Progettazione di Algoritmi **Anno Accademico 2023–2024**

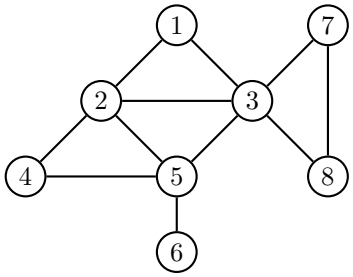
Note per la Lezione 22

Ugo Vaccaro

In questa lezione introdurremo il concetto di grafo, esamineremo le loro più comuni rappresentazioni ed introdurremo i primi problemi algoritmici su grafi.

Formalmente un grafo G consiste di una coppia di insiemi (V, E), dove V =insieme dei vertici (o nodi) del grafo e E =insieme degli archi tra coppie di vertici del grafo.

Esempio:



V={1,2,3,4,5,6,7,8}

E={1–2,1–3, 2–3, 2–4, 2–5,
4–5, 5–6, 3–5, 3–7, 3–8, 7–8}

Tipicamente gli archi descrivono "relazioni" tra coppie di "entità " (rappresentate dai vertici). I grafi trovano applicazioni in numerosissimi ambiti, ad esempio...

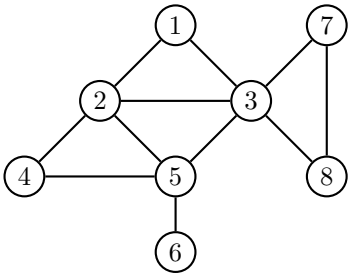
Grafi	Vertici	Archi
Reti di Trasporto	Incroci	Strade
Reti di Comunicazione	Computer	Fibre Ottiche
World Wide Web	Pagine Web	Hyperlinks
Sociale	Persone	Relazioni
Catene Alimentari	Specie Animali	Preda-Predatore
Sistemi Software	Funzioni	Chiamate di Funzioni
Scheduling	Attività	Vincoli di compatibilità
Circuiti	Porte	Connessioni

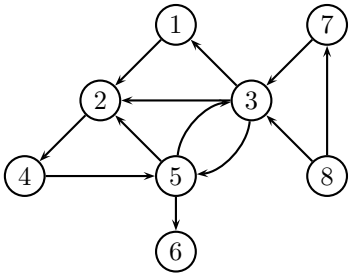
Gli archi di un grafo possono rappresentare sia relazioni "simmetriche" tra vertici (es., in una rete di calcolatori, se un calcolatore A è connesso ad un calcolatore B allora B è ovviamente simmetricamente connesso ad A), che relazioni "asimmetriche" (es., in un grafo che rappresenta la rete stradale in una città, a caua di eventuali sensi unici si può andare direttamente da un punto A ad un punto B ma non necessariamente vale il viceversa).

Per rappresentare relazioni simmetriche si usano grafi non diretti. Come dice la parola, in tali grafi gli archi non hanno direzioni e *non* si fa differenza tra l'arco dal vertice u al vertice v e l'arco dal vertice v al vertice u. Formalmente, un grafo G non diretto è rappresentato da una coppia $G = (V, E)$, dove V è l'insieme dei vertici, ed un arco $e \in E$ tra i vertici $u, v \in V$ consiste del sottoinsieme $\{u, v\} \subset V$ (per cui $\{u, v\} = \{v, u\}$).

Per rappresentare relazioni asimmetriche si usano grafi diretti. In essi gli archi hanno "direzioni" e *c'è* differenzia tra l'arco dal vertice u al vertice v e l'arco dal vertice v al vertice u. Formalmente, un grafo G diretto è rappresentato da una coppia $G = (V, E)$, dove V è sempre l'insieme dei vertici, ed un arco $e \in E$ dal vertice u ad il vertice v consiste della coppia *ordinata* (u, v) (per cui $(u, v) \neq (v, u)$, e se nel grafo c'è l'arco (u, v) non è detto che nel grafo che ci sia anche (v, u)).

