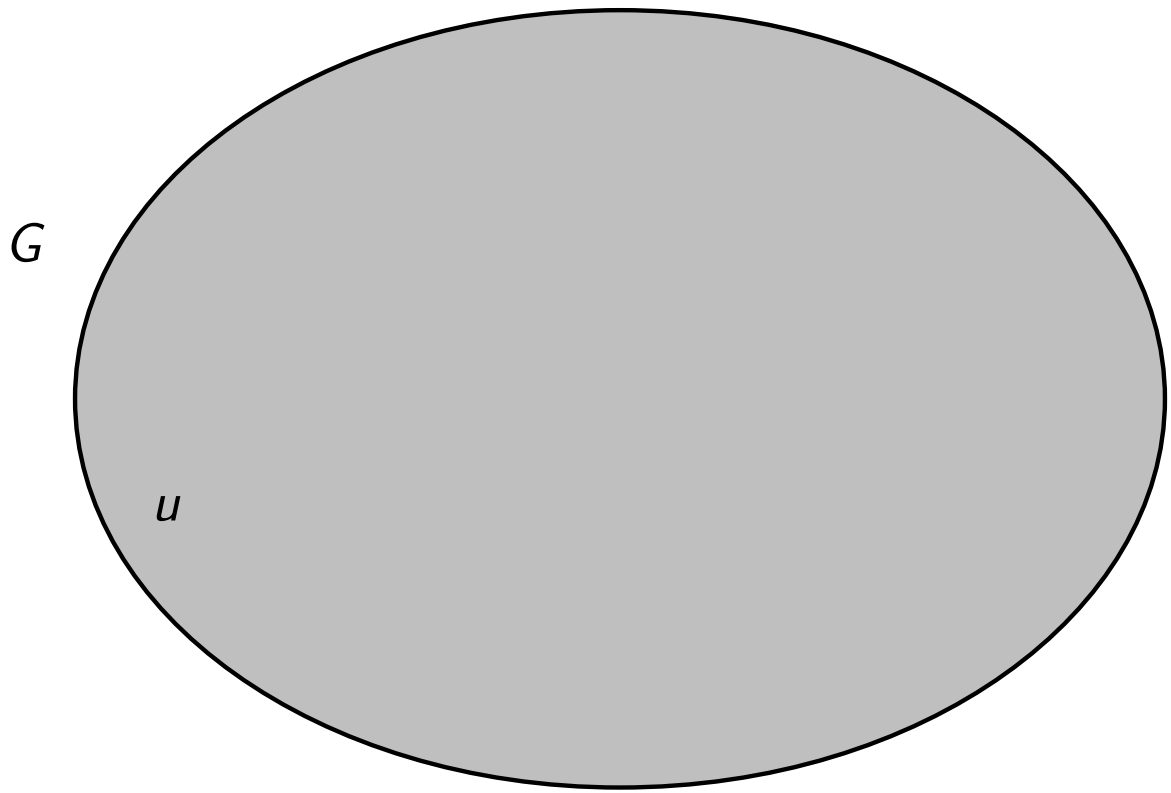


Grafo non fortemente connesso

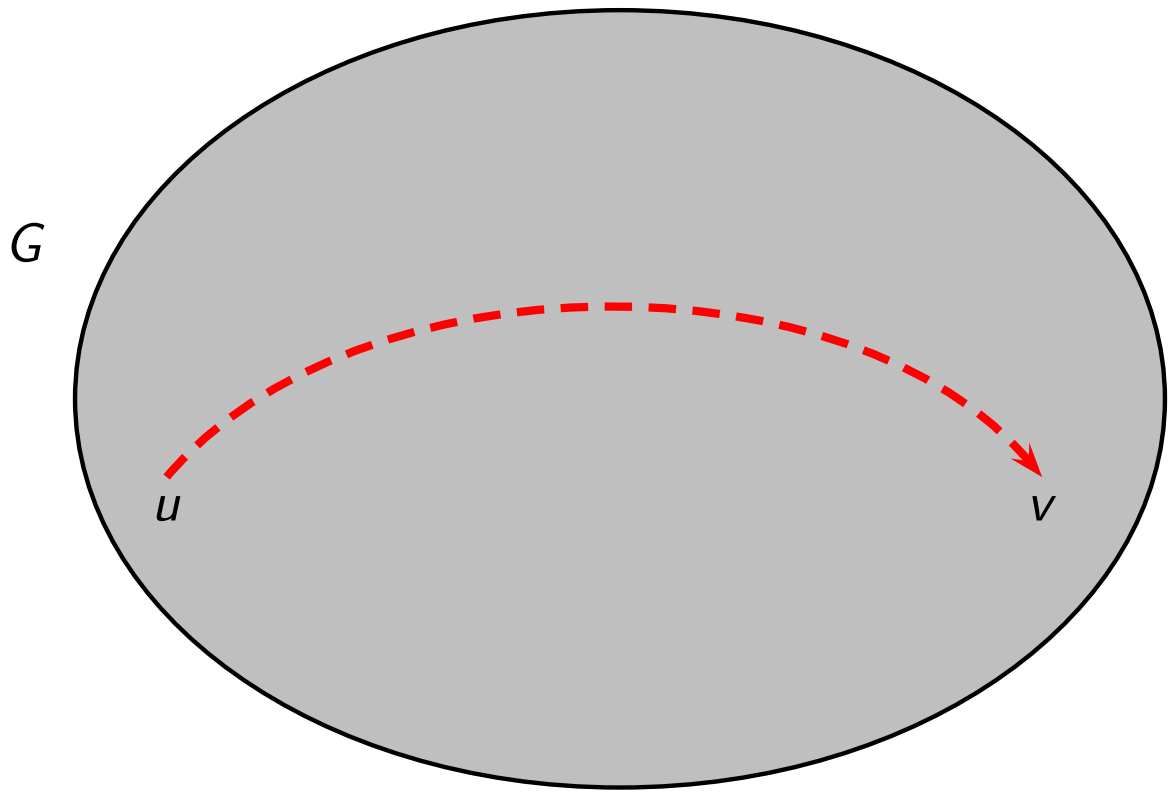
Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.



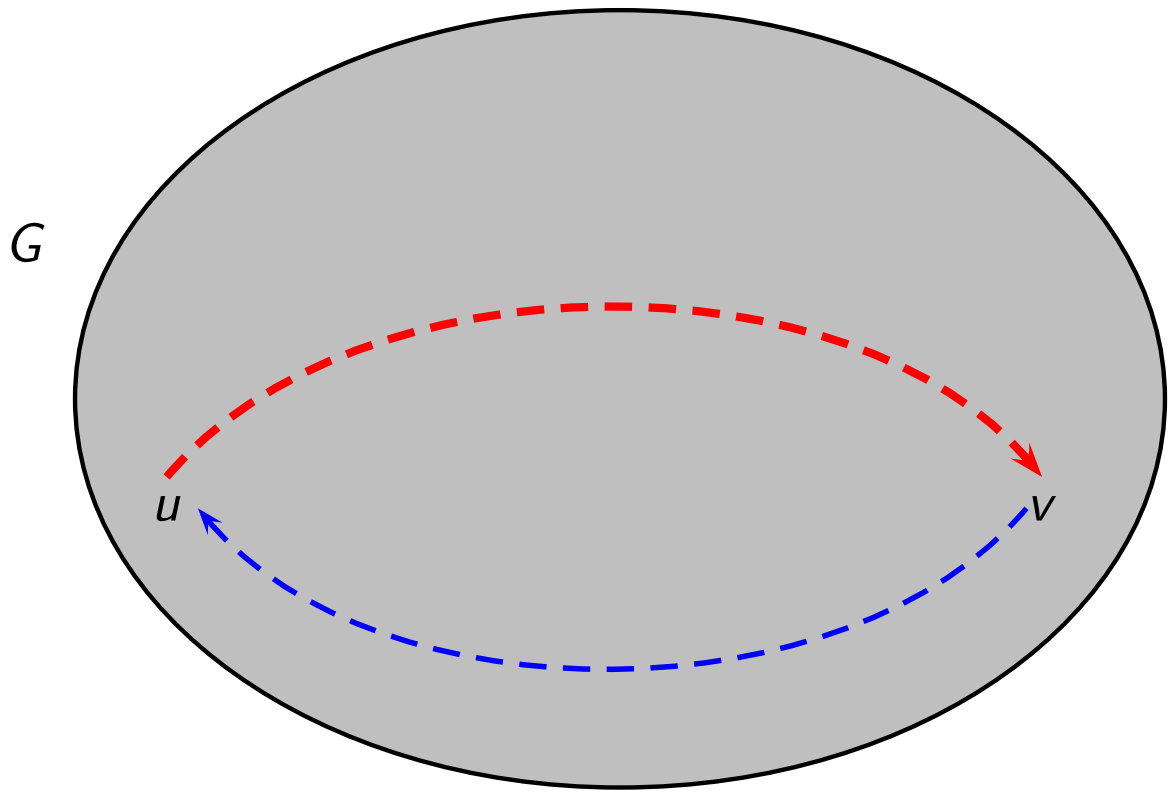
Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.



Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.



Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.



Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Controllare se un grafo è fortemente connesso applicando la definizione ci costringerebbe a verificare che, per ogni coppia di nodi u, v nel grafo, vale la proprietà che esiste un cammino diretto da u a v ed uno da v ad u.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Controllare se un grafo è fortemente connesso applicando la definizione ci costringerebbe a verificare che, per ogni coppia di nodi u, v nel grafo, vale la proprietà che esiste un cammino diretto da u a v ed uno da v ad u.

Se il grafo ha n nodi, questo vorrebbe dire che dovremmo controllare una proprietà per $\Theta(n^2)$ coppie.

Definizione: Un grafo diretto G è fortemente connesso se e solo se ogni coppia dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Controllare se un grafo è fortemente connesso applicando la definizione ci costringerebbe a verificare che, per ogni coppia di nodi u, v nel grafo, vale la proprietà che esiste un cammino diretto da u a v ed uno da v ad u.

Se il grafo ha n nodi, questo vorrebbe dire che dovremmo controllare una proprietà per $\Theta(n^2)$ coppie.

È possibile verificare la proprietà che un grafo sia, o meno, fortemente connesso verificando la proprietà *solo* su $\Theta(n)$ coppie di nodi.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y.
Occorre provare che esiste un cammino da x a y ed uno da y a x.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s,

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y e da y a s.

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y e da y a s. Quindi esiste un cammino diretto da x a y (che passa per s)

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y e da y a s. Quindi esiste un cammino diretto da x a y (che passa per s) ed un cammino diretto da y ad x (che passa per s).

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y e da y a s. Quindi esiste un cammino diretto da x a y (che passa per s) ed un cammino diretto da y ad x (che passa per s). Quindi ogni coppia di nodi x e y è mutualmente raggiungibile

Fatto 4. Sia s un qualsiasi *fissato* nodo di G. Abbiamo che G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da qualsiasi nodo u di G.

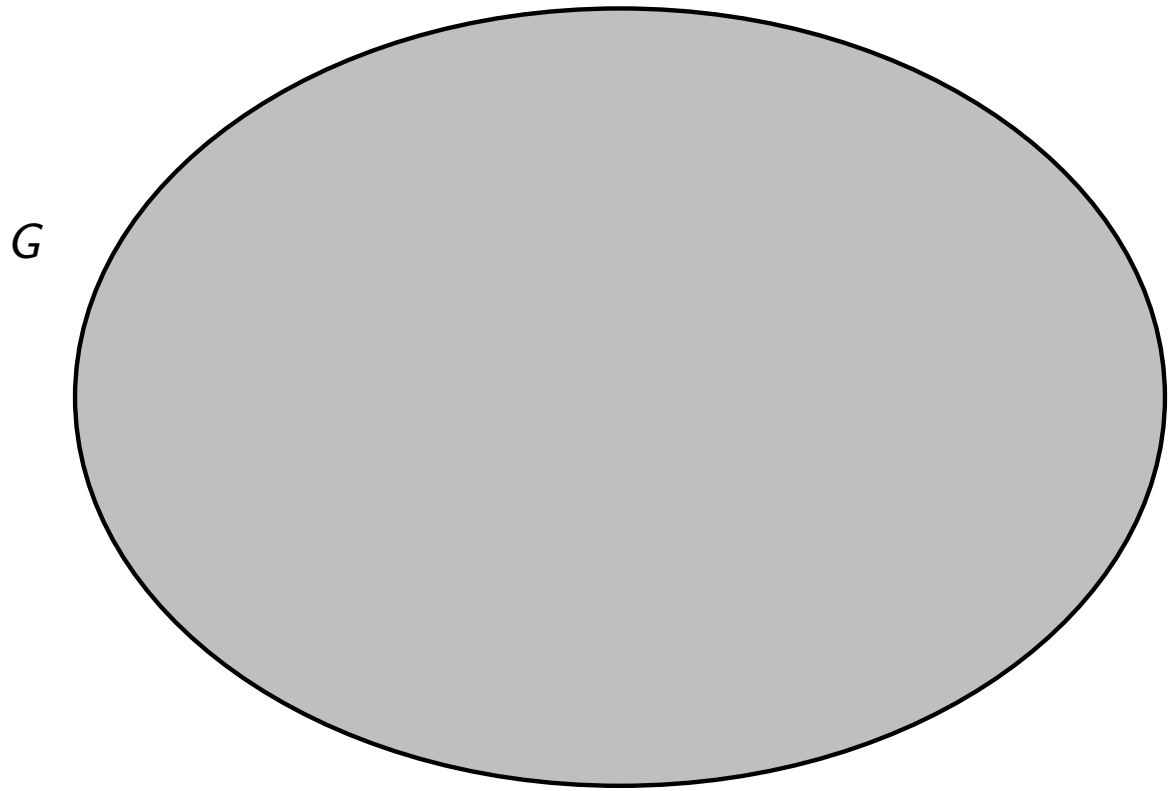
L'implicazione $\Rightarrow$ segue dalla definizione di grafo fortemente connesso.

