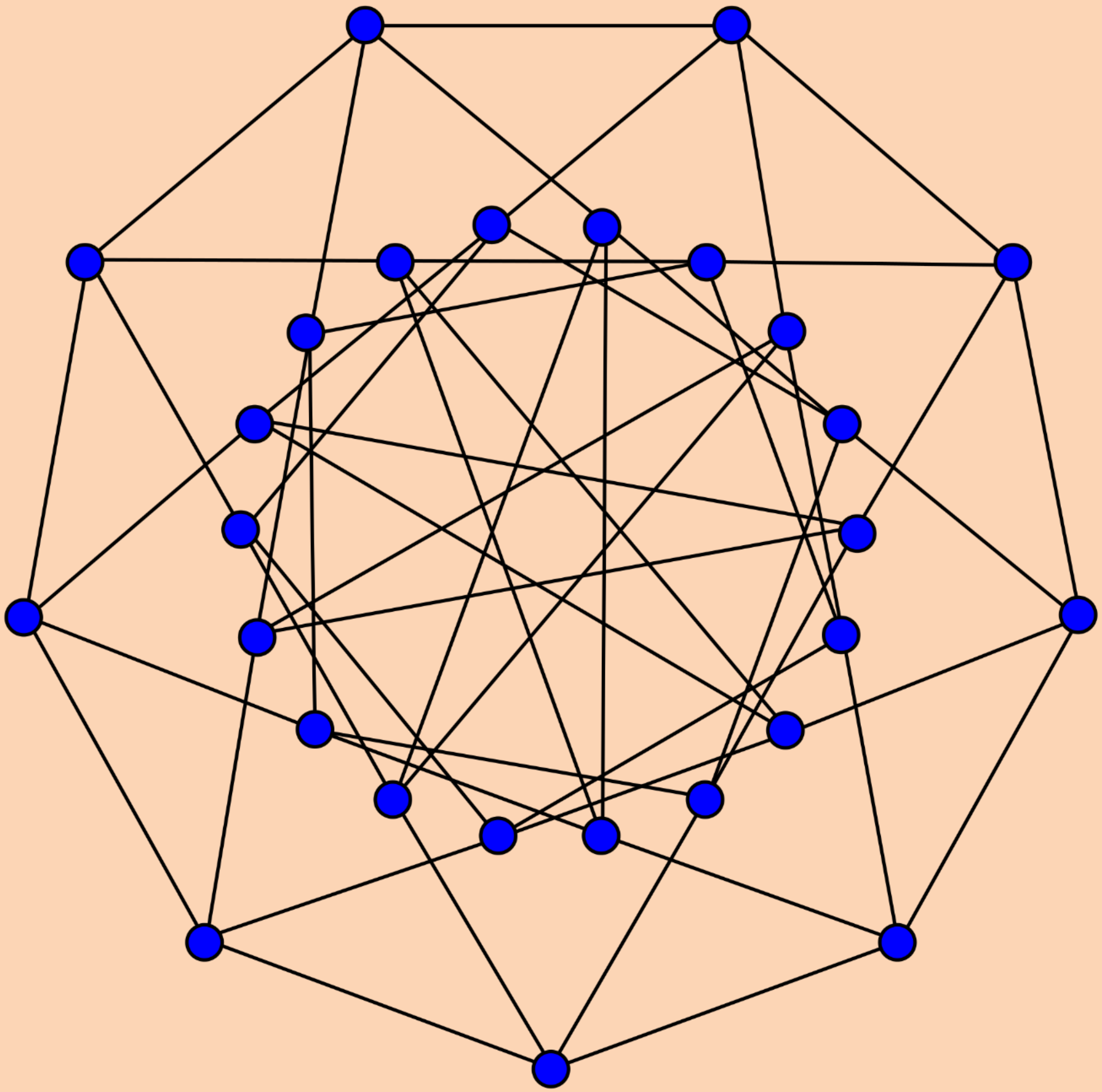
BFS e DFS: implementazioni e applicazioni

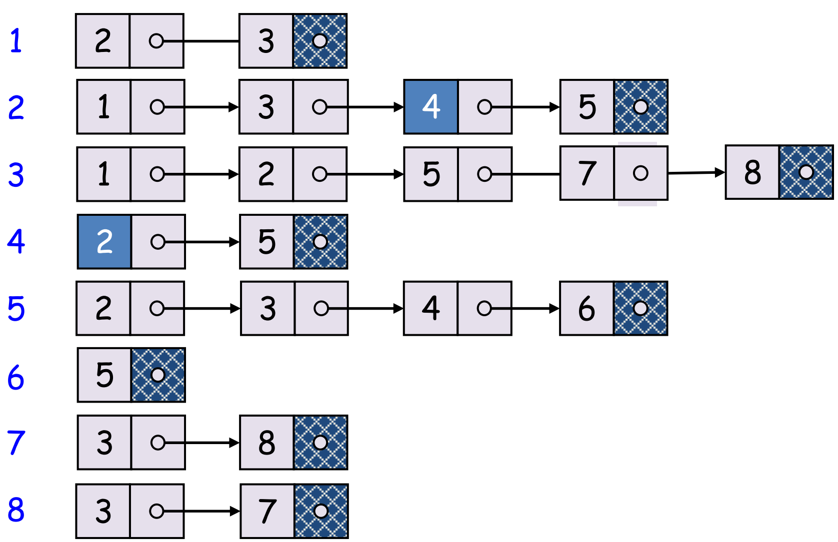
3 Maggio 2023



Esplorare un grafo da una sua rappresentazione

What parts of the graph are reachable from a given vertex?

