ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

Visita DFS su grafi non orientati
e sue applicazioni

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Visita di grafi

Strategia per **analizzare i nodi del grafo** una volta sola

VISITA IN PROFONDITÀ (**Depth First Search - DFS**)

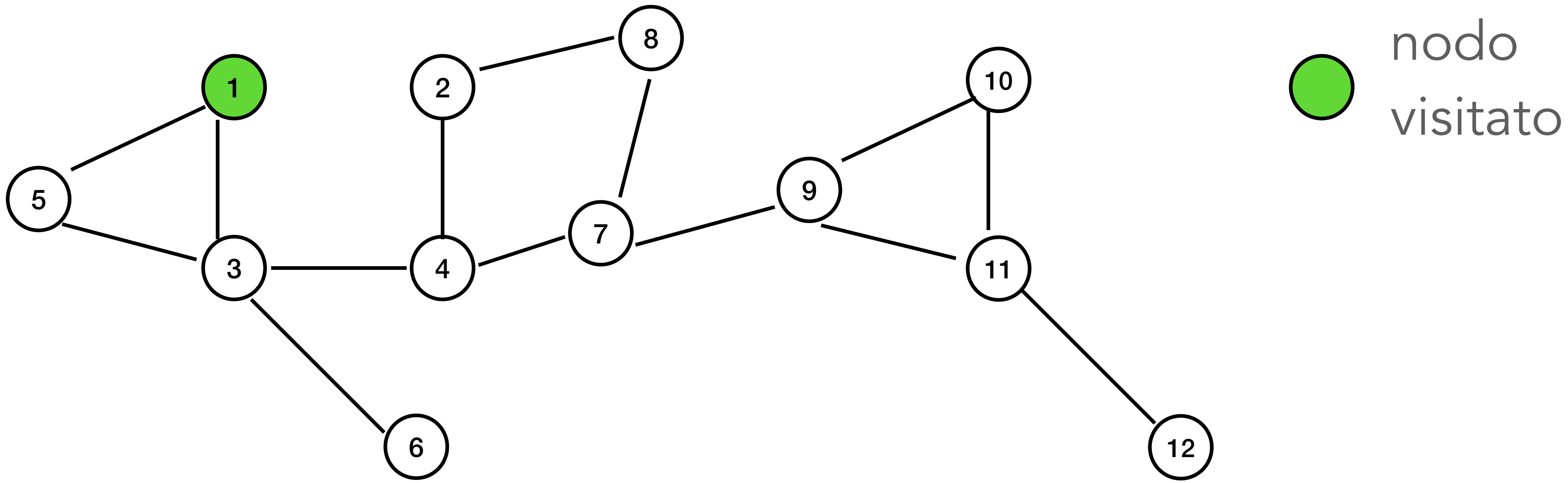
- Estensione della pre/post-order DFS sugli alberi:
  dopo aver visitato un nodo si continua la visita cercando "di allontanarsi" sempre più.
- Strategia utilizzata per visitare tutti i nodi del grafo, anche se ci sono più componenti connesse

VISITA IN AMPIEZZA (**Breadth First Search - BFS**)

- Estensione della BFS sugli alberi:
  partendo da un nodo, si visitano tutti i suoi vicini, poi i vicini dei vicini, e cosi' via.
- Strategia utilizzata per visitare tutti i nodi raggiungibili da un nodo sorgente

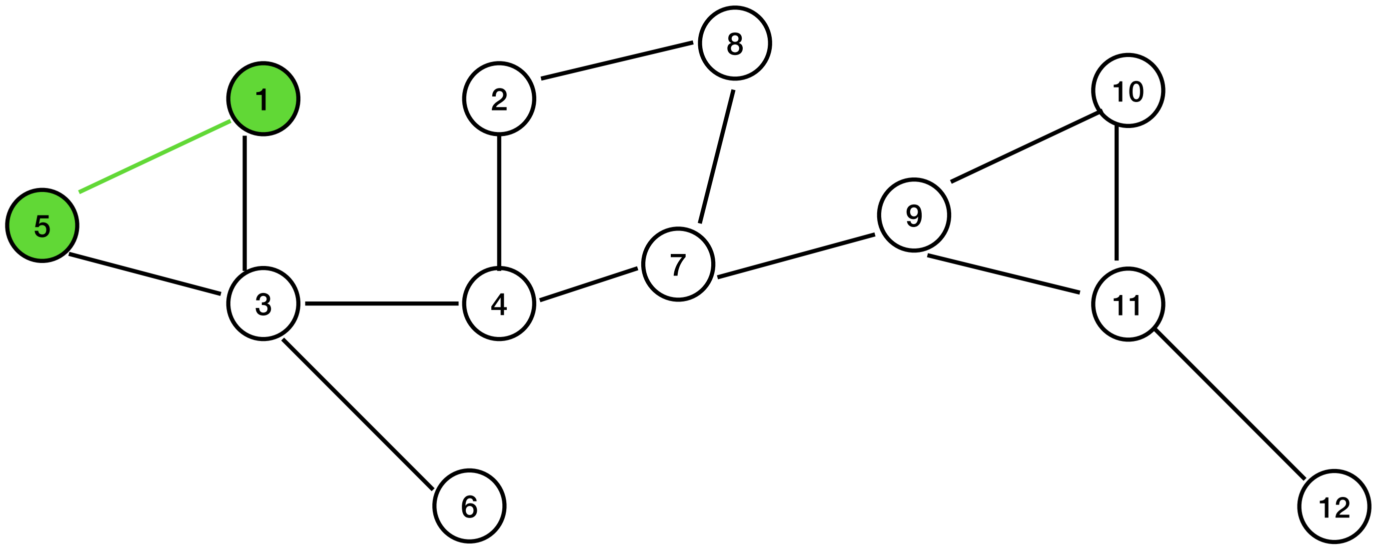
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO

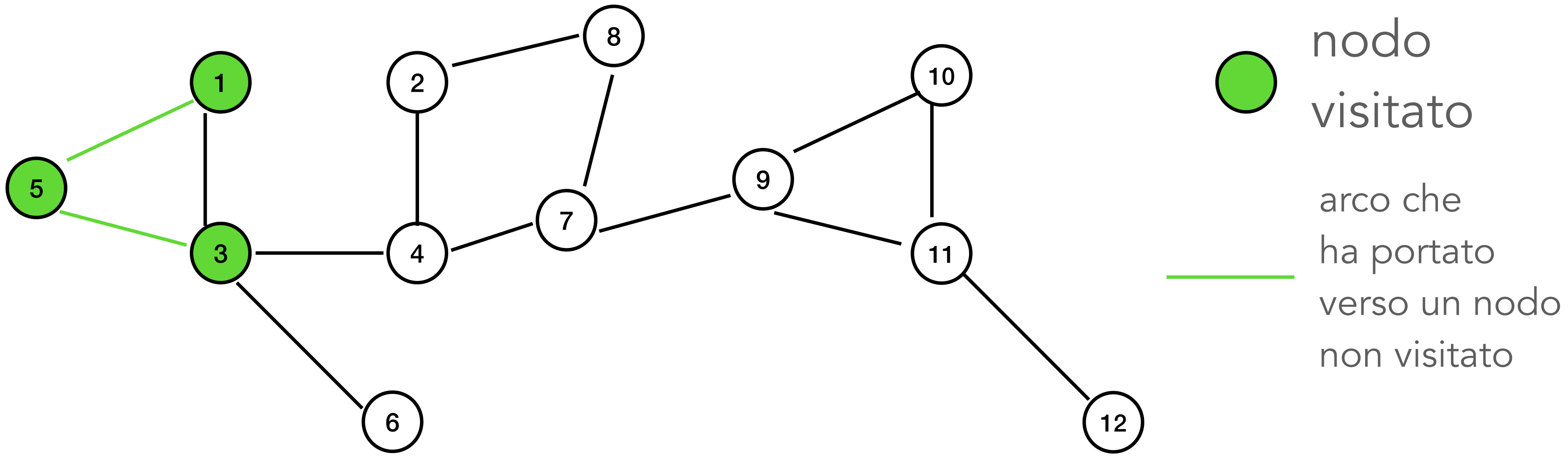


nodo visitato

arco che ha portato verso un nodo non visitato

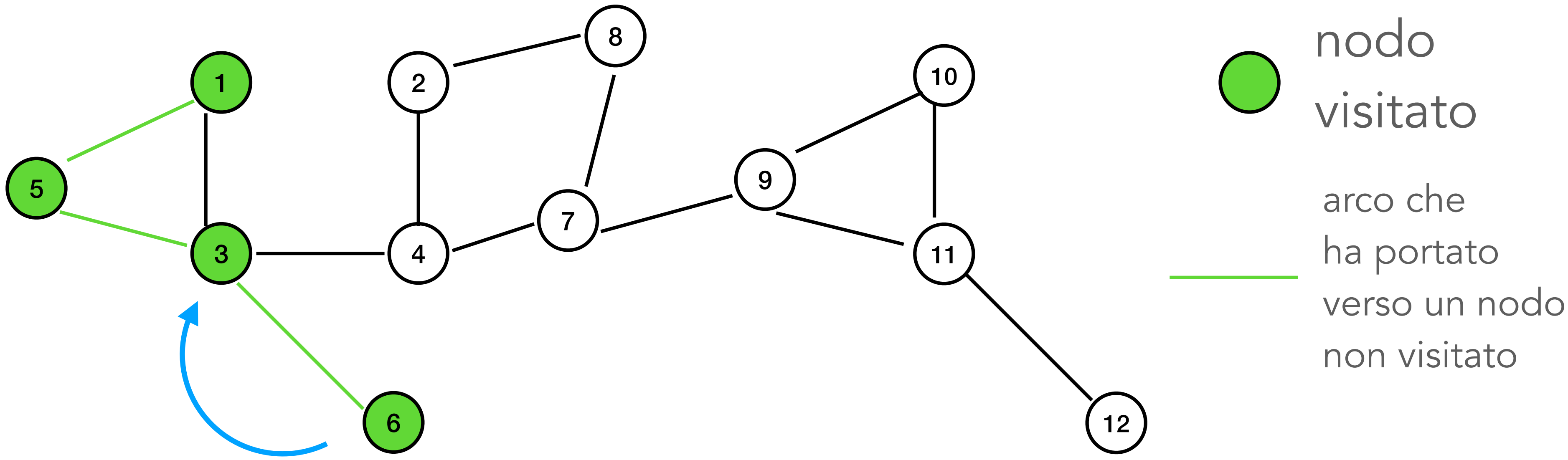
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