Per provare l'impicazione $\Leftarrow$, consideriamo due *arbitrari* nodi x e y. Occorre provare che esiste un cammino da x a y ed uno da y a x. Per ipotesi esiste un cammino diretto da s a x e da x a s, ed esiste anche un cammino diretto da s a y e da y a s. Quindi esiste un cammino diretto da x a y (che passa per s) ed un cammino diretto da y ad x (che passa per s). Quindi ogni coppia di nodi x e y è mutualmente raggiungibile e di conseguenza G è fortemente connesso.

Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

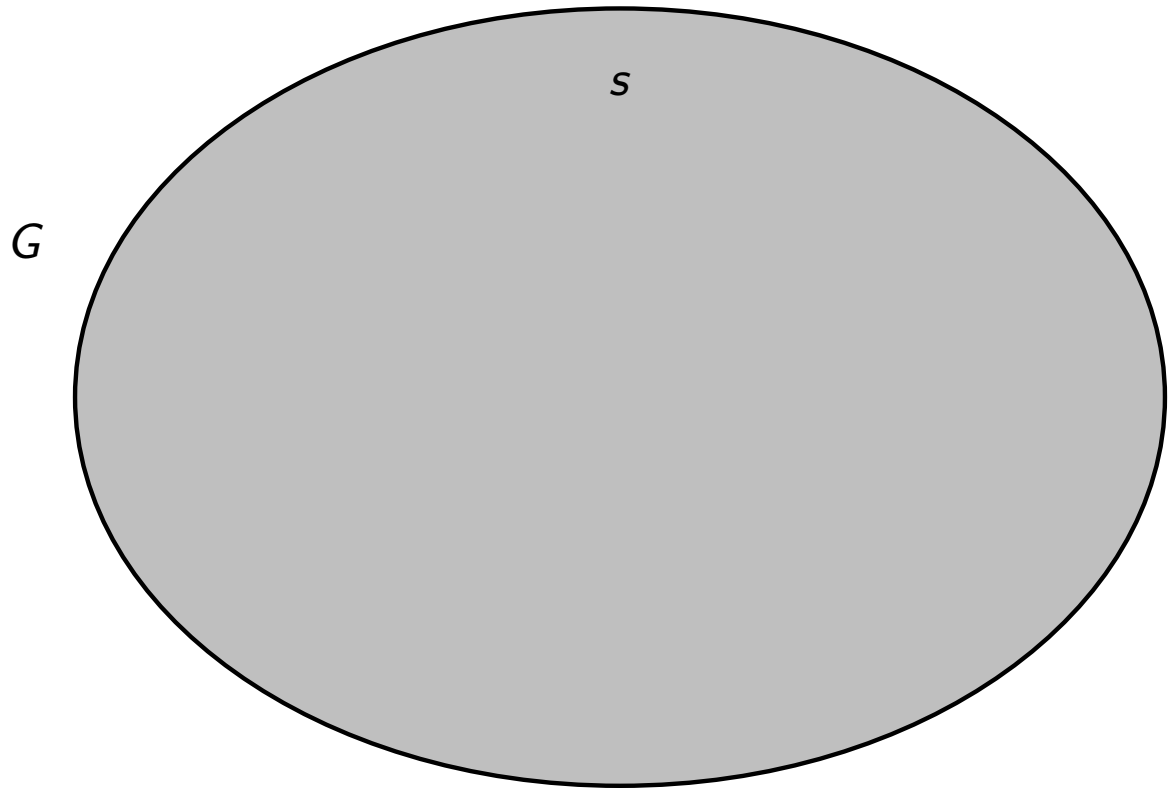
Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