Esempio: a sinistra grafo non diretto, a destra grafo diretto





V={1,2,3,4,5,6,7,8}

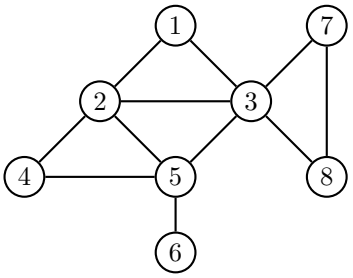
E={{1,2},{1,3}, {2,3}, {2,4}, {2,5},
{4,5}, {5,6}, {3,5}, {3,7}, {3,8}, {7,8}}

V={1,2,3,4,5,6,7,8}

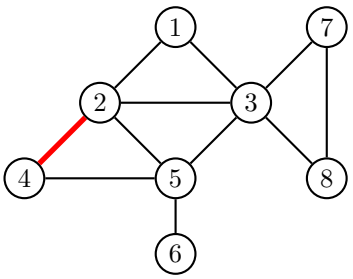
E={(1,2),(3,1), (3,2), (2,4), (5,2), (4,5)
(5,6), (3,5), (5,3), (7,3), (8,3), (8,7)}

Come descrivere un grafo $G = (V, E)$, $|V| = n$, $|E| = m$ ad un calcolatore?

Primo Metodo – Matrice di adiacenza:
Una matrice di adiacenza è una matrice binaria con n righe ed n colonne, (una riga ed una colonna per ciascun vertice del grafo) in cui l'elemento $A[i, j]$ nella riga i-esima e colonna j-esima di A è uguale a 1 se e solo se esiste nel grafo G l'arco tra i vertici i e j.



$$\mathrm{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



$$\mathrm{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & \mathbf{1} & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Pregi e difetti della matrice di adiacenza di un grafo.

Lo spazio richiesto è $\Theta(n^2)$, $n = |V|$.

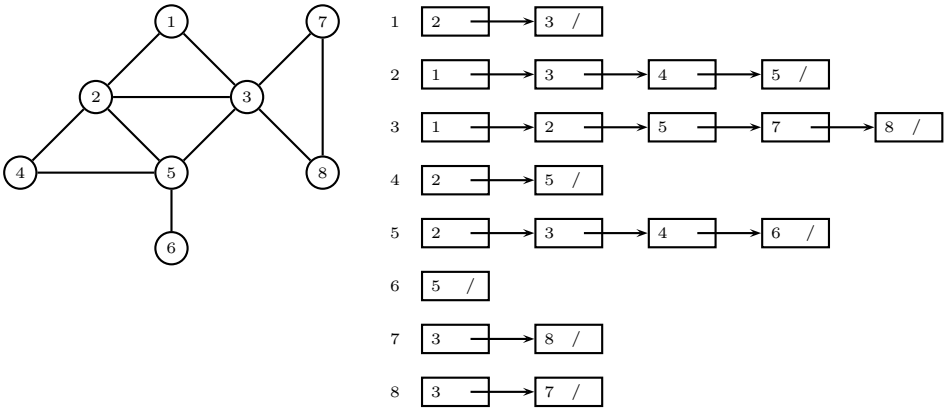
Verificare se c'è un arco tra i vertici i e j prende tempo $\Theta(1)$

Identificare tutti gli archi del grafo prende tempo $\Theta(n^2)$

Secondo Metodo: Liste di Adiacenza.

Le liste di adiacenza di un grafo consistono in un insieme di n liste a puntatori, una per ogni vertice del grafo La lista associata al generico nodo i contiene *tutti* i nodi j che sono connessi al nodo i mediante un arco. Tali vertici j sono detti vicini (o adiacenti) del nodo i nel grafo.

Esempio:



Pregi e difetti della matrice di adiacenza di un grafo.

Lo spazio richiesto è $\Theta(n + m)$, $n = |V|$, $m = |E|$.

Verificare se c'è un arco tra i vertici i e j prende tempo $\Theta(\texttt{grado}(i))$, dove $\texttt{grado}(i)$ =numero di nodi adiacenti al nodo i.

Identificare tutti gli archi del grafo prende tempo $\Theta(n + m)$.

Un pò di nomenclatura sui grafi.

Un cammino in un grafo $G = (V, E)$ non diretto è una sequenza di nodi $P = v_1 v_2 \dots v_k$ del grafo, tale $\{v_i, v_{i+1}\} \in E$ per ogni $i = 1, \dots, k - 1$.