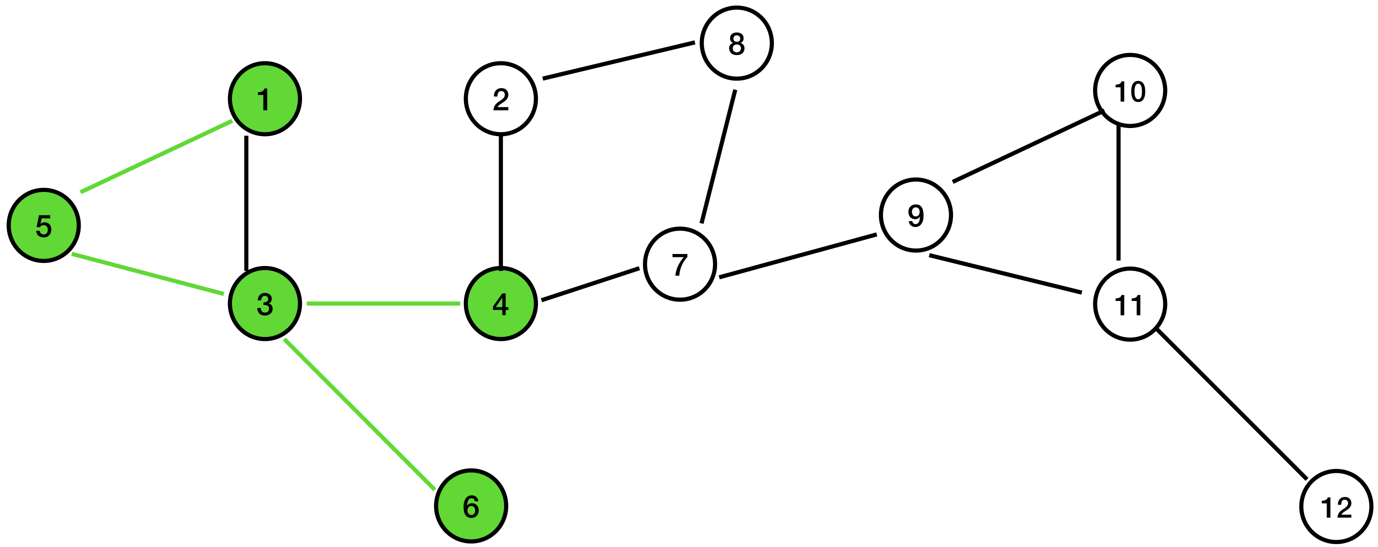
ESEMPIO - GRAFO NON ORIENTATO



Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

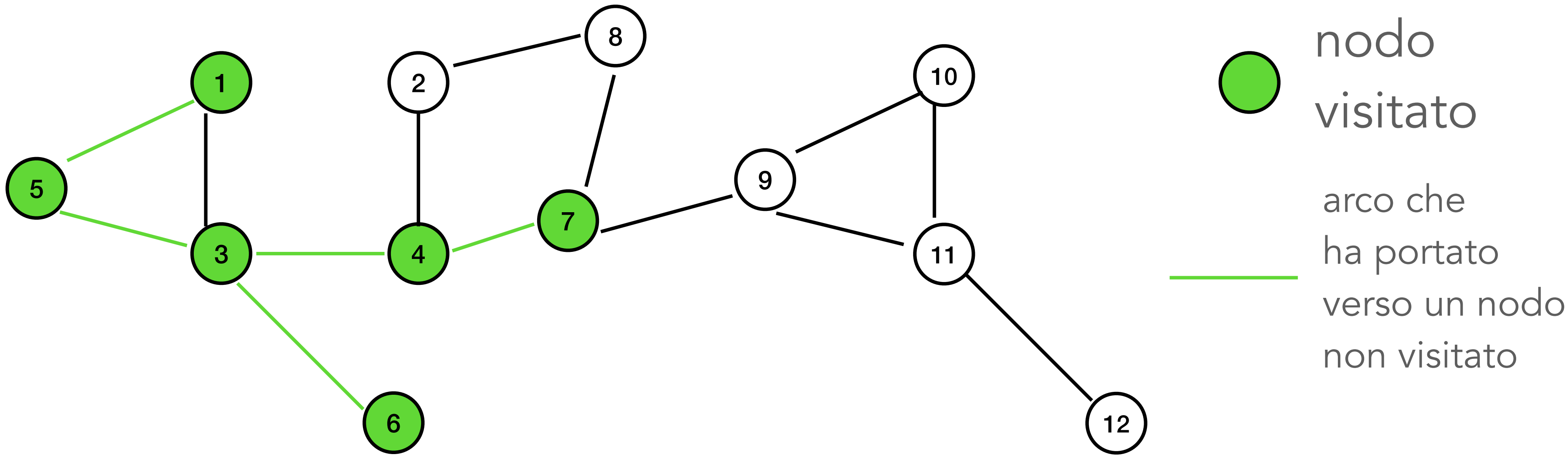
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



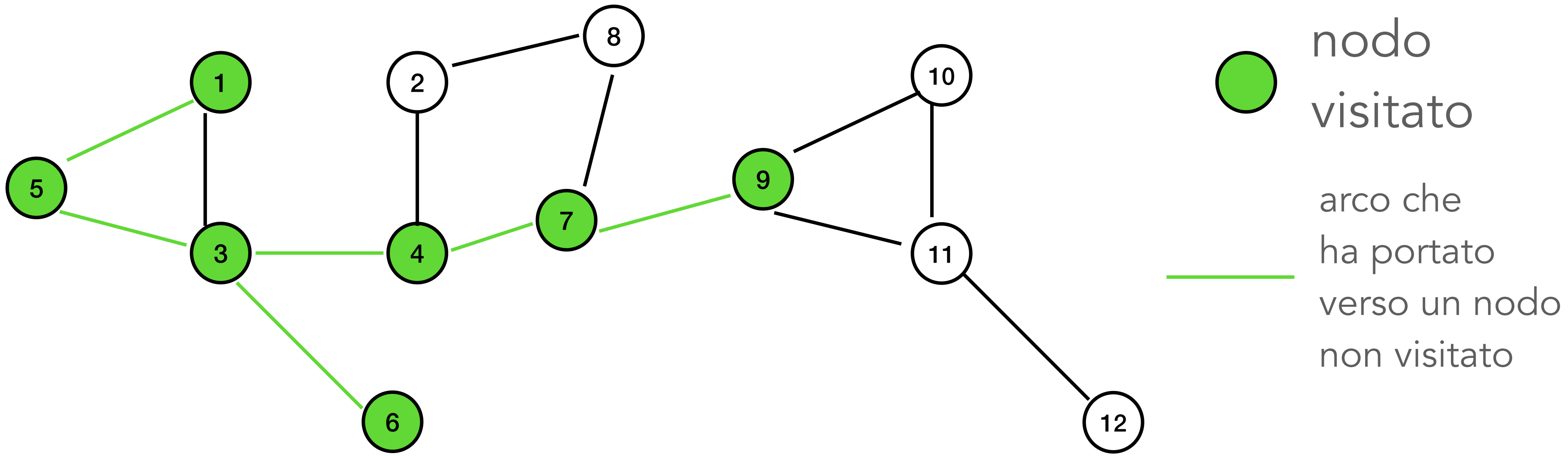
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



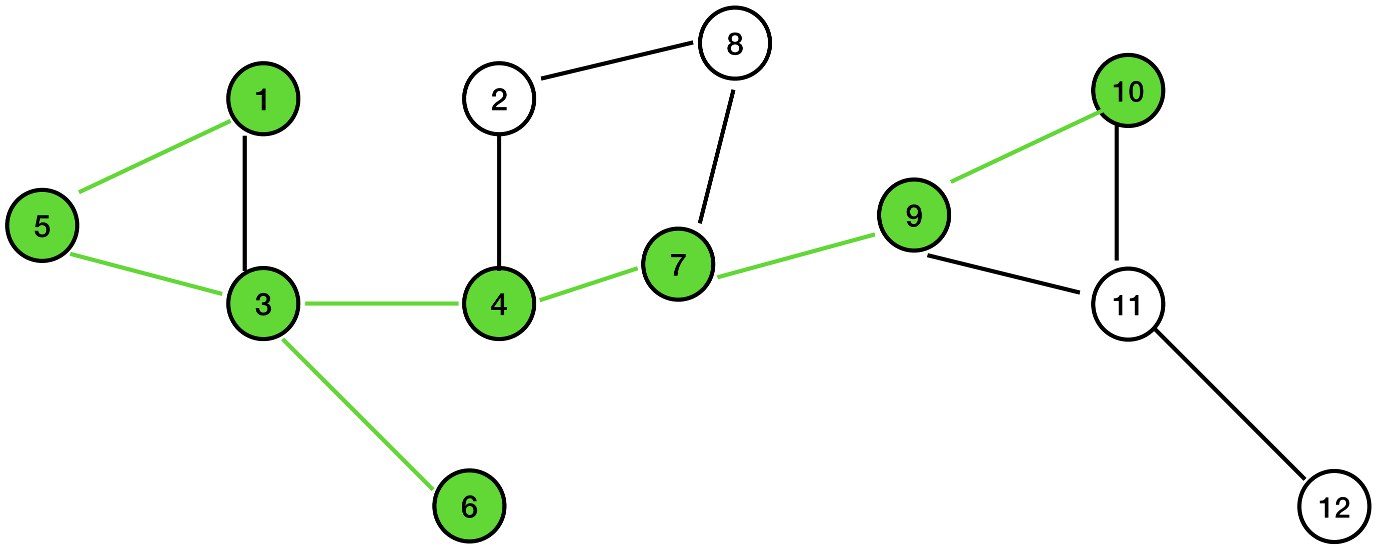
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO