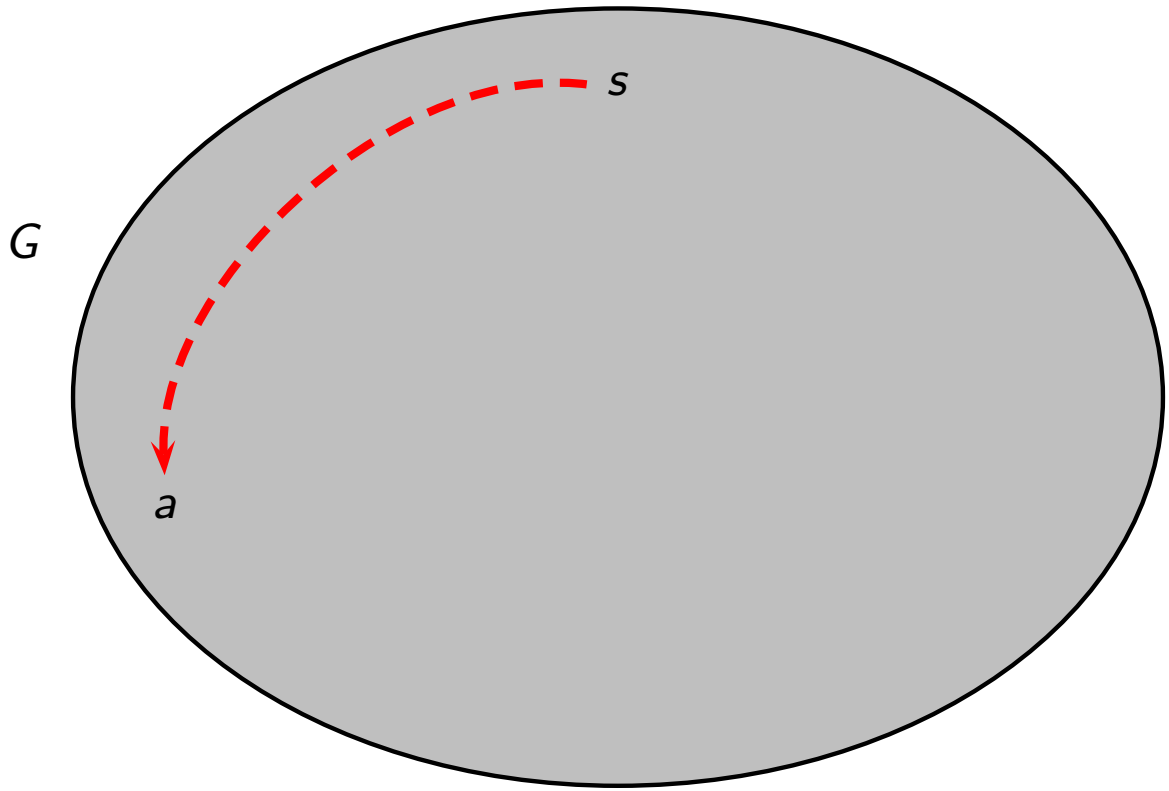
Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