To understand this task, try putting yourself in the position of a computer that has just been given a new graph, say in the form of an adjacency list. This representation offers just one basic operation: finding the neighbors of a vertex. With only this primitive, the reachability problem is rather like exploring a labyrinth (Figure 3.2). You start walking from a fixed place

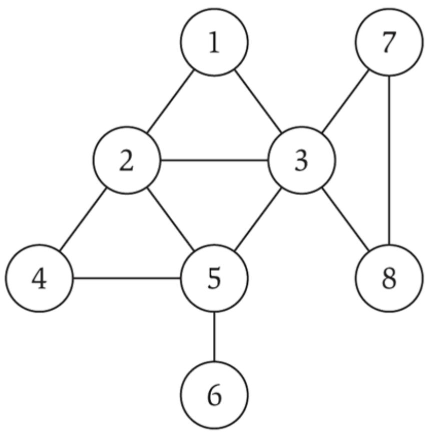


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

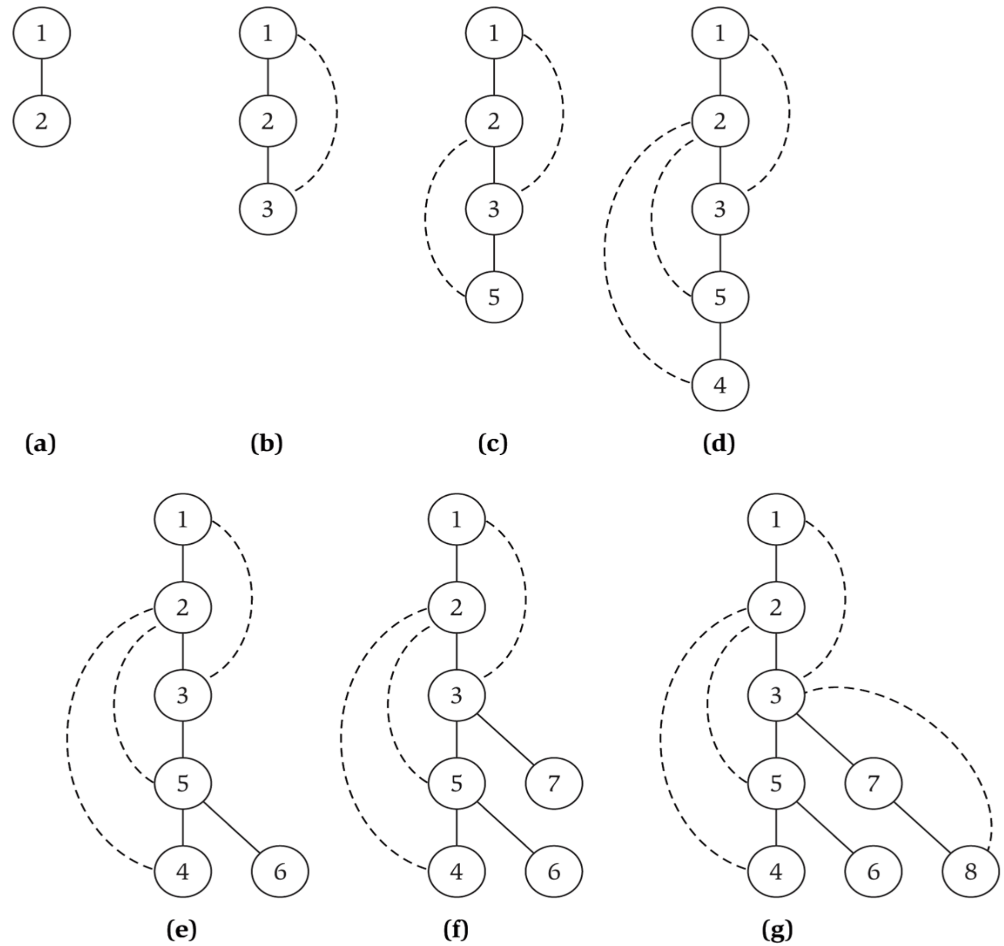
da [DPV]

DFS: visita in profondità

Idea di DFS: Esplorare quanto più in profondità possibile e tornare indietro ("backtrack") solo quando è necessario (*come in un labirinto…)*