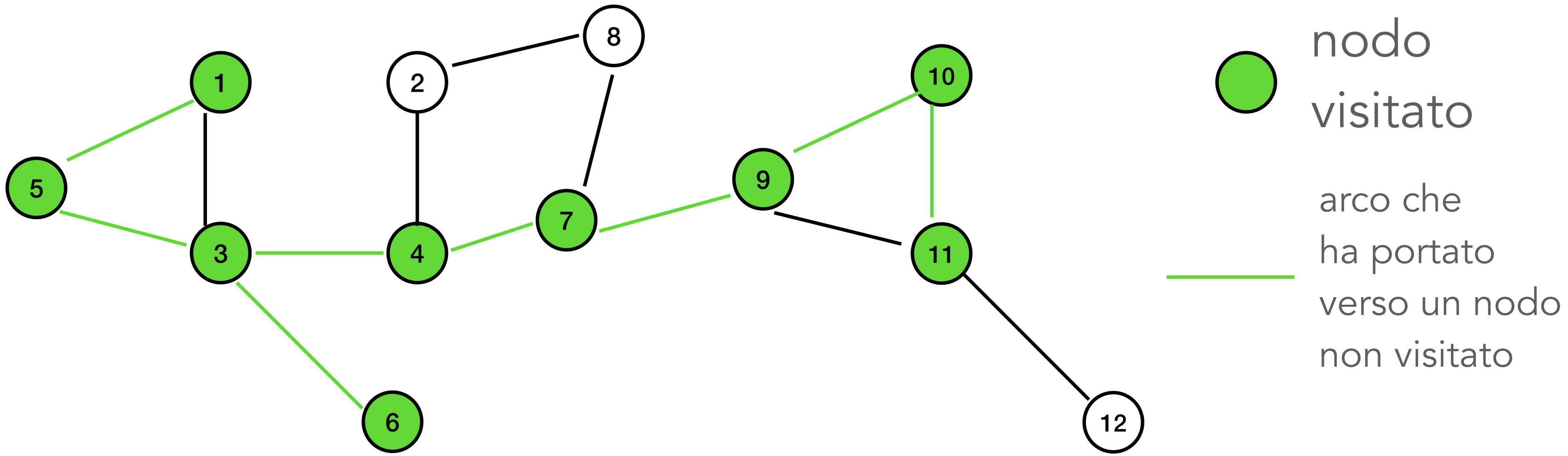


nodo visitato

arco che ha portato verso un nodo non visitato

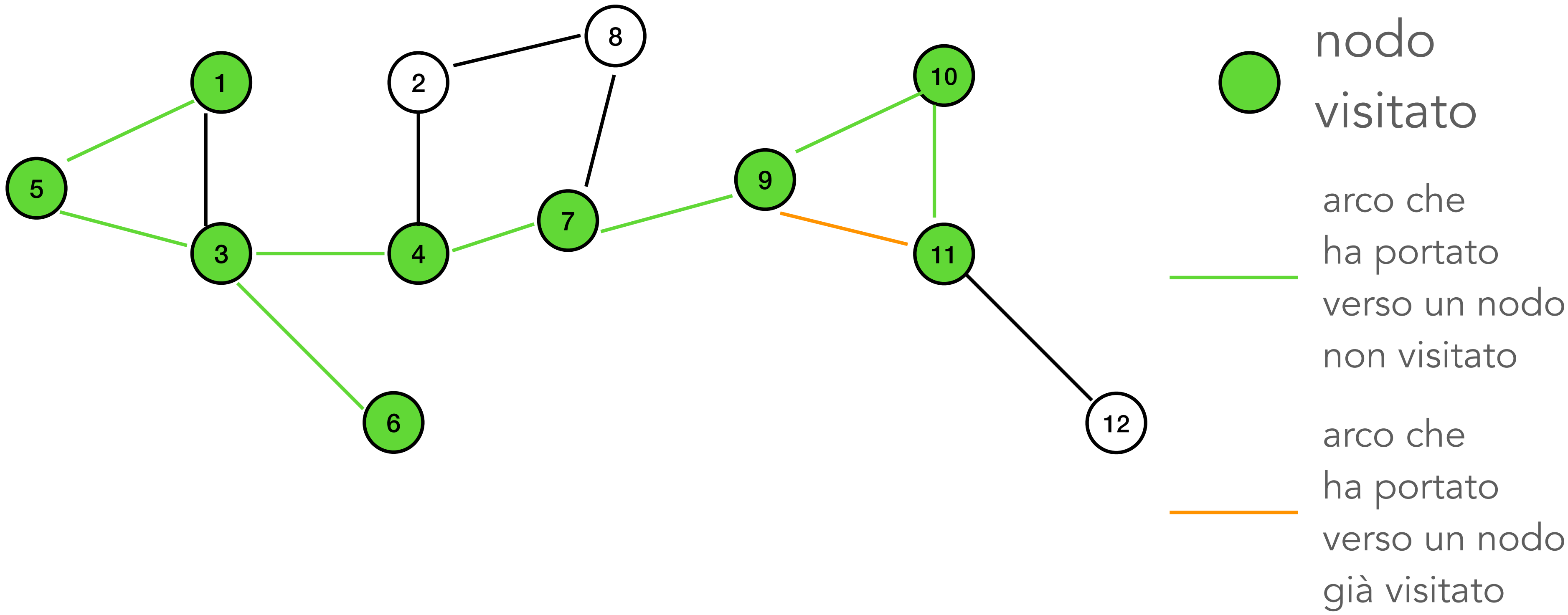
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

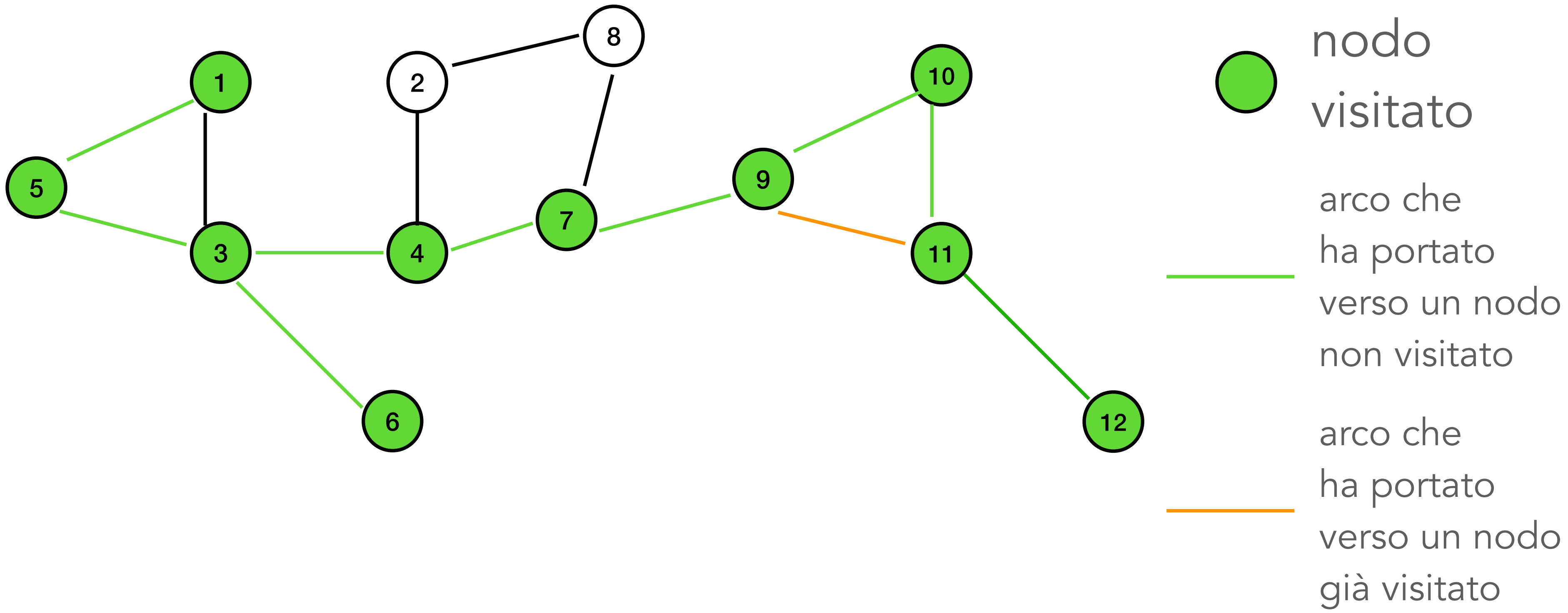
ESEMPIO - GRAFO NON ORIENTATO



UNIMORE

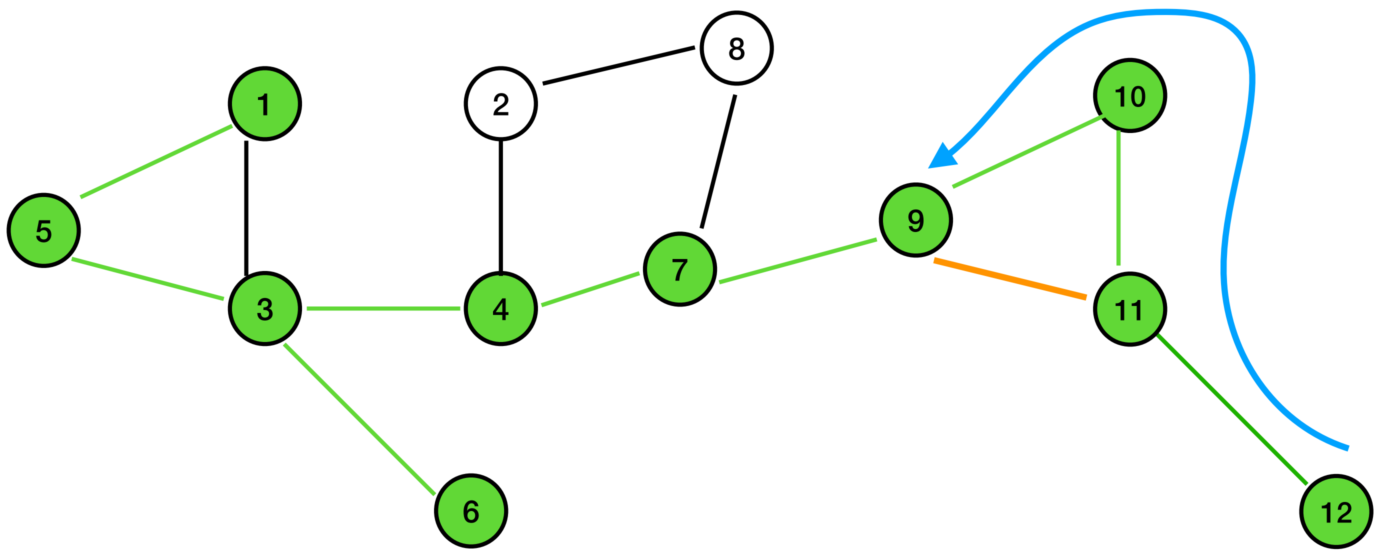
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



nodo visitato

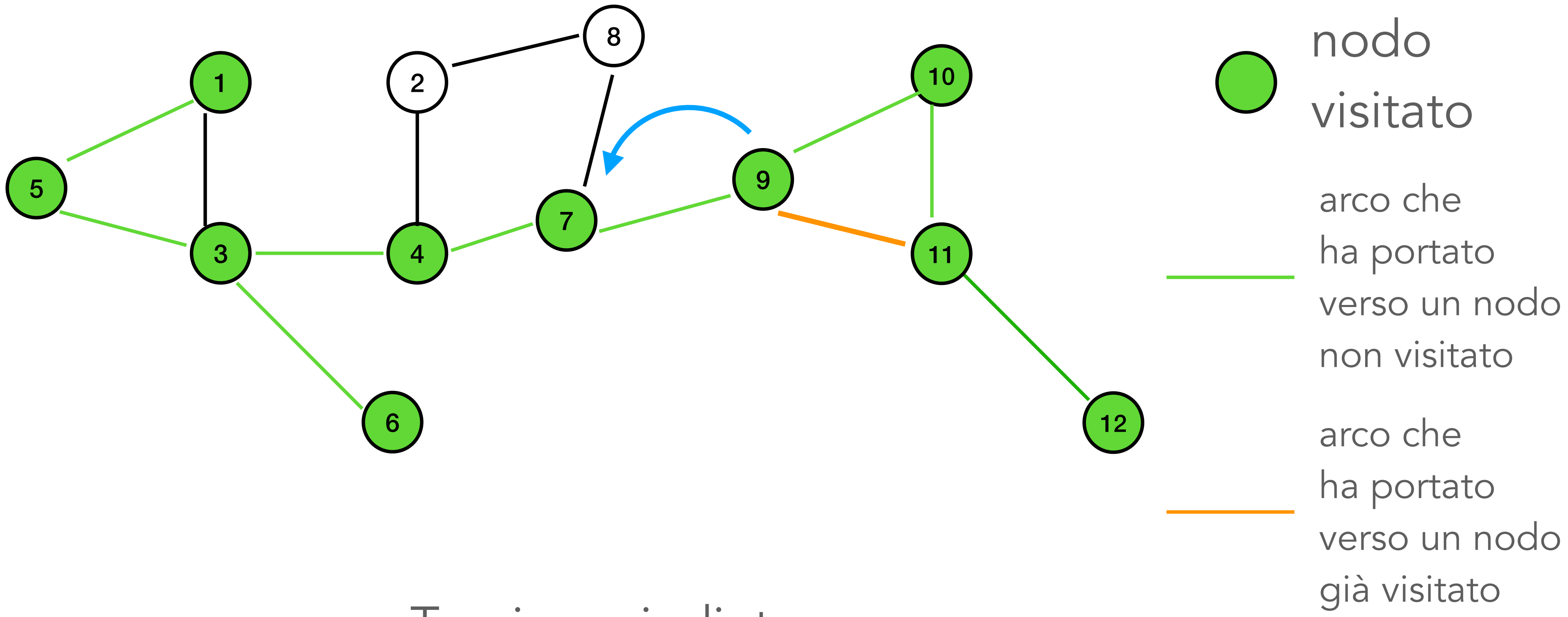
arco che ha portato verso un nodo non visitato

arco che ha portato verso un nodo già visitato

Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

VISITA in PROFONDITA' (DFS)