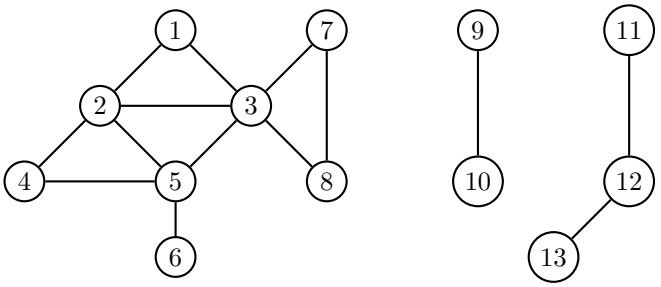
Un cammino è semplice se tutti i suoi vertici sono distinti. Ad esempio, nel grafo di sopra $P = 12456$ è un cammino semplice dal vertice 1 al vertice 6.

Un grafo è connesso se per ogni coppia di vertici i e j in esso c' è un cammino tra i e j.

Un ciclo in un grafo $G = (V, E)$ non diretto è un cammino $P = v_1 v_2 \dots v_k$ tale $v_i = v_k$, $k > 2$, ed i primi $k - 1$ nodi sono distinti.

Es.: Il grafo nella figura di sotto, con insieme di vertici $\{1, 2, \dots, 13\}$, **non** è connesso.

La distanza tra due vertici i e j è la lunghezza (misurata in numeri di archi) del più breve cammino tra i e j (se c'è).



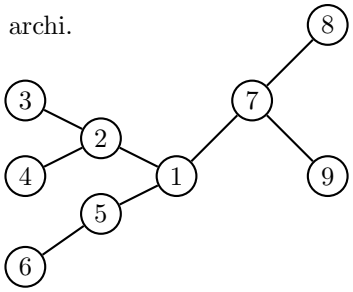
Es.: Il grafo in figura con insieme di vertici $\{1, 2, \ldots, 13\}$ **non** è connesso. Il grafo con vertici $\{1, 2, \ldots, 8\}$ è connesso. La sequenza di nodi $P = 1, 2, 5, 6$ è un cammino semplice dal nodo 1 al nodo 6. La distaza tra il vertice 4 ed il vertice 7 è pari a 3. La sequenza dei nodi $C = 1, 2, 4, 5, 3, 1$ è un ciclo.

Alberi.

Un grafo non diretto è detto albero se esso è connesso e non contiene cicli.

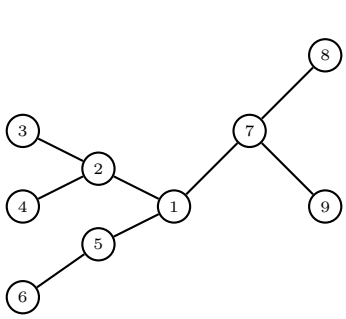
Fatto: Sia G un grafo connesso con n nodi. Ciascuna coppia delle seguenti affermazioni implica la terza (e quindi ogni coppia rappresenta una *equivalente* ed alternativa definizione di albero):

G è connesso.
G non contiene cicli.
G ha esattamente $n - 1$ archi.

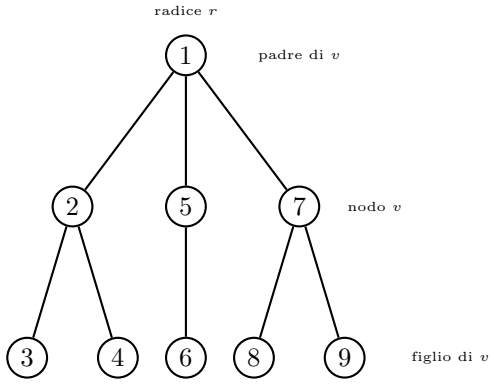


Alberi radicati

Dato un albero T, scegli un nodo, chiamalo radice, ed orienta gli archi "via" dalla radice.