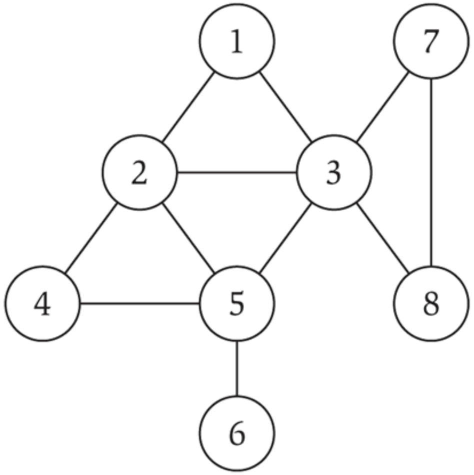
G

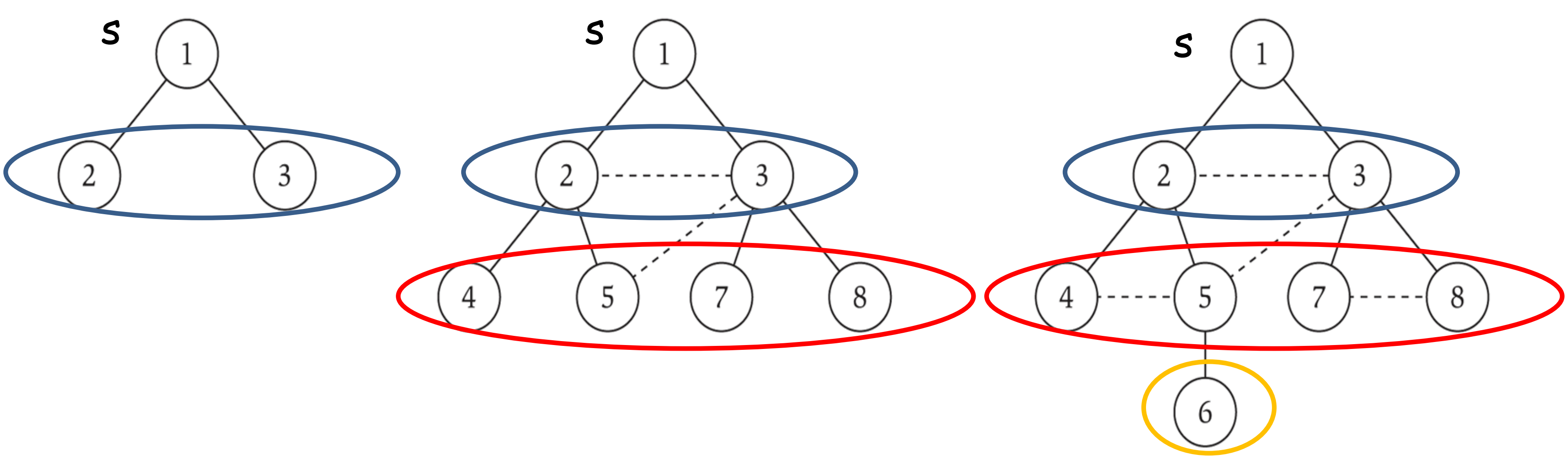


Gli archi non tratteggiati in (g) formano l'albero DFS di G

BFS: visita in ampiezza

Idea della BFS: Esplorare a partire da s in tutte le possibili direzioni, aggiungendo nodi, uno strato ("layer") alla volta.

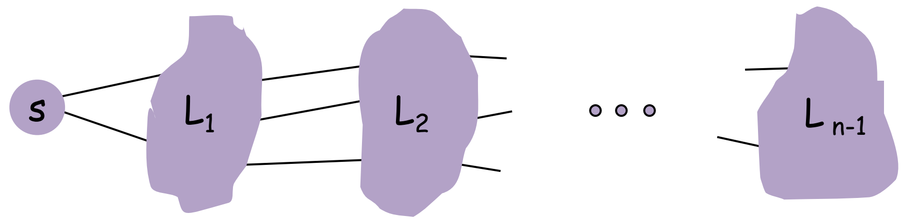




(a) (b) (c)

Breadth First Search

L_i sono i layers:



Algoritmo BFS:

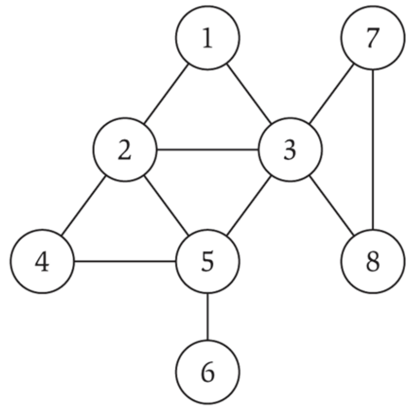
- $L_0 = \{ s \}$.
- L_1 = tutti i vicini di L_0.
- L_2 = tutti i nodi che non sono in L_0 o L_1, e che hanno un arco con un nodo in L_1.
- …
- L_{i+1} = tutti i nodi che non sono in un layer precedente, e che hanno un arco con un nodo in L_i.

Teorema.
Per ogni i, L_i consiste di tutti i nodi a distanza i da s.
Esiste un cammino da s a t **se e solo se** t appare in qualche layer.

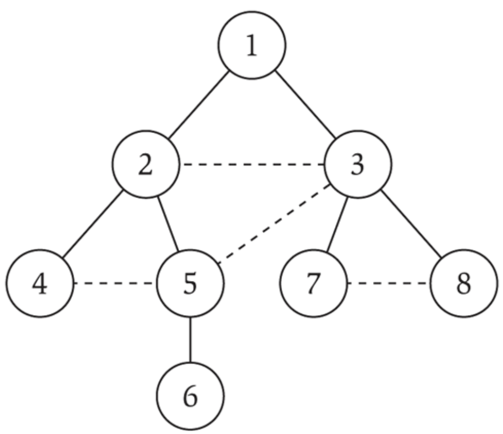
Prova: per induzione su i

Alberi BFS e DFS

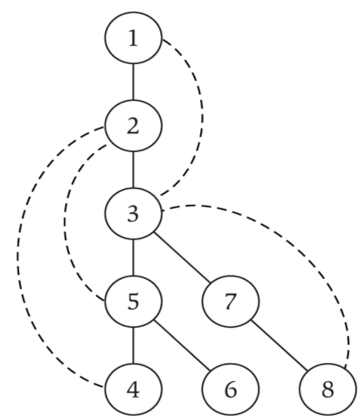
Grafo G



Albero BFS di G



Albero DFS di G



(g)

Implementazioni di BFS e DFS (par. 3.3)

Vedremo che BFS e DFS possono essere visti come lo stesso algoritmo con l'unica differenza che BFS mantiene i vertici da analizzare in una coda (*queue*) DFS in una pila (*stack*).