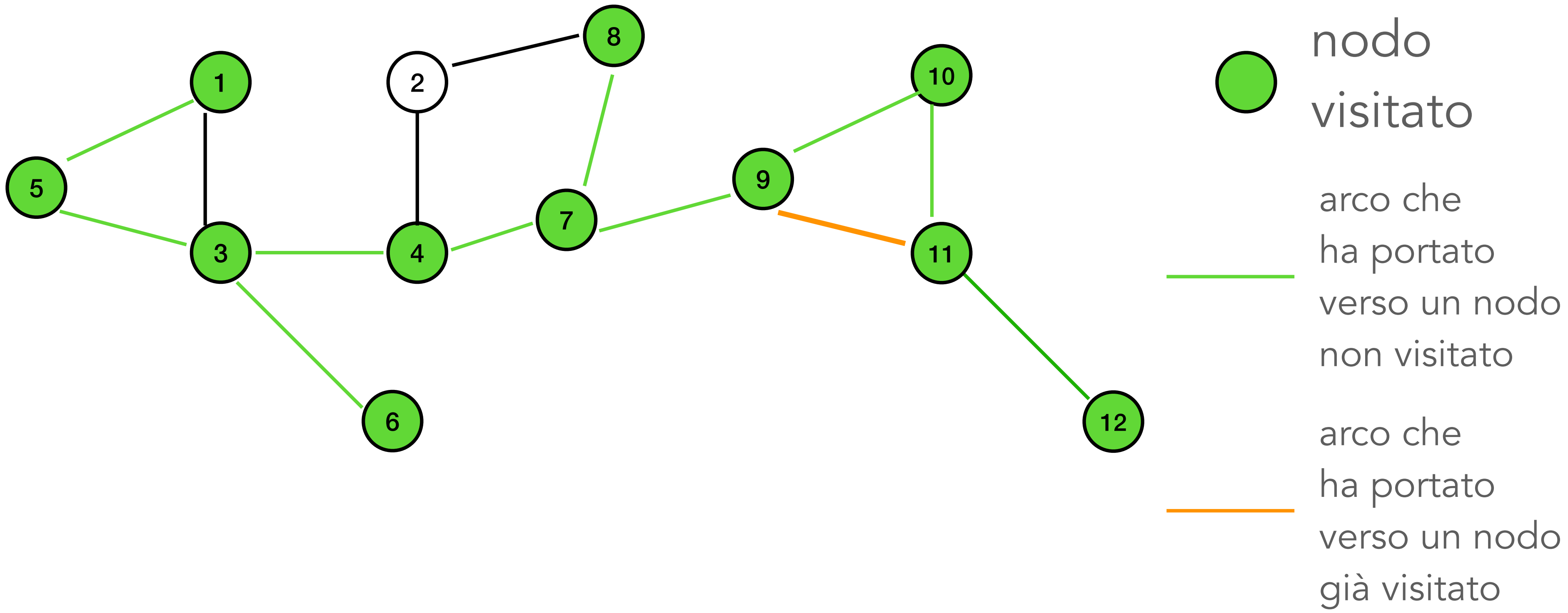
ESEMPIO - GRAFO NON ORIENTATO



Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

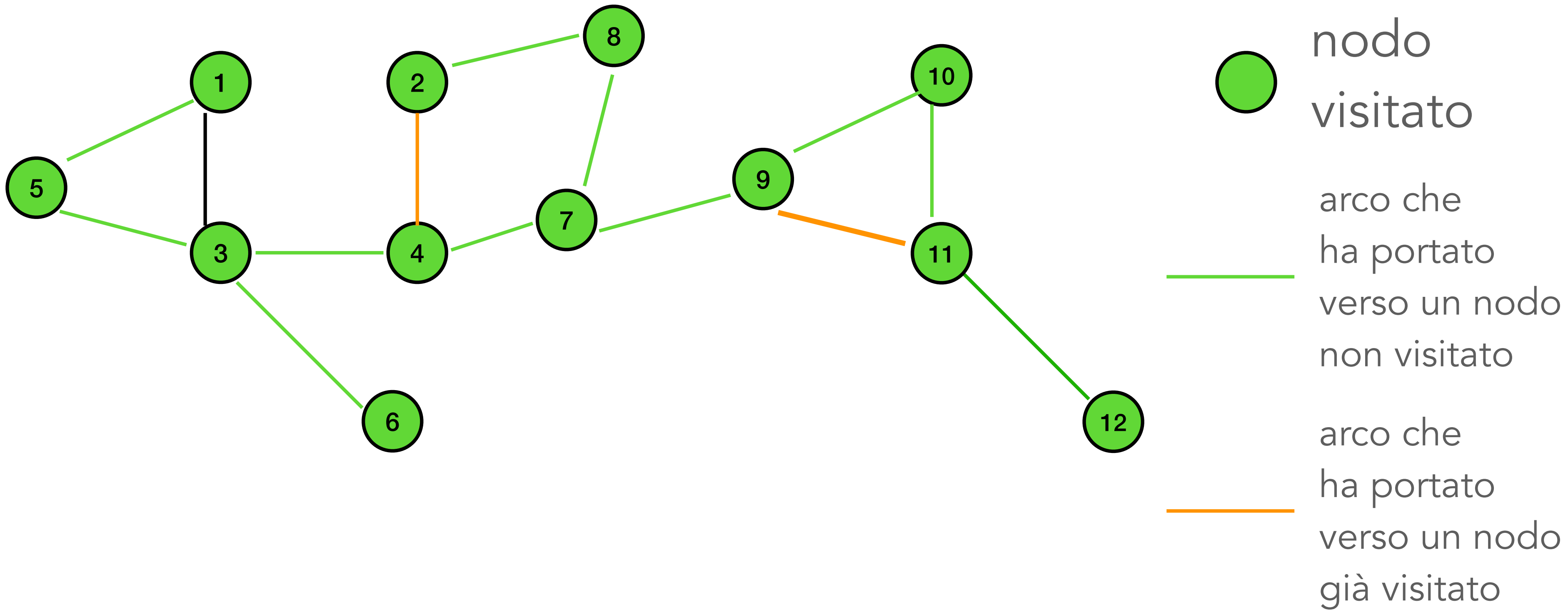
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



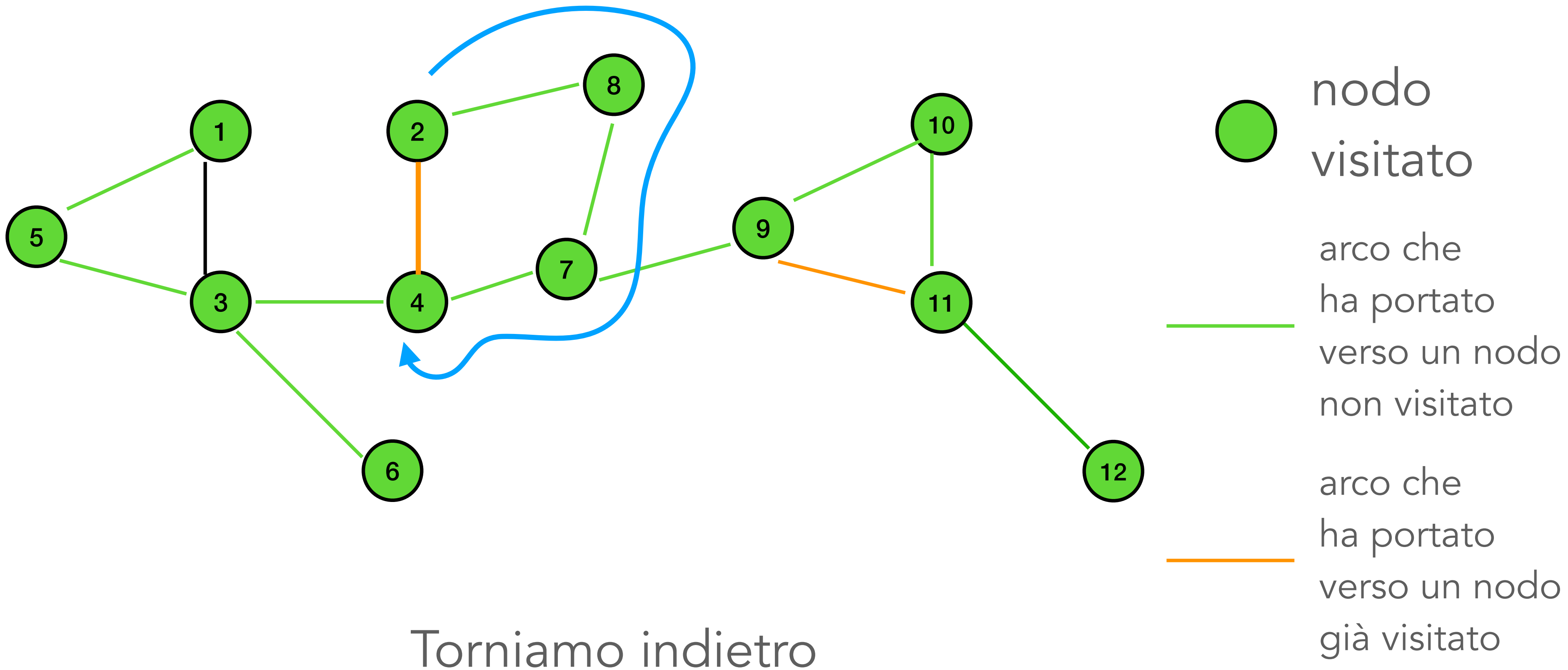
VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



VISITA in PROFONDITA' (DFS)

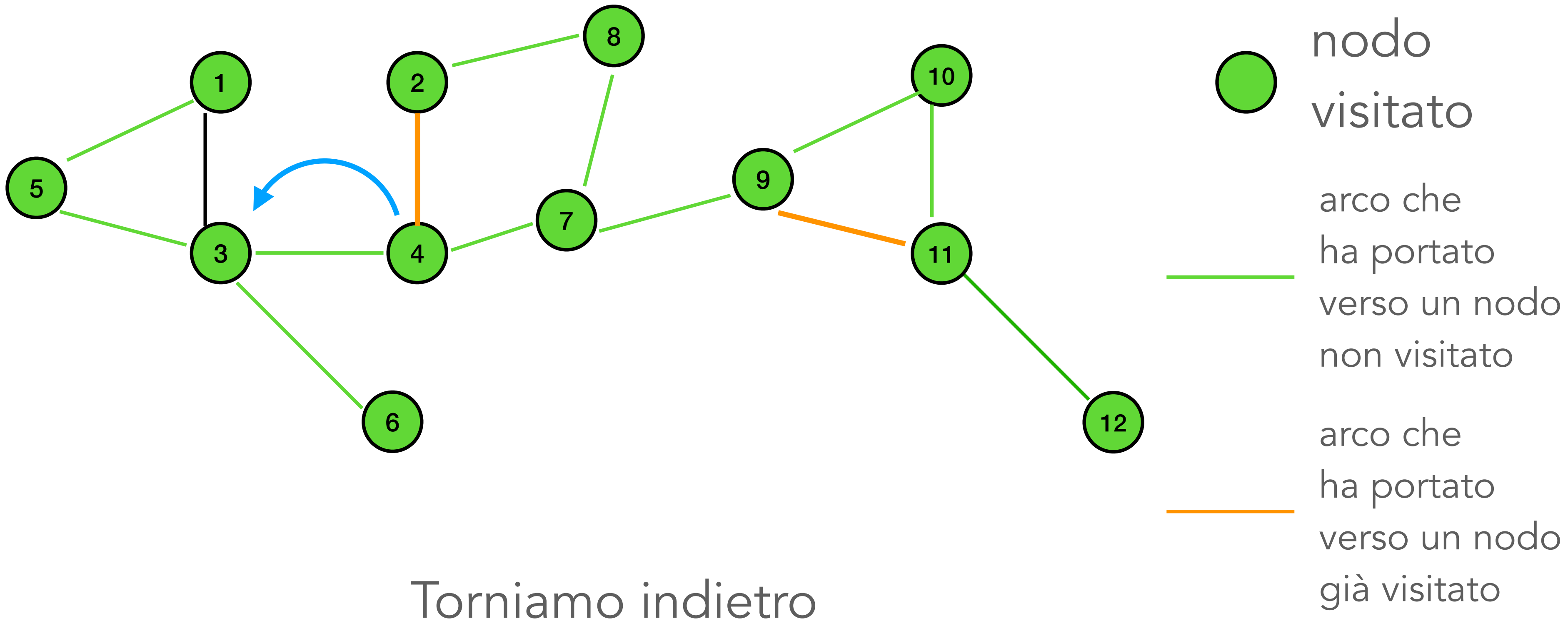
ESEMPIO - GRAFO NON ORIENTATO



Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

VISITA in PROFONDITA' (DFS)