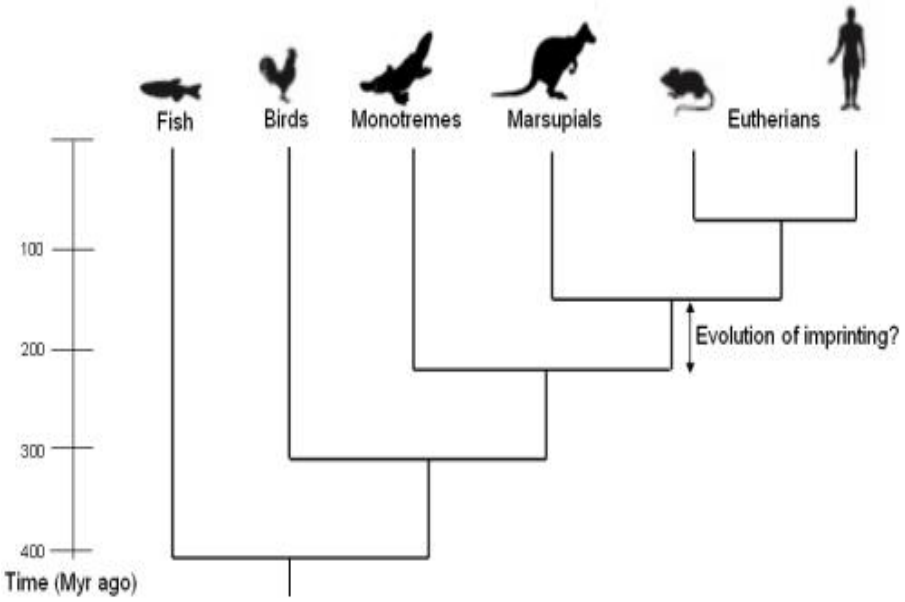
un albero



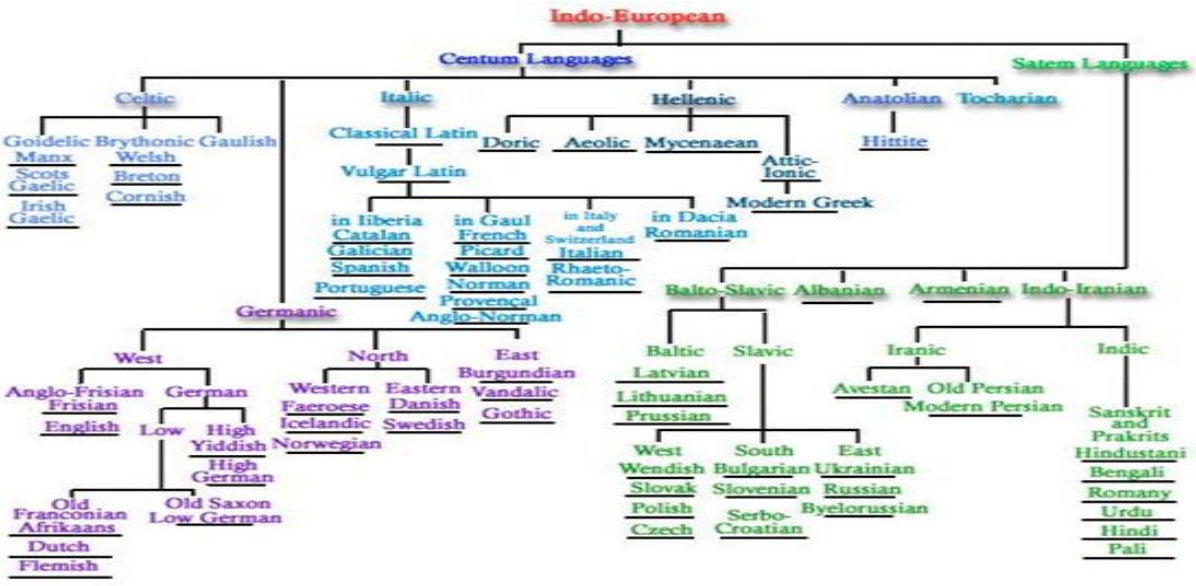
lo stesso albero, radicato in 1

Gli alberi sono molto usati per rappresentare strutture gerarchiche.

Es: Alberi Filogenetici per descrivere la storia evolutiva di specie



Es: Alberi linguistici per descrivere le relazioni tra lingue



Esplorazione (altrimenti dette visite) di grafi

Per esplorazione di grafi intendiamo le tecniche di attraversamento sistematico di grafi, allo scopo di aquisire informazioni sulla struttura del grafo e/o risolvere problemi algoritmici in grafi.