In entrambi gli algoritmi vi è una distinzione fra
l'azione di scoprire un nodo (*discover*, la prima volta che vi arriviamo) e
quella di esplorare un nodo (*explore*, quando tutti gli archi uscenti sono stati esaminati).

BFS e DFS differiscono nell'ordine in cui queste azioni sono eseguite.

BFS usa Discovered[v]; DFS usa Explored[v].

BFS si può implementare con una coda (queue FIFO); DFS con uno stack (LIFO).

BFS implementazione

BFS(s):

Set Discovered[s] = true and Discovered[v] = false for all other v

Initialize $L[0]$ to consist of the single element s

Set the layer counter $i = 0$

Set the current BFS tree $T = \emptyset$

While $L[i]$ is not empty

  Initialize an empty list $L[i+1]$

  For each node $u \in L[i]$

    Consider each edge (u, v) incident to u

    If Discovered[v] = false then

      Set Discovered[v] = true

      Add edge (u, v) to the tree T

      Add v to the list $L[i+1]$

    Endif

  Endfor

  Increment the layer counter i by one

Endwhile

BFS: analisi

Teorema: L'implementazione di BFS richiede tempo $O(m+n)$ se il grafo è rappresentato con liste di adiacenza.

Prova:
E' facile provare un running time $O(n^2)$.
Un'analisi più accurata da $O(m+n)$. (segue)

Nota: tempo $O(m+n)$ significa lineare nella taglia del grafo.

BFS analisi

```
BFS(s):
  Set Discovered[s] = true and Discovered[v] = false for all other v
  Initialize L[0] to consist of the single element s
  Set the layer counter i = 0
  Set the current BFS tree T = Ø
  While L[i] is not empty
    Initialize an empty list L[i+1]
    For each node u ∈ L[i]
      Consider each edge (u,v) incident to u
      If Discovered[v] = false then
        Set Discovered[v] = true
        Add edge (u,v) to the tree T
        Add v to the list L[i+1]
      Endif
    Endfor
    Increment the layer counter i by one
  Endwhile
```

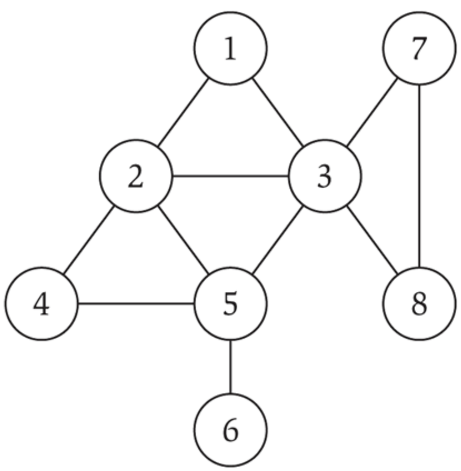
Il tempo di esecuzione è O(m+n):

- Inizializzazione in O(n)
- Al massimo n+1 liste L[i] da creare in O(n)
- Ogni nodo è presente in al più una lista: per un fissato nodo u vi sono deg(u) archi incidenti (u,v)

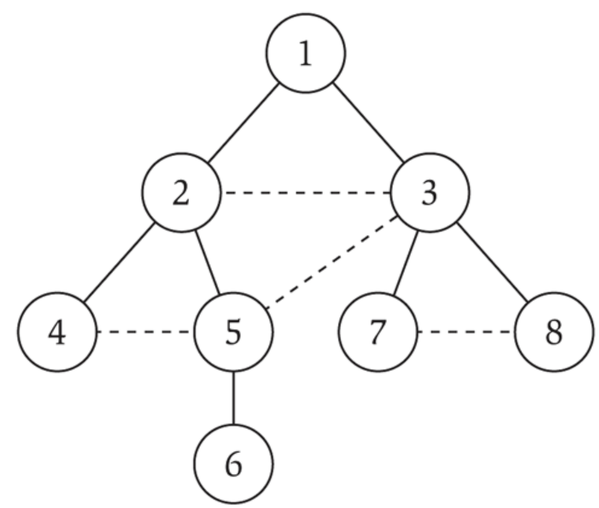
- Tempo totale per processare gli archi è $\sum_{u \in V} \deg(u) = 2m$

Implementazione di BFS con una coda (queue)

- L'implementazione vista usa varie liste L[i], una per ogni layer.
- Si può implementare BFS con una singola lista gestita come una coda.
- L'algoritmo inserisce un nodo alla fine della coda appena è scoperto la prima volta (discovered), mentre li esamina dal fronte della coda (da quello scoperto per primo). Quando estrae un nodo, inserisce i suoi nodi adiacenti non ancora esplorati.
- Si possono ottenere le stesse informazioni (layers e albero)



(1)
(2, 3)
(3, 4, 5)
(4, 5, 7, 8)
(5, 7, 8)
(7, 8, 6)
(8, 6)
(6)
()