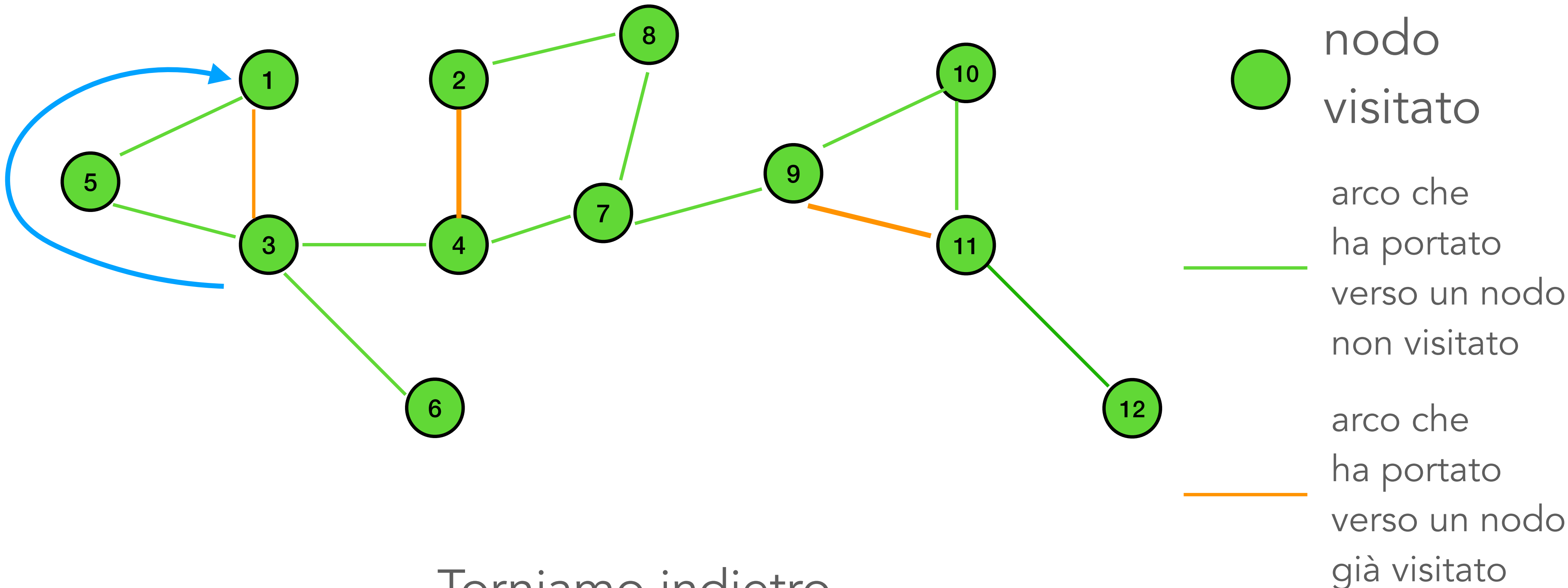
ESEMPIO - GRAFO NON ORIENTATO



Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

VISITA in PROFONDITA' (DFS)

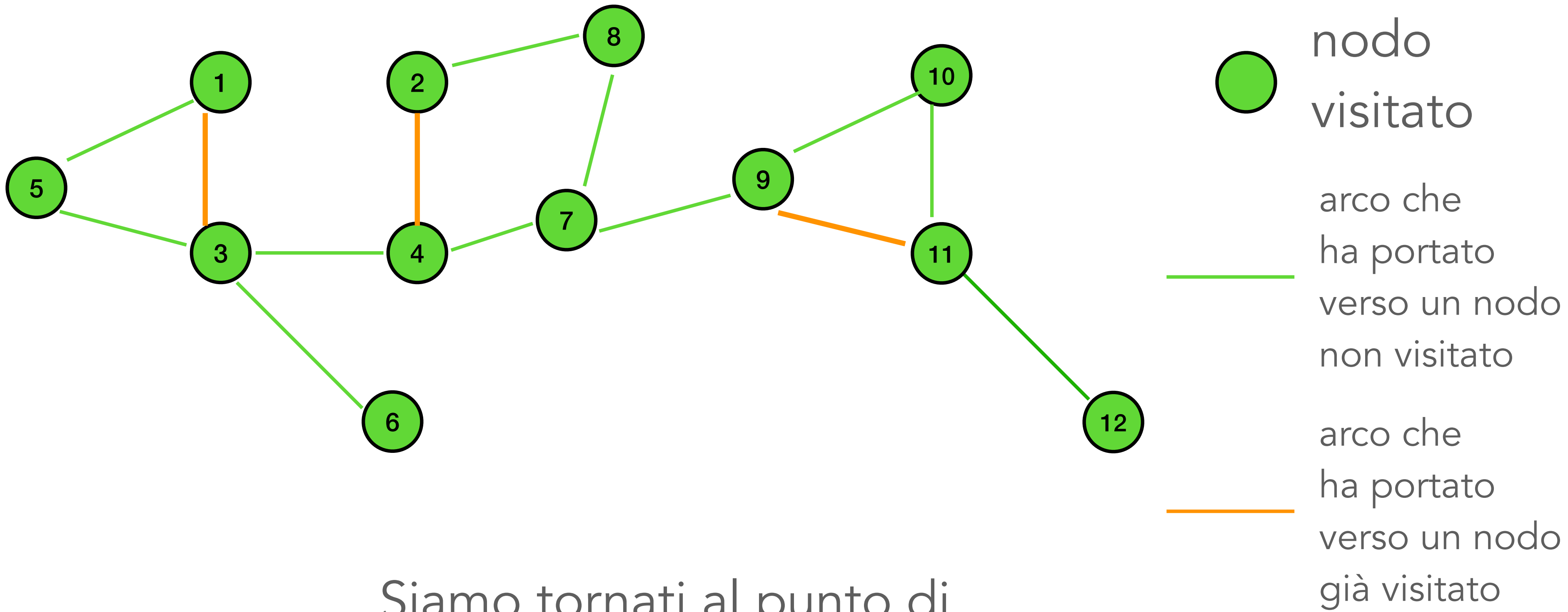
ESEMPIO - GRAFO NON ORIENTATO



Torniamo indietro
(seguendo a ritroso il percorso dell'andata)
fino ad arrivare ad un nodo con ancora un vicino inesplorato

VISITA in PROFONDITA' (DFS)

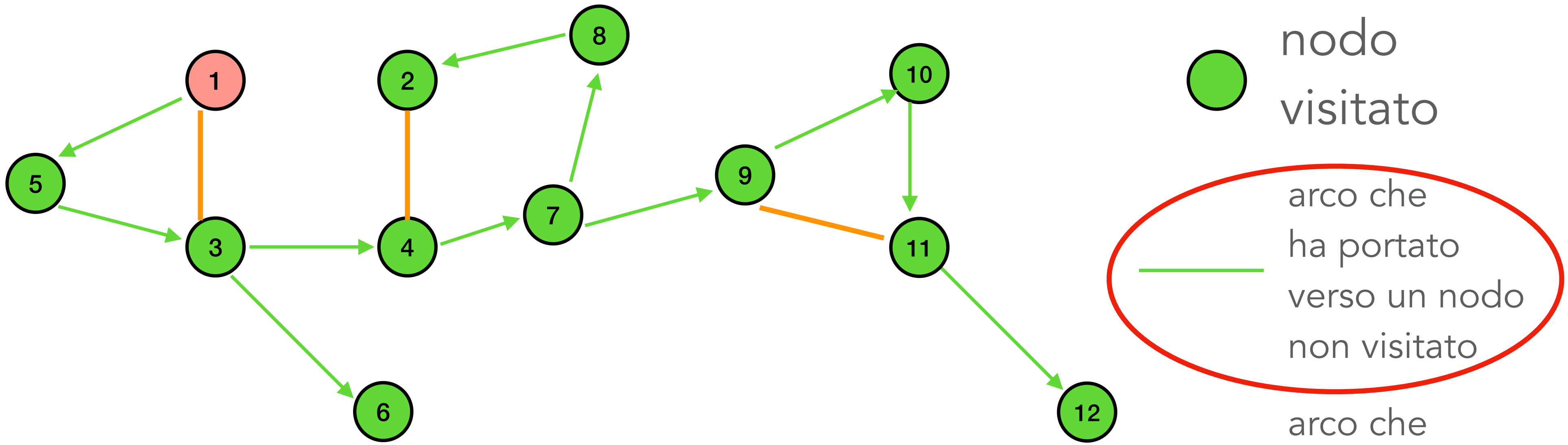
ESEMPIO - GRAFO NON ORIENTATO



Siamo tornati al punto di partenza e non ci sono altri archi da esplorare

VISITA in PROFONDITA' (DFS)

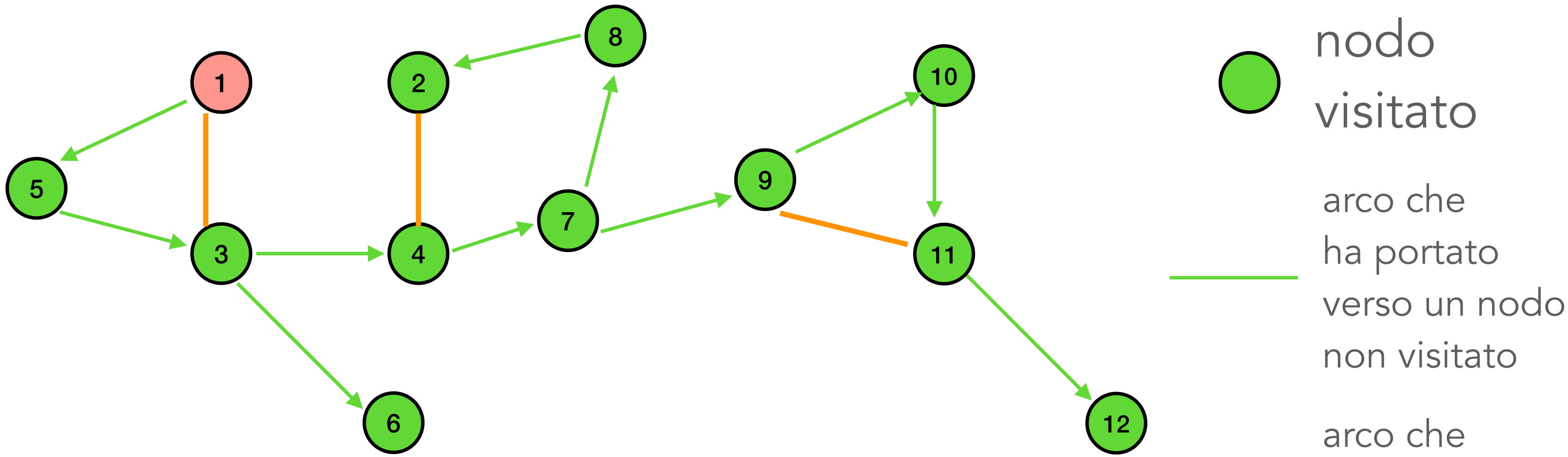
ESEMPIO - GRAFO NON ORIENTATO



Consideriamo solo gli archi che hanno portato verso un nodo non visitato e "appendiamo" il grafo dal nodo di partenza

VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO



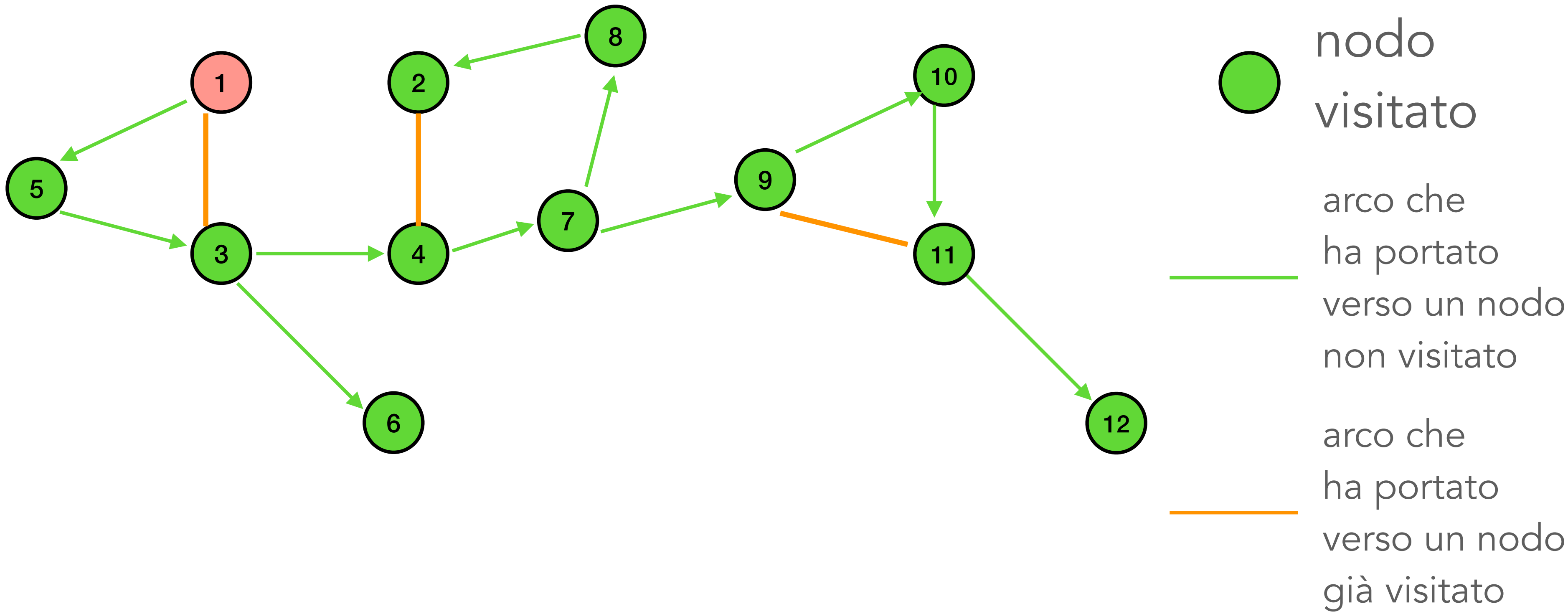
ALBERO DI COPERTURA:

- T = (V,E') sottografo di G = (V,E)
- T contiene TUTTI i nodi di G
- T è un albero

L'albero può essere salvato e utilizzato successivamente per visitare i nodi più velocemente

VISITA in PROFONDITA' (DFS)

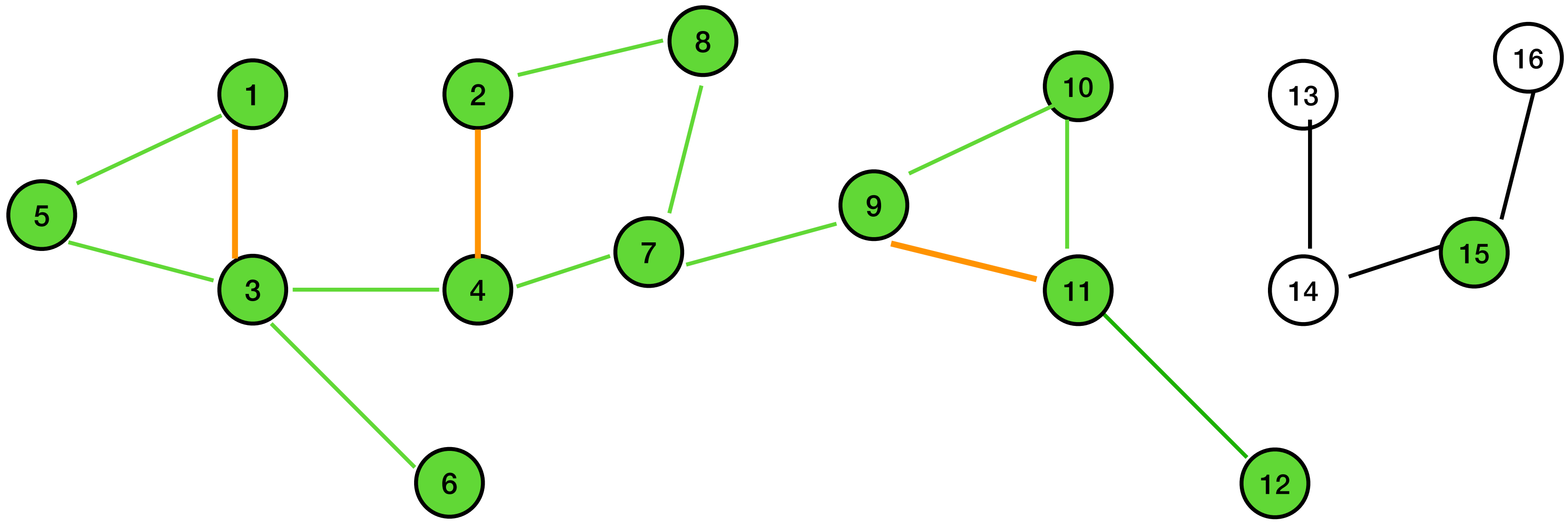
ESEMPIO - GRAFO NON ORIENTATO



Partendo con la visita da un altro nodo si ottiene un albero di copertura **DIVERSO**

VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO NON CONNESSO

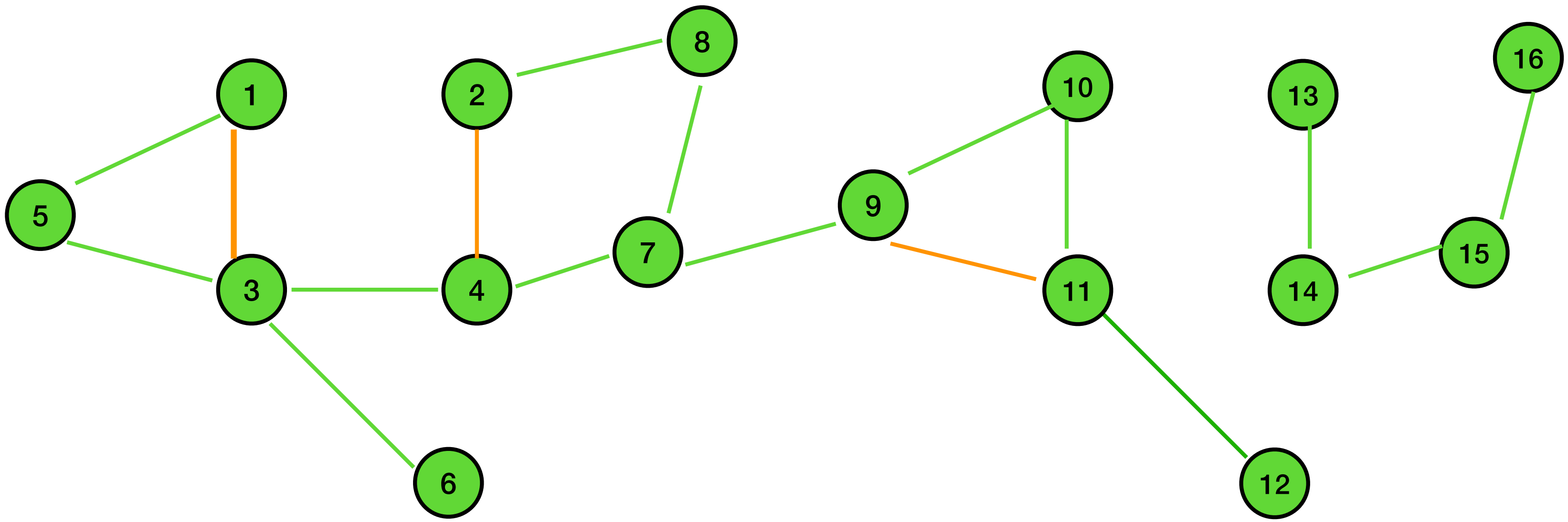


Ricominciamo da un nodo non ancora visitato dell'altra componente

UNIMORE

VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO NON CONNESSO

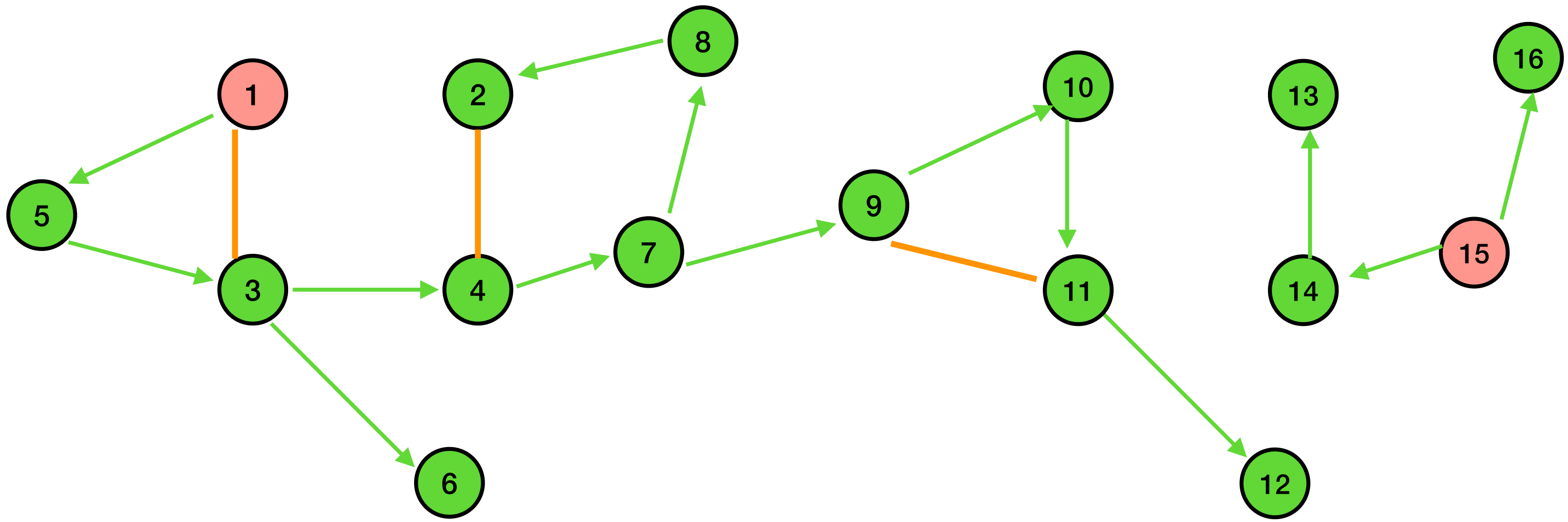


Ricominciamo da un nodo non ancora visitato dell'altra componente

UNIMORE

VISITA in PROFONDITA' (DFS)

ESEMPIO - GRAFO NON ORIENTATO NON CONNESSO



FORESTA DI COPERTURA:

- {T1 = (V1,E1), T=(V2,E2)… Tk=(Vk,Ek)} sottografi di G = (V,E)
- L'unione dei nodi dei Ti è uguale all'insieme di TUTTI i nodi di G
- Ti è un albero, per ogni i = 1,..k

UNIMORE

VISITA in PROFONDITA' (DFS)

PSEUDO-CODICE

IDEE:

- Usiamo la ricorsione

⇨ Dopo aver visitato un nodo, visitiamo ricorsivamente i suoi vicini non ancora visitati

- Dobbiamo avere un modo per sapere se un nodo è già stato visitato

⇨ Usiamo un array di booleani `visited[1..n]` in cui ricordare questa informazione

VISITA in PROFONDITA' (DFS)

PSEUDO-CODICE

Procedura principale

```
DFS(G)
 for all v ∈ V
   visited[v] := FALSE
 for all v ∈ V
   if visited[v] = FALSE
    then DFS-Visit(G,v)
```

Il nodo è già stato visitato?

`visited[v] = FALSE`
se non è ancora stato incontrato durante la visita

`visited[v] = TRUE`
se il nodo è già stato incontrato durante la visita

Procedura ricorsiva

```
DFS-Visit(G,v)
 visited[v] := TRUE
 esamina nodo v (caso pre-visita)
 for all (v,u) ∈ E
 // archi incidenti in v
   if visited[u] = FALSE
    then DFS-Visit(G,u)
 esamina nodo v (caso post-visita)
```

Costo computazionale?

