

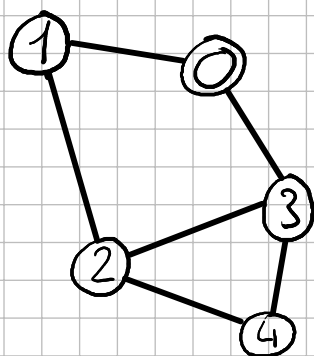
Algoritmo di visita

Un algoritmo di visita per un grafo deve

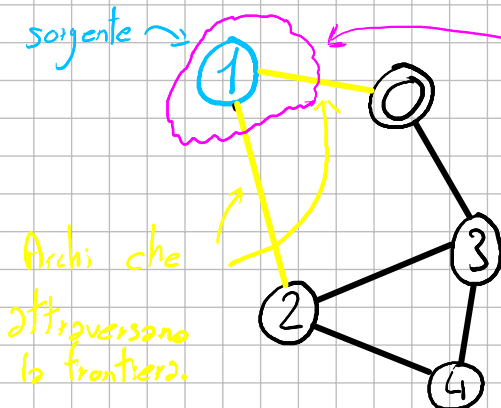
- 1) Partire da una sorgente
- 2) Tenere traccia dei nodi scoperti:
- 3) Attraversare Archi che hanno un estremo scoperto ed un estremo non ancora scoperto

Esempio:

(visits non condizionati)
visits tutto il grafo



Passo 1°) Entro nella sorgente, ovvero il primo nodo che visitiamo.



Frontiers
to S-V-S

Frontiers: Nodes visited

Pseudo Codice:

```
visitaGenerica(sorg){
    S = {sorg};
    finché è possibile
        scegli(u, v) tali che  $u \in S, v \notin S$ 
         $S = S \cup v$  // aggiungo ad S il nodo appena visitato.
}
// dipende dall'algoritmo
```

Proprietà dell'algoritmo:

Cosa deve fare \rightarrow Come lo fa

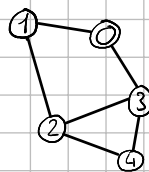
1) deve terminare \rightarrow ad ogni iterazione aggiungi un nodo a quelli scoperti, termina in $n-1$ iterazioni

2) deve scoprire tutti i nodi raggiungibili \rightarrow mediante ad un cammino su grafo; ovvero una sequenza ordinata di nodi v_1, v_2, \dots, v_k dove

$$(v_i, v_{i+1}) \in E, \text{ con } v_{i+1} \text{ vicino di } v_i$$

significa che questo arco deve appartenere al grafo dell'arco

Es.



1, 2, 3 è un cammino ✓

1, 2, 4 non è un cammino ✗

1, 2, 1, 2 è un cammino ✓

Cammino Semplice: i nodi non si ripetono

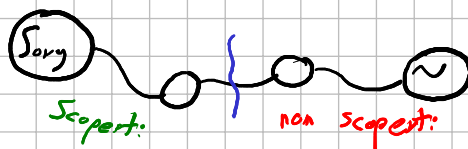
1, 2, 3 è semplice ✓

1, 2, 1, 2 non è semplice ✗

3) u raggiungibile da v se esiste in G un cammino. $V = v_1, v_2, \dots, v_k = u$

\rightarrow In questo caso dobbiamo usare una dimostrazione per assurdo, ovvero negare la tesi, nel nostro caso che u non viene scoperto

È possibile che u non venga mai scoperto?



```
visitaGenerica(sorg){  
  S = {sorg};  
  finché è possibile  
    scegli  $(u, v)$  tali che  $u \in S, v \notin S$   
     $S = S \cup v$   
}
```

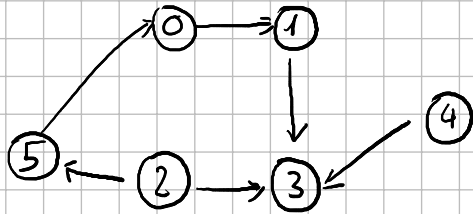
No, non è possibile che u non venga mai scoperto, perché questo pezzo di codice si occupa di questo.

Abbiamo dimostrato l'ipotesi.

Per i grafi orientati?

L'algoritmo è ancora valido, ma non possiamo più dire che scopre sempre tutti i nodi (punto 2), perché questo dipende da come sono orientati gli archi.

Es:



Per esempio non puoi andare da 3 a 4.

Basta dire che nel punto 2 invece che usare un cammino usiamo un cammino orientato, ovvero un cammino che segue il senso della freccia.

$$v_1, v_2, \dots, v_k, (v_i, v_{i+1}) \in E$$

Modifica dell'algoritmo di visita

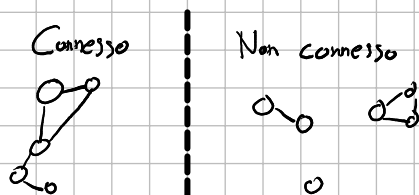
```
visitaGenerica(sorg){  
  S = {sorg};  
  • A = {} // archi attraversati:  
  finché è possibile  
    scegli(u, v) tali che  $u \in S, v \notin S$   
    S = S  $\cup$  v  
    • A = A  $\cup$  {(u, v)}  
}
```

// grafo di partenza non deve essere ordinato

Con queste due aggiunte (•) siamo in grado di generare il grafo $T(s, A)$, un grafo connesso e ciclico.