Algoritmo DFS

Idea di DFS: Esplorare quanto più in profondità possibile e tornare indietro ("backtrack") solo quando è necessario (*come in un labirinto…)*

Algoritmo ricorsivo

R, insieme dei nodi esplorati

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u,v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

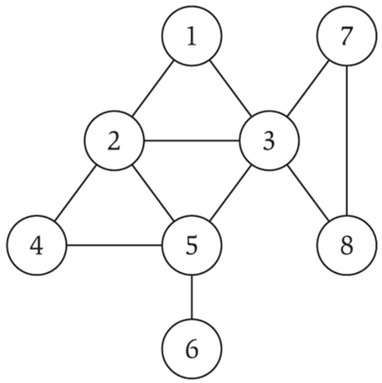
DFS implementazione (iterativa con stack)

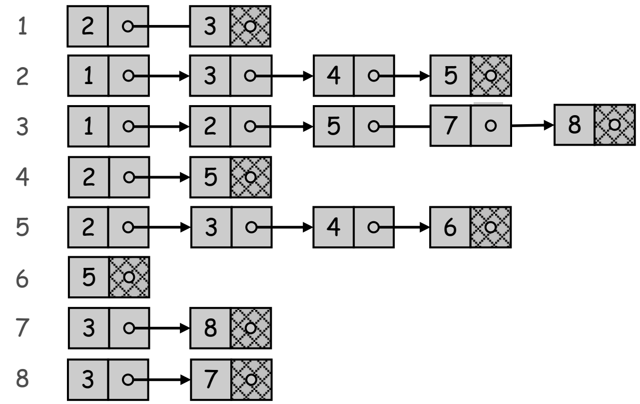
- L'algoritmo visto è ricorsivo.
- Si può implementare DFS in maniera analoga alla BFS, ma con una singola lista gestita come uno stack.
- L'algoritmo marca un nodo esplorato, quando lo toglie dallo stack per inserirvi i suoi nodi adiacenti.
- Si possono ottenere le stesse informazioni della versione ricorsiva.

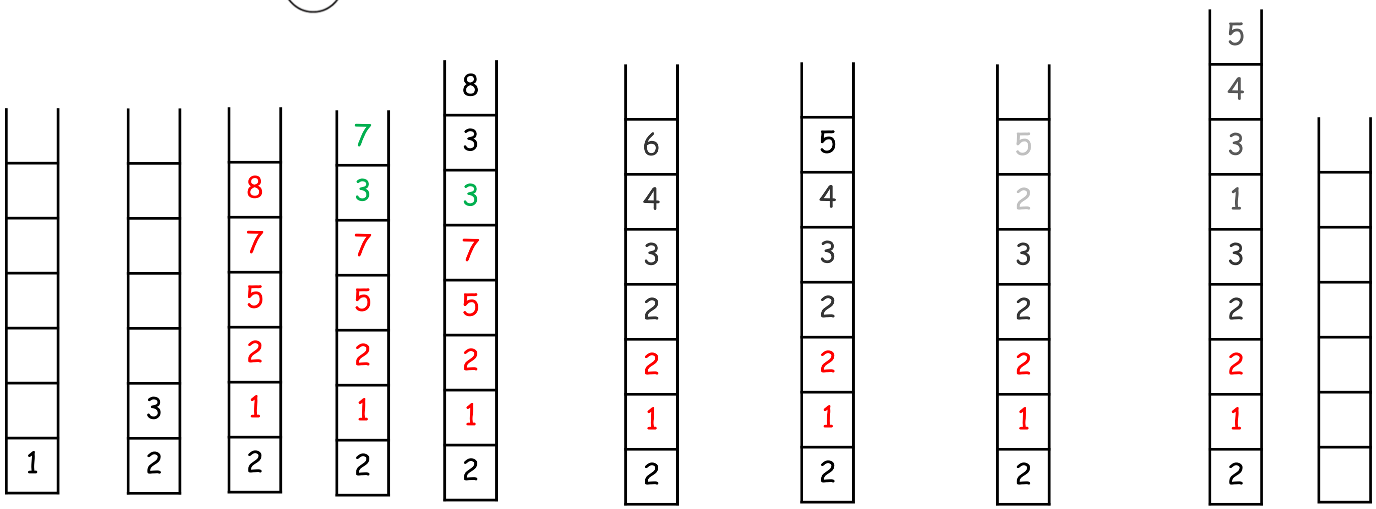
```
DFS(s):
  Initialize S to be a stack with one element s
  While S is not empty
    Take a node u from S
    If Explored[u] = false then
      Set Explored[u] = true
      For each edge (u, v) incident to u
        Add v to the stack S
      Endfor
    Endif
  Endwhile
```

Esempio DFS con stack

Grafo G







Explored: 1, 3, 8, 7, 5, 6, 4, 2

Nota: i nodi adiacenti ad u sono esaminati nell'ordine inverso in cui appaiono nella lista di adiacenza (differentemente dalla versione ricorsiva)

DFS: analisi

Teorema: L'implementazione di DFS richiede tempo $O(m+n)$ se il grafo è rappresentato con una lista delle adiacenze.