DFS senza chiamate ricorsive —> $O(|V|)$

+

una chiamata ricorsiva per ogni nodo:

per nodo v paghiamo $O(deg(v))$, il grado del nodo

In totale $\sum_{v \in V} deg(v) = 2|E| \in O(|E|)$

VISITA in PROFONDITA' (DFS)

PSEUDO-CODICE

Procedura principale

```
DFS(G)
 for all v ∈ V
   visited[v] := FALSE
 for all v ∈ V
   if visited[v] = FALSE
    then DFS-Visit(G,v)
```

Procedura ricorsiva

```
DFS-Visit(G,v)
 visited[v] := TRUE
 esamina nodo v (caso pre-visita)
 for all (v,u) ∈ E
 // archi incidenti in v
  if visited[u] = FALSE
   then DFS-Visit(G,u)
 esamina nodo v (caso post-visita)
```

Il nodo è già stato visitato?

`visited[v] = FALSE`
se non è ancora stato incontrato durante la visita

`visited[v] = TRUE`
se il nodo è già stato incontrato durante la visita

Costo computazionale?

$$O(|V|+|E|)$$

Foresta di Copertura

PSEUDO-CODICE

Funzione principale

```
DFS(G)
  for all v ∈ V
   visited[v] := FALSE
   prev[v] := 0
  for all v ∈ V
   if visited[v] = FALSE
    then DFS-Visit(G,v)
  return prev[]
```

Procedura ricorsiva

```
DFS-Visit(G,v)
 visited[v] := TRUE
 for all (v,u) ∈ E
 // archi incidenti in v
  if visited[u] = FALSE
   then
    prev[u] := v
    DFS-Visit(G,u)
```

prev[u] memorizza il predecessore (padre) di **u** nell'albero di copertura a cui appartiene

Alla fine dell'esecuzione, tutti i nodi **v** per cui **prev[v] = 0** sono radici degli alberi che compongono la foresta di copertura

Foresta di Copertura

PSEUDO-CODICE

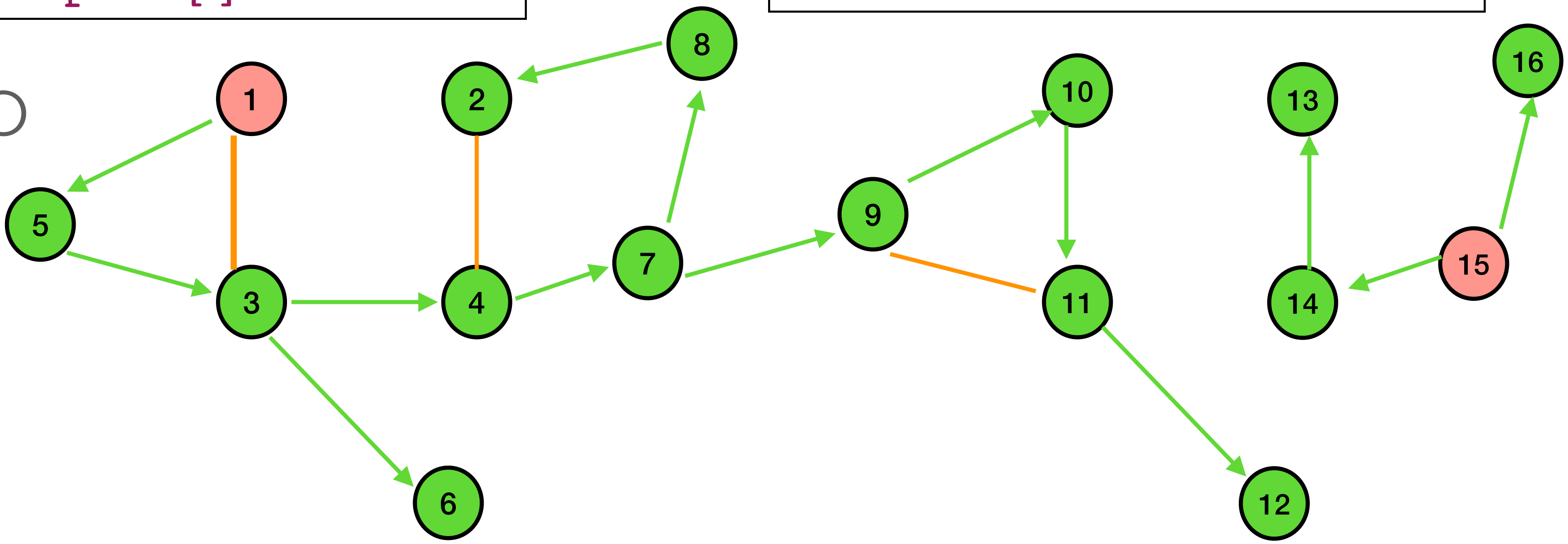
Funzione principale

```
DFS(G)
  for all v ∈ V
   visited[v] := FALSE
   prev[v] := 0
  for all v ∈ V
   if visited[v] = FALSE
    then DFS-Visit(G,v)
  return prev[]
```

Procedura ricorsiva

```
DFS-Visit(G,v)
  visited[v] := TRUE
  for all (v,u) ∈ E
  // archi incidenti in v
   if visited[u] = FALSE
    then
     prev[u] := v
     DFS-Visit(G,u)
```

ESEMPIO



prev	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	0	8	5	3	1	3	4	7	7	9	10	11	14	15	0	15

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

IDEA:

- Facciamo una DFS
- Se troviamo un back-edge, nel grafo c'è un ciclo
- Altrimenti no

arco che porta ad un antenato dell'albero di copertura della visita DFS

L'esplorazione di un arco ci porta in un nodo già visitato che non sia il padre nell'albero di copertura della visita DFS

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

L'esplorazione di un arco ci porta in un nodo già visitato

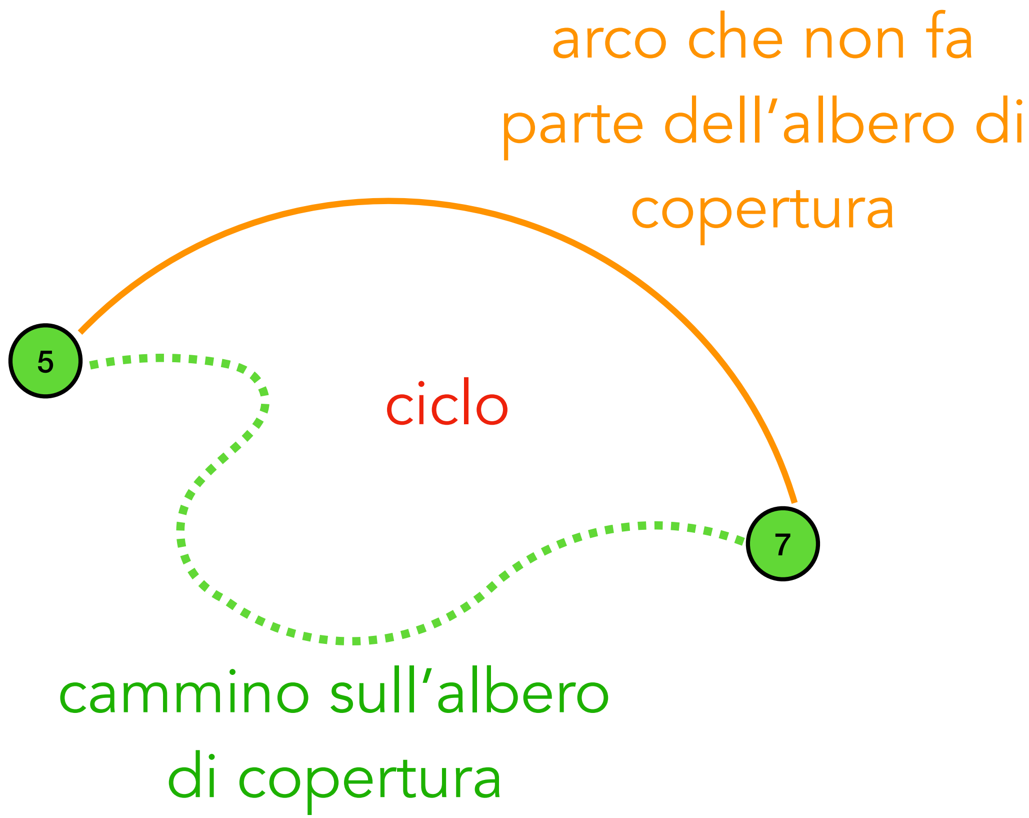
⇓

I nodi stanno nella stessa componente connessa

⇓

I nodi stanno nello stesso albero di copertura

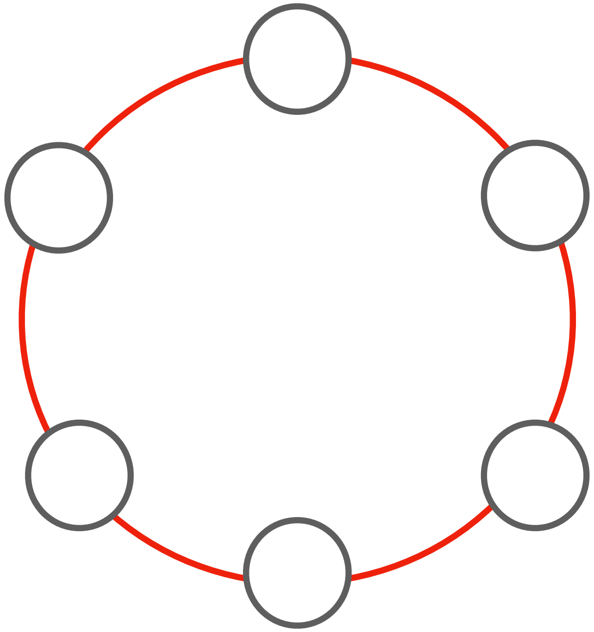
⇗



Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

se il grafo ha un ciclo, l'esplorazione ci porta in un nodo già visitato

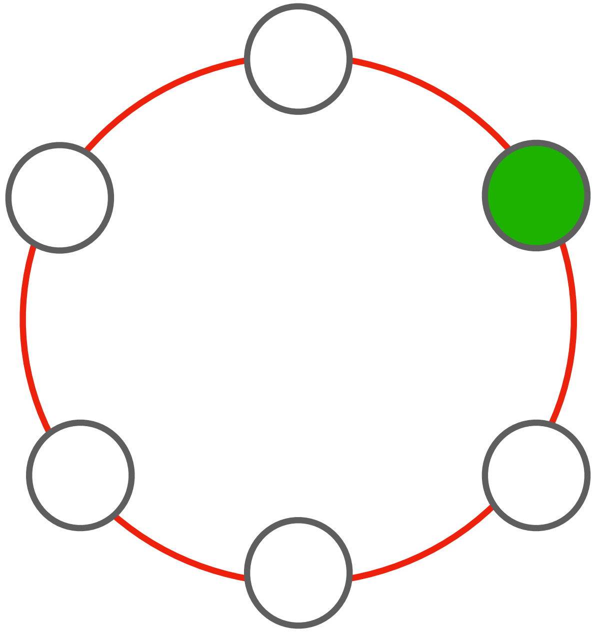


Il grafo ha un ciclo

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

se il grafo ha un ciclo, l'esplorazione ci porta in un nodo già visitato



Il grafo ha un ciclo
(supponiamo uno solo per semplicità)