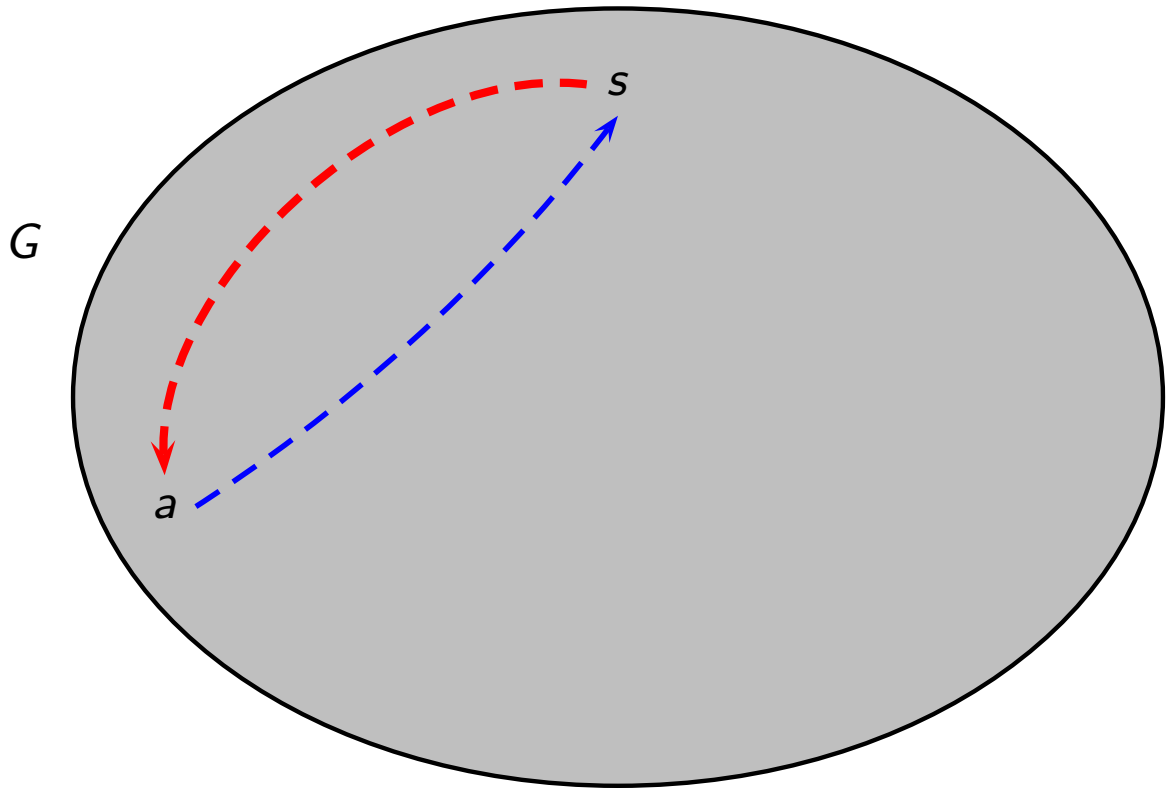
Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



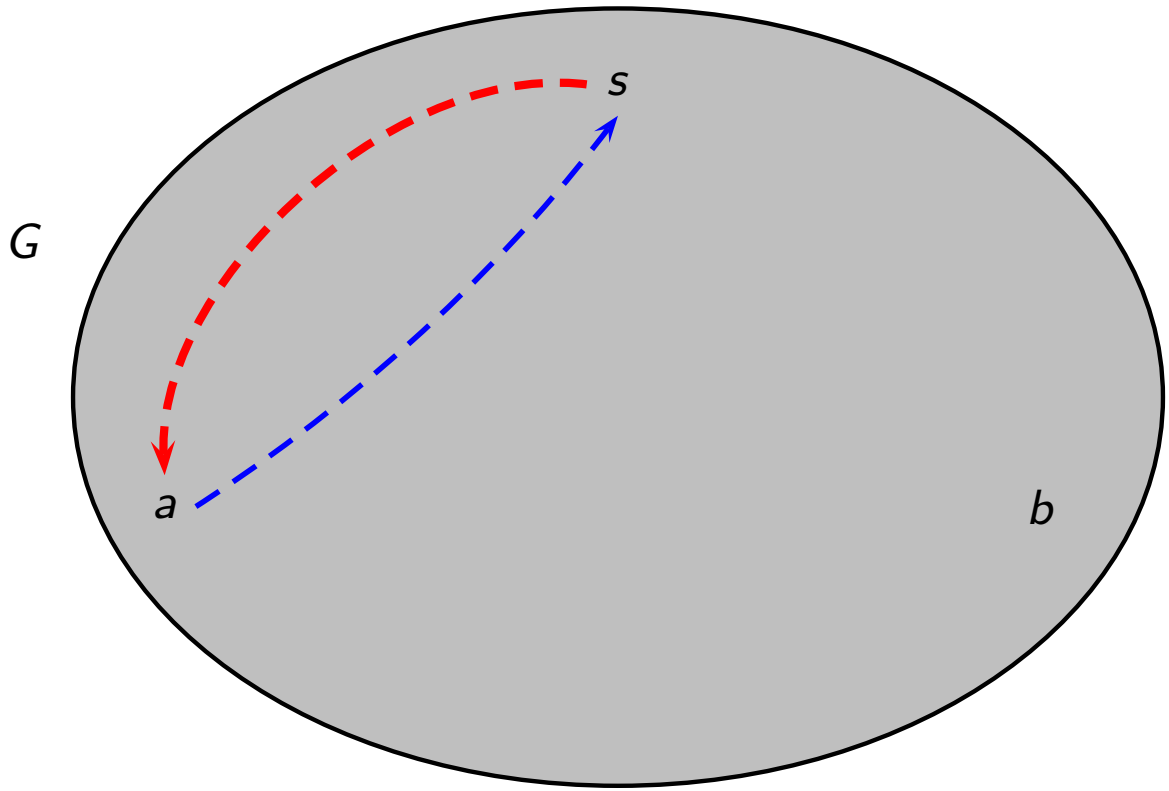
Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