Prova:
Le operazioni elementari sono *push* e *pop* in $O(1)$.
Quante sono al più?
Contiamo il numero di push (il numero di pop sarà uguale).
Ogni elemento è inserito nello stack ogni volta che un suo adiacente è esplorato, cioè $deg(v)$.
In totale

$$\sum_{v \in V} deg(v) = 2m$$

Osserva

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)
Poni `Scoperto`[s] =`true` e `Scoperto`[v] =`false` $\forall v \neq s$
Inserisci s nella coda Q;
While Q non è vuota
Estrai il nodo u dalla testa della lista Q
  $\forall (u, v)$ incidente su u
  If Scoperto[v]=false **then**
    Poni Scoperto[v]=true
   Aggiungi v alla fine della coda Q

Applicazioni di BFS e DFS

- Problema della connettività s-t:

  Dati due nodi s e t, esiste un cammino fra s e t?

- Problema del cammino minimo s-t:

  Dati due nodi s e t, qual è la lunghezza del cammino minimo fra s e t (ovvero la distanza di s da t)?

- Problema della componente connessa di s: trovare tutti i nodi raggiungibili da s

- Problema di tutte le componenti connesse di un grafo G: trovare tutte le componenti connesse di G

Altre applicazioni di BFS e DFS

- Problema della verifica se un grafo è bipartito (par. 3.4)

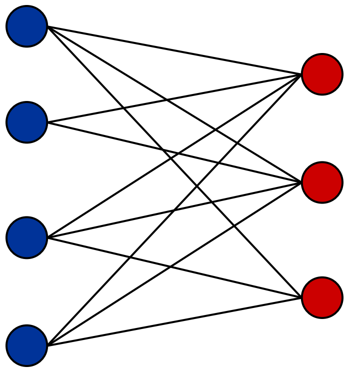
- Problema della connessione nei grafi diretti (par. 3.5)

… e altre ancora.

3.4 Testing Bipartiteness

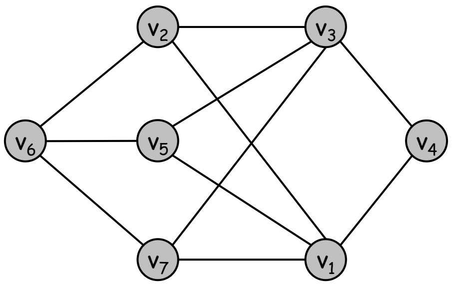
Def. An undirected graph $G = (V, E)$ is **bipartite** if the nodes can be colored red or blue such that every edge has one red and one blue end.

Applications.
- Stable marriage: men = red, women = blue.
- Scheduling: machines = red, jobs = blue.

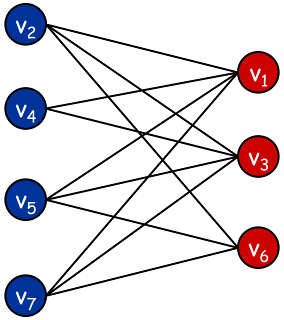


a bipartite graph

Testing bipartiteness. Given a graph G, is it bipartite?
- Many graph problems become:
  - easier if the underlying graph is bipartite (matching)
  - tractable if the underlying graph is bipartite (independent set)
- Before attempting to design an algorithm, we need to understand structure of bipartite graphs.



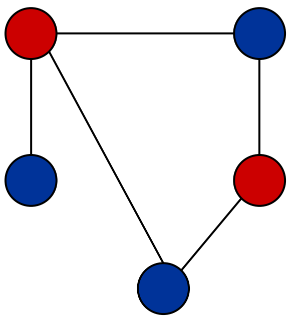
a bipartite graph G



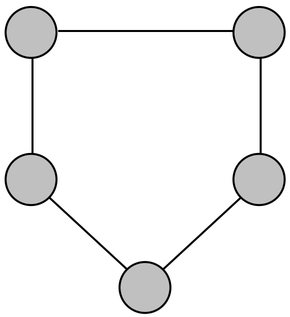
another drawing of G

Lemma. If a graph G is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone G.



bipartite
(2-colorable)

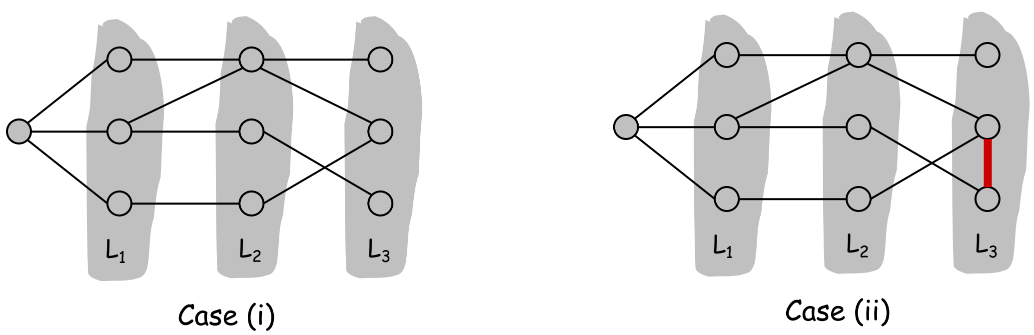


not bipartite
(not 2-colorable)

Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).