Primo nodo raggiunto dalla DFS

⇓

la DFS continua in profondità

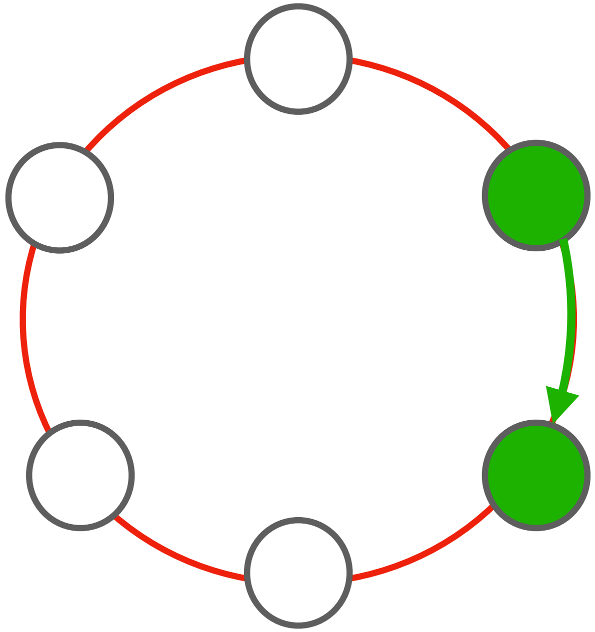
⇓

prima o poi verrà esplorato l'arco verso il secondo nodo del ciclo

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

se il grafo ha un ciclo, l'esplorazione ci porta in un nodo già visitato



Il grafo ha un ciclo
(supponiamo uno solo per semplicità)

Primo nodo raggiunto dalla DFS

⇓

la DFS continua in profondità

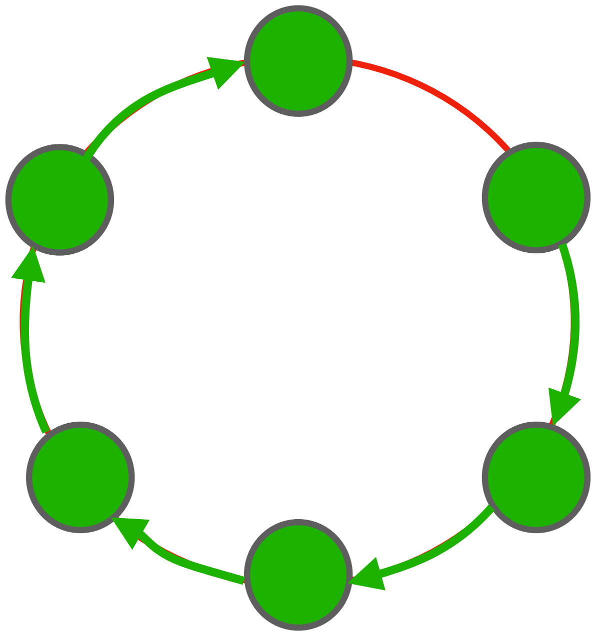
⇓

prima o poi verrà esplorato l'arco verso il secondo nodo del ciclo

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

se il grafo ha un ciclo, l'esplorazione ci porta in un nodo già visitato



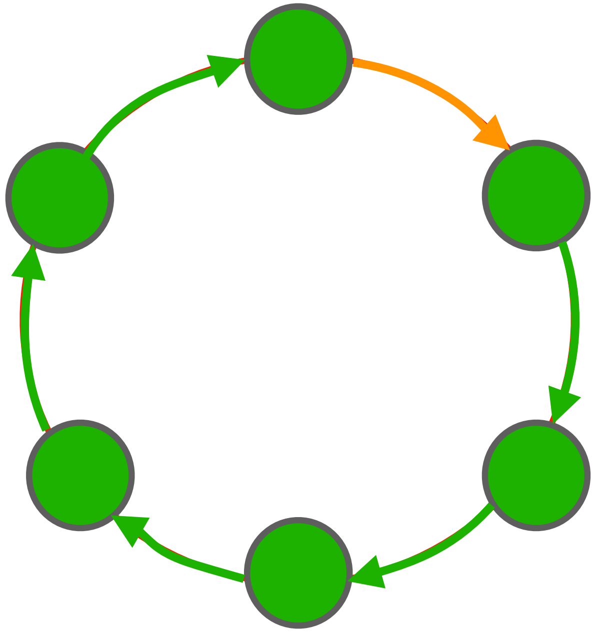
Il grafo ha un ciclo
(supponiamo uno solo per semplicità)

prima o poi verranno esplorati tutti gli archi che portano agli altri nodi del ciclo

Ricerca ciclo in un grafo non orientato

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

se il grafo ha un ciclo, l'esplorazione ci porta in un nodo già visitato



Il grafo ha un ciclo
(supponiamo uno solo per semplicità)

prima o poi verranno esplorati tutti gli archi che portano agli altri nodi del ciclo

⇓

prima o poi verrà esplorato l'ultimo arco del ciclo portando al primo nodo visitato

Ricerca ciclo in un grafo

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

ESERCIZIO

modificare la DFS per risolvere il problema del test grafo aciclico

Osservazione 1: ci deve essere una funzione principale che prende in input il grafo G e restituisce un valore (T/F), e una funzione ricorsiva che restituisce True se è stato trovato un ciclo durante la DFS da un nodo v, False altrimenti
Osservazione 2: ci deve essere un modo per distinguere un antenato già visitato dal padre

PROVATE a farlo DA SOLI prima di guardare il lucido successivo

Ricerca ciclo in un grafo

TEST GRAFO NON ORIENTATO con CICLO:
INPUT: grafo NON ORIENTATO G=(V,E)
OUTPUT: TRUE se il grafo G contiene un ciclo, FALSE altrimenti

Funzione principale

```
DFS(G)
  for all v ∈ V
   visited[v] := FALSE
   prev[v]:=0
  for all v ∈ V
   if visited[v] = FALSE AND DFS-Visit(G,v)
    then return TRUE
  return FALSE
```

Restituisce VERO se a partire da **v** si determina un ciclo, FALSO altrimenti

Funzione ricorsiva

```
DFS-Visit(G,v)
  visited[v] := TRUE
  for all (v,u) ∈ E
  // archi incidenti in v
   if visited[u] = FALSE
    then prev[u]:= v
   if prev[v] ≠ u AND (visited[u] = TRUE OR DFS-Visit(G,u))
          then return TRUE
  return FALSE
```

Se il nodo u non è il padre di v (ovvero non è il nodo attraverso cui è stato scoperto v) ma è già stato visitato⇒ l'arco (v,u) è un back-edge

Se il nodo u non è il padre di v (ovvero non è il nodo attraverso cui è stato scoperto v), u non è già stato visitato, ma la visita partendo da u trova un back-edge, allora c'è un ciclo nel grafo

ESERCIZI - componenti connesse

Nei lucidi che seguono trovate degli
esercizi relativi
all'utilizzo della DFS per risolvere problemi su
grafi indiretti.

In un lucido trovate il problema,
nel successivo un suggerimento
(che non è altro che il ragionamento che abbiamo fatto a lezione).
Alla fine trovate le soluzioni.

Cercate di ragionare sul problema PRIMA di
guardare il suggerimento
e
di scrivere lo pseudo codice PRIMA di
guardare la soluzione.

In ogni caso si tratta di riadattare il codice della DFS che
abbiamo visto a lezione

Componenti connesse - grafo indiretto

PROBLEMA 1:
INPUT: Grafo G=(V,E) indiretto
OUTPUT: Il grafo è connesso?

Esercizio
scrivere lo pseudocodice di
un algortimo che risolva il problema
sfruttando la DFS

Componenti connesse - grafo indiretto

PROBLEMA 1:

INPUT: Grafo G=(V,E) indiretto

OUTPUT: Il grafo è connesso?

SUGGERIMENTO:

Il grafo è connesso se una vista DFS che parte da un solo nodo arriva a visitare tutti i nodi.

Componenti connesse - grafo indiretto

PROBLEMA 2:

INPUT: Grafo G=(V,E) indiretto

OUTPUT: Il numero di componenti connesse

Esercizio

scrivere lo pseudocodice di
un algortimo che risolva il problema
sfruttando la DFS

Componenti connesse - grafo indiretto

PROBLEMA 2:

INPUT: Grafo G=(V,E) indiretto

OUTPUT: Il numero di componenti connesse

SUGGERIMENTO:

Tutti i nodi raggiungibili da una vista DFS che parte da un nodo fanno parte della stessa componente connessa.
Se ci sono nodi non ancora visitati, si inizia una nuova visita DFS da uno di questi nodi e si determina un'altra componente connessa.
E cosi' via fino a quando tutti i nodi sono stati visitati.

Usare un contatore che viene incrementato ogni volta che si inizia una nuova DFS da un nuovo nodo che non è stato raggiunto dalle visite precedenti.

Componenti connesse - grafo indiretto

PROBLEMA 3:

INPUT: Grafo `G=(V,E)` indiretto

OUTPUT: Determinare le componenti connesse di `G`

Associare ad ogni nodo di G l'identificativo della componente connessa a cui appartiene.

Gli identificativi possono essere interi.

Nodi che si trovano nella stessa componente connessa DEVONO essere associati allo stesso identificativo di componente connessa.

Esercizio

scrivere lo pseudocodice di
un algoritmo che risolva il problema
sfruttando la DFS

Componenti connesse - grafo indiretto

PROBLEMA 3:

INPUT: Grafo `G=(V,E)` indiretto

OUTPUT: Determinare le componenti connesse di `G`

SUGGERIMENTO:

Tutti i nodi raggiungibili da una vista DFS che parte da un nodo fanno parte della stessa componente connessa.
Se ci sono nodi non ancora visitati, si inizia una nuova visita DFS da uno di questi nodi e si determina un'altra componente connessa.
E cosi' via fino a quando tutti i nodi sono stati visitati.

Usiamo un array `cc[1..n]` per contenere gli identificatori delle componenti connesse:

- `cc[v] = id` —> il nodo `v` appartiene alla componente connessa con identificativo `id`
- `cc[v] = 0` —> il nodo non è ancora stato analizzato

SOLUZIONI ESERCIZI
componenti connesse

Guardare solo dopo aver provato a scrivere lo pseudocodice da soli!

Componenti connesse - grafo indiretto

PROBLEMA 1:

INPUT: Grafo G=(V,E) indiretto

OUTPUT: Il grafo è connesso?

```
GC(G)
 for all v ∈ V
  visited[v] := FALSE
 DFS-Visit(G,1)
 for all v ∈ V
  if visited[v] = FALSE
   then return FALSE
 return TRUE
```

```
DFS-Visit(G,v)
 visited[v] := TRUE
 for all (v,u) ∈ E
 // archi incidenti in v
  if visited[u] = FALSE
   then DFS-Visit(G,u)
```

Costo computazionale

$$O(|V|+|E|)$$

Componenti connesse - grafo indiretto

PROBLEMA 2:

INPUT: Grafo G=(V,E) indiretto

OUTPUT: Il numero di componenti connesse

```
#CC(G)
  for all v ∈ V
   visited[v] := FALSE
  c := 0
  for all v ∈ V
   if visited[v] = FALSE
    then
     c := c +1
     CC-Visit(G,v)
  return c
```

```
DFS-Visit(G,v)
  visited[v] := TRUE
  for all (v,u) ∈ E
  // archi incidenti in v
   if visited[u] = FALSE
    then DFS-Visit(G,u)
```

c viene incrementato ogni volta che si inizia l'esplorazione di una nuova componente connessa

Costo computazionale $O(|V|+|E|)$

Componenti connesse - grafo indiretto

PROBLEMA 3:

INPUT: Grafo `G=(V,E)` indiretto

OUTPUT: Determinare le componenti connesse di `G`

```
CC(G)
  for all v ∈ V
    cc[v] := 0
  id := 0
  for all v ∈ V
    if cc[v]=0
      then
        id := id +1
        CC-Visit(G,v,id)
  return cc
```

```
CC-Visit(G,v,id)
  cc[v] := id
  for all (v,u) ∈ E
  // archi incidenti in v
    if cc[u] = 0
      then DFS-Visit(G,u,id)
```

Tutti i nodi raggiungibili da **v** apparterranno alla componente connessa con identificativo **id**

id viene incrementato ogni volta che si inizia una nuova componente connessa

Costo computazionale $O(|V|+|E|)$