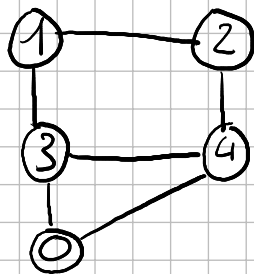
• **Connesso**: se $\forall u, v \in V$ u è raggiungibile da v e viceversa.

Es:



• **Ciclico**: se esiste un cammino v_1, v_2, \dots, v_k con $v_1 = v_k$, viene anche chiamato cammino chiuso.

Es:



$1, 2, 4, 3, 1$ é un ciclo ✓

$1, 2, 4, 3$ non é un ciclo ✗

$1, 2, 4, 1$ non é un ciclo perché non esiste quel cammino ✗

Inoltre esistono:

Cicli semplici: i nodi non si ripetono

$1, 2, 4, 3, 1$ é un ciclo semplice ✓

Occhio che ciclo semplice \neq cammino semplice

Cicli non semplici: i nodi si ripetono

$1, 2, 4, 3, 0, 4, 3, 1$ non é un ciclo semplice

Cicli banali: sono cicli che ad un certo punto fanno lo stesso percorso al contrario.

Sono cicli sempre presenti, perché derivano dal fatto che il grafo é orientato

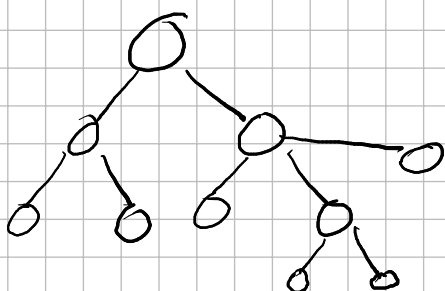
$1, 2, 1$ é un ciclo banale

$1, 2, 4, 3, 0, 4, 2, 1$ non é semplice e non é banale

Grafo aciclico: Grafo senza Cicli semplici

Adesso che abbiamo capito questi concetti possiamo andare avanti con il nostro algoritmo.

Possiamo dire che T é un albero di visita.



essendo non orientato non ha una vera radice, dato che ogni nodo può essere una radice.

Come noti non c'è la possibilità di fare

Cicli non banali

Dimostrazione di $T(S, A)$ è un albero di ricerca:

Dimostrazione per induzione:

Induzione su $|S|$ ovvero i nodi scoperti

Caso Base:

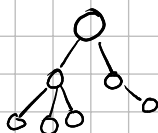
$$S = \{\text{sorg}\} \quad |S| = 1$$



Passo Induttivo:

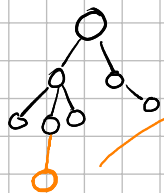
Se è vero per $|S|=k$ è vero anche per $|S|=k+1$

Significa che quando sono stati visitati k nodi allora la struttura costruita è effettivamente un albero



Ora facciamo finta di fare un ulteriore passo di visita, ovvero

```
visitaGenerica(sorg){  
  S = {sorg};  
  A =  $\emptyset$   
  finché è possibile  
    scegli  $(u, v)$  tali che  $u \in S, v \notin S$   
    S =  $S \cup v$   
    A =  $A \cup \{(u, v)\}$   
}
```



Il nuovo nodo viene aggiunto alla struttura già esistente, quindi la struttura rimane connessa.

In più non si creano cicli perché, per aggiungere un ciclo dovresti aggiungere un arco tra

nodi che sono già stati scoperti, ma questo

non può mai accadere perché l'arco viene

collegato con un nodo che non era ancora

scoperto. $v \notin S$

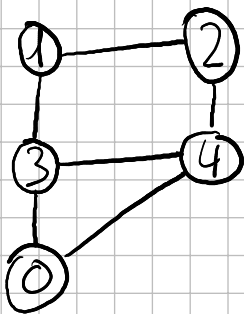
In questo modo abbiamo dimostrato l'ipotesi induttiva.

Visita BFS (Ricerca in ampiezza):

- la frontiera viene gestita con una coda.

```
visitaGenerica(sorg){  
  S = {sorg};  
  A =  $\emptyset$   
  coda  $\leftarrow$  sorg  
  finché coda non vuota  
    u  $\leftarrow$  coda // dequeue / rimozione da coda  
    per ogni vicino u di v  
      se  $v \notin S$  // se non è stato scoperto  
        S = S  $\cup$  v  
        coda  $\leftarrow$  u  
        A = A  $\cup$  {(u,v)}  
}
```

Esempio Funzionamento



prendiamo come sorgente il nodo 1

Coda

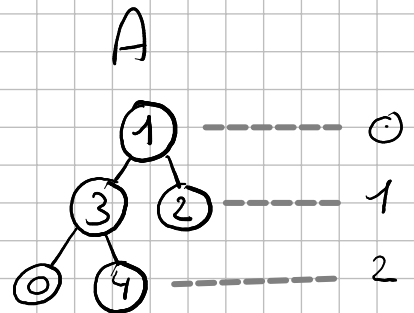
~~1~~ ~~3~~ ~~2~~ ~~0~~ ~~4~~

S

1, 3, 2, 0, 4

A

(1,3) (1,2) (3,0) (3,4)



A ti dice la distanza dei nodi dalla sorgente

Distanza: lunghezza

minima per arrivare ad un nodo.