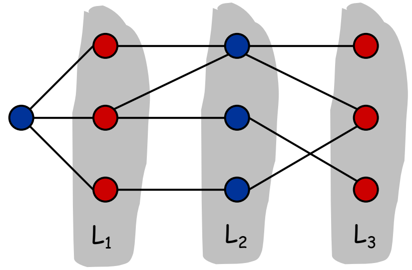
Lemma. Let G be a connected graph, and let L_0, …, L_k be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

Pf. (i)

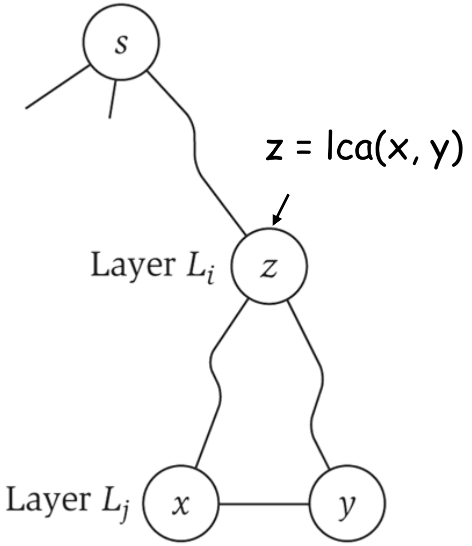
- Suppose no edge joins two nodes in the same layer.
- By previous lemma, this implies all edges join nodes on adjacent levels.
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.



Case (i)

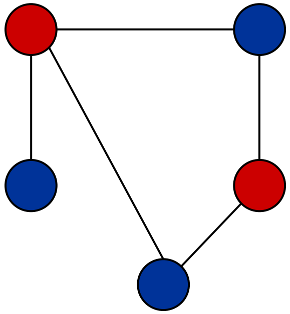
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.
(i) No edge of G joins two nodes of the same layer, and G is bipartite.
(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

Pf. (ii)
- Suppose (x, y) is an edge with x, y in same level L_j.
- Let $z = lca(x, y)$ = lowest common ancestor.
- Let L_i be level containing z.
- Consider cycle that takes edge from x to y, then path from y to z, then path from z to x.
- Its length is $\underbrace{1 +}_{(x,\,y)} \underbrace{(j-i)}_{\text{path from } y \text{ to } z} + \underbrace{(j-i)}_{\text{path from } z \text{ to } x}$, which is odd. ▪

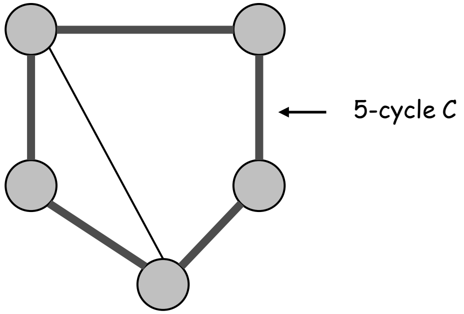


Obstruction to Bipartiteness

Corollary. A graph G is bipartite **iff** it contains no odd length cycle.



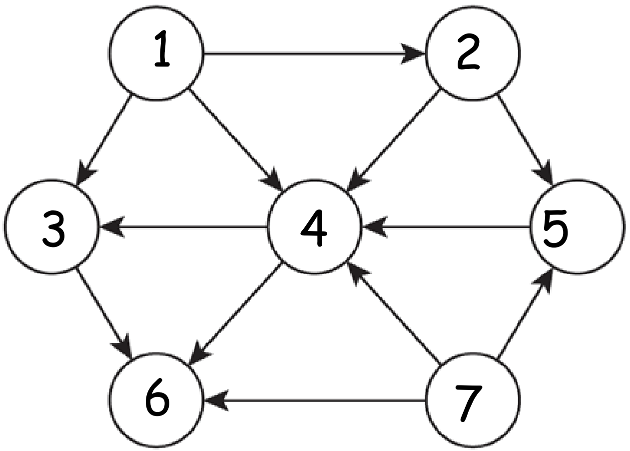
bipartite
(2-colorable)



not bipartite
(not 2-colorable)

3.5 Connectivity in Directed Graphs

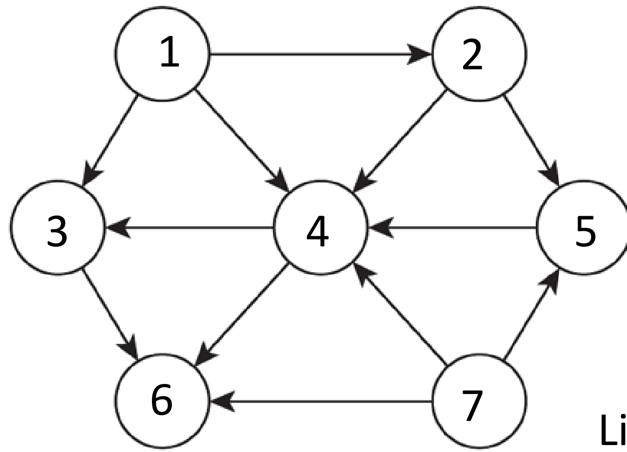
Directed graph. $G = (V, E)$
- Edge (u, v) goes from node u to node v.