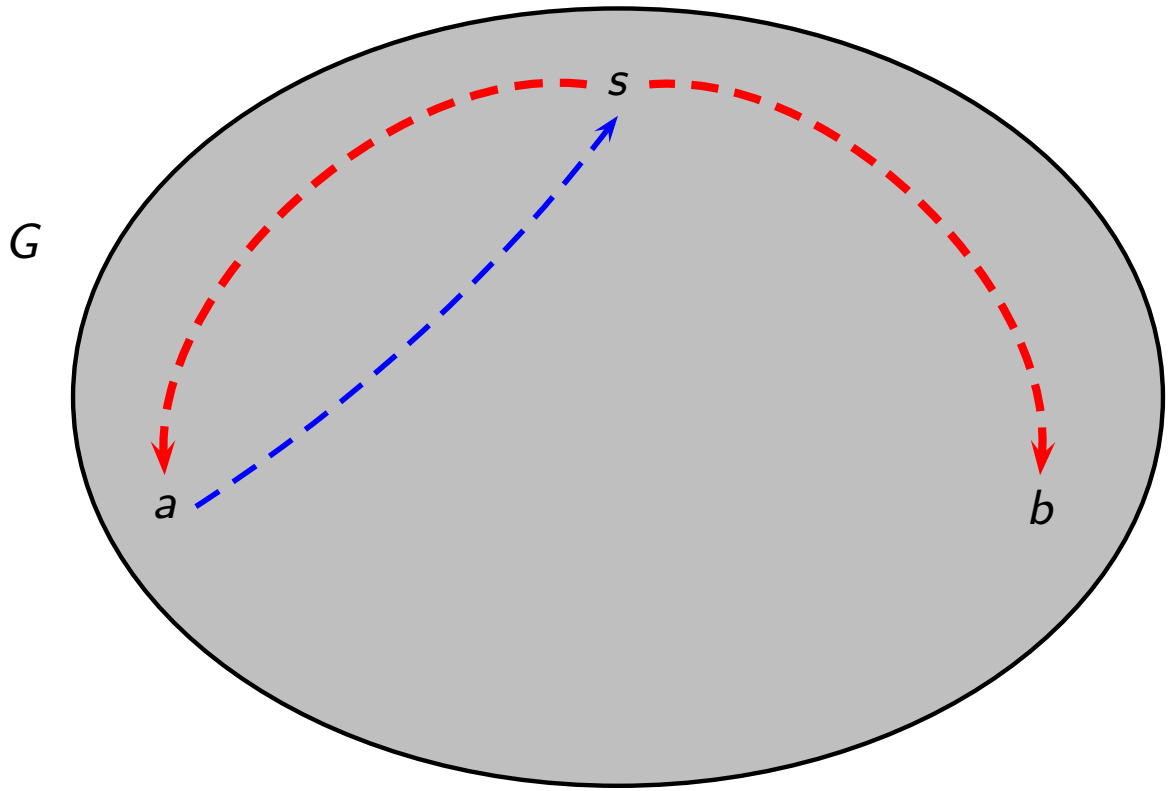
Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.



Definizione: Il grafo diretto G è fortemente connesso se e solo se *ogni coppia* dei suoi nodi è mutualmente raggiungibile, ovvero se per ogni sua coppia di nodi u, v, esiste un cammino diretto da u e v ed un cammino diretto da v ad u.

Fatto. Sia s un qualsiasi *fissato* nodo di G. G è fortemente connesso $\Leftrightarrow$ ogni nodo u in G è raggiungibile da s ed inoltre s è raggiungibile da un qualsiasi nodo u di G.