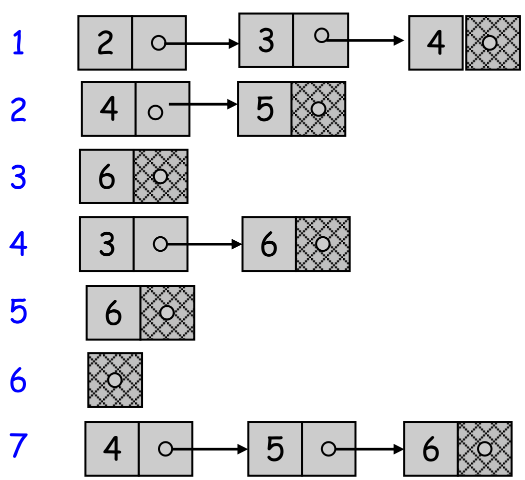
Ex. Web graph - hyperlink points from one web page to another.
- Directedness of graph is crucial.
- Modern web search engines exploit hyperlink structure to rank web pages by importance.

Rappresentazione di un grafo diretto: 2 liste di adiacenza

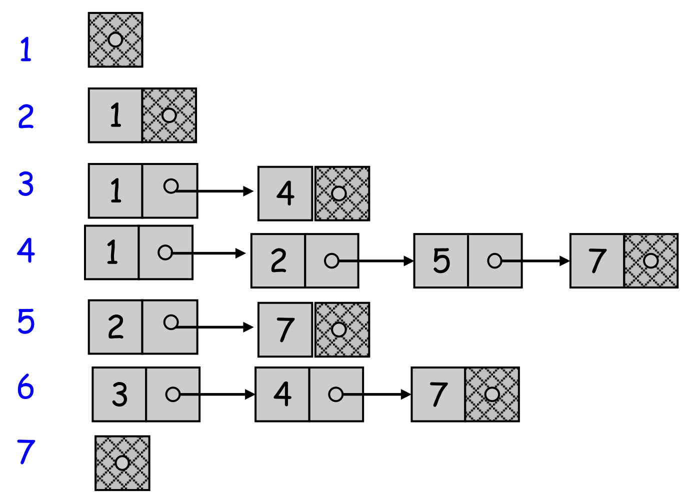
Lista di adiacenza: array di n liste indicizzate dai nodi.



Lista out



Lista in



Graph Search

Directed reachability. Given a node s, find all nodes reachable from s.

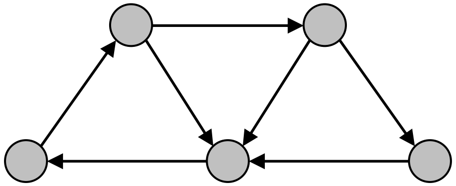
Directed s-t shortest path problem. Given two node s and t, what is the length of the shortest path from s to t?

Graph search. BFS extends naturally to directed graphs.

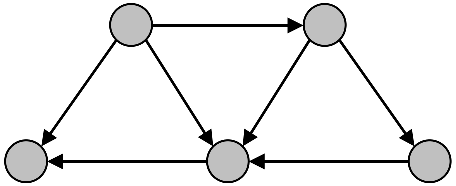
Strong Connectivity

Def. Nodes u and v are mutually reachable if there is a path from u to v and also a path from v to u.

Def. A graph is strongly connected if every pair of nodes is mutually reachable.



strongly connected



not strongly connected

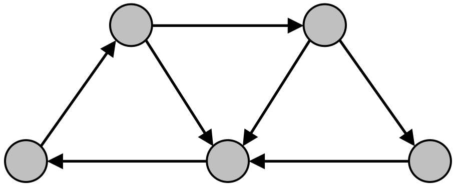
To test strong connectivity by definition it would be necessary to execute for any v in V, BFS(v).

This would result in time complexity $T(m,n) = O(\ n(m+n))$.

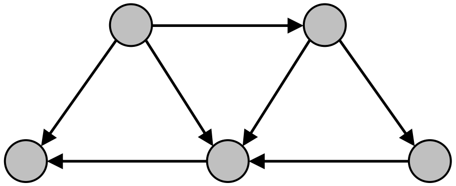
Strong Connectivity

Def. Nodes u and v are **mutually reachable** if there is a path from u to v and also a path from v to u.

Def. A graph is **strongly connected** if every pair of nodes is mutually reachable.

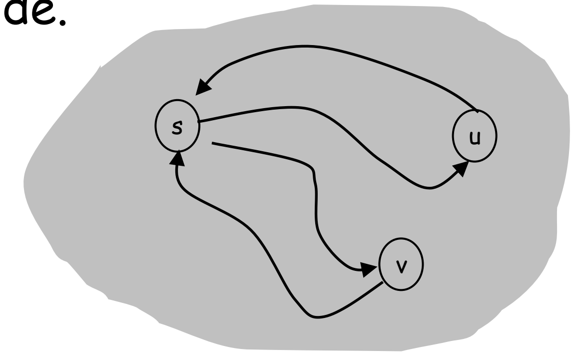


strongly connected



not strongly connected

Lemma. Let s be any node. G is strongly connected iff every node is reachable from s, and s is reachable from every node.



Pf. $\Rightarrow$ Follows from definition.

Pf. $\Leftarrow$ Path from u to v: concatenate u-s path with s-v path.
Path from v to u: concatenate v-s path with s-u path. ▪

Theorem. Can determine if G is strongly connected in $O(m + n)$ time.

Pf.

- Pick any node s.
- Run BFS from s in G.
- Run BFS from s in G^{rev}.
  reverse orientation of every edge in G
- Return true iff all nodes reached in both BFS executions.
- Correctness follows immediately from previous lemma. ▪

G $\qquad$ G^{rev}