Esempio di problemi algoritmici che risolveremo mediante visite di grafi:

- Il Problema della $s-t$ connettività: dati due nodi s e t in un grafo G, esiste un cammino da s a t?
- Il Problema del più corto cammino da $s-t$: dati due nodi s e t in un grafo G, qual è la lunghezza del più corto cammino da s a t? (ed eventualmente trovarlo).
- ...

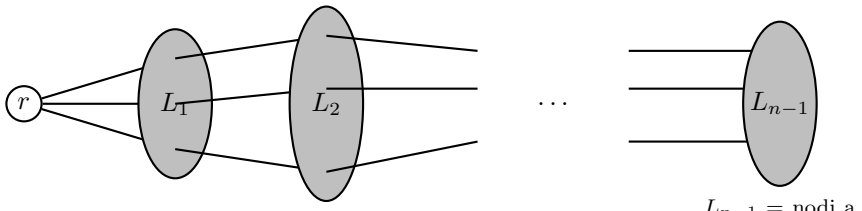
e relative applicazioni:

- Ricerca in "reti sociali"
- Attraversamento di labirinti
- Percorsi minimi in reti di comunicazione ⋮
- ...

Visita in ampiezza (breadth first) di un grafo $G = (V, E)$

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, per ogni vertice v del grafo tale distanza viene memorizzata in $d[v]$. Inoltre, l'algoritmo produce in output anche un albero T che contiene tutti i e soli i vertici u nel grafo G che sono raggiungibili mediante un cammino da s. Inoltre, il percorso dalla radice s dell'alberto T ad ogni nodo u è un cammino di lunghezza minimina da s a u nel grafo 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 $k = 1, 2, \ldots$



L_1= nodi a distanza 1 da r

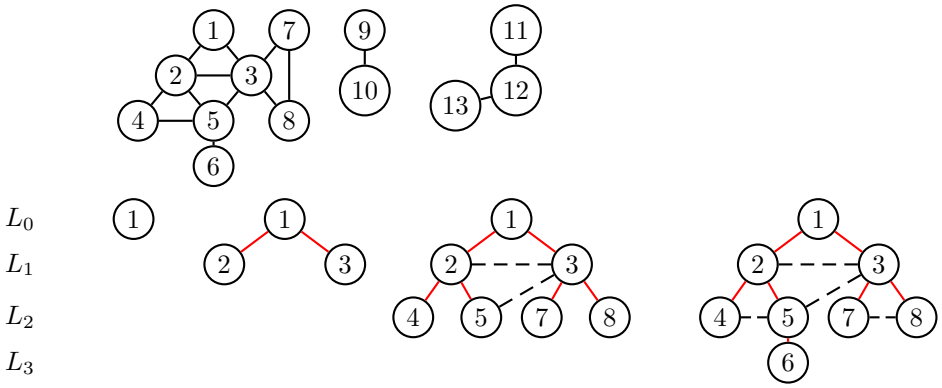
L_2= nodi a distanza 2 da r

L_{n-1} = nodi a distanza $n-1$ da r

Visita in ampiezza di un grafo $G = (V, E)$, partendo da un vertice input v: l'algoritmo informale

```
BFS(G, v) 1. marca v; d[v] ← 0
2. inserisci v in una lista L; inserisci v nell' albero T
3. while L non è vuota
4.   prendi il vertice u dalla testa della lista
5.   for ciascun vertice w non marcato e vicino di u
6.     marca w, d[w] ← d[u] + 1
7.     aggiungi w alla coda della lista L
8.     aggiungi w e l'arco (u, w) all'albero T
```

Vediamo come opera l'algoritmo sul seguente esempio di grafo, suppondo che l'algoritmo parta dal vertice $v = 1$. Gli archi rossi sono gli archi dell'albero prodotto dall'algoritmo $\texttt{BFS}(G, 1)$, gli archi tratteggiati sono gli archi del grafo G che **non** vengono inseriti nell'albero.



"Implementazione" dell'algoritmo $\texttt{BFS}(G, s)$

L'algoritmo $\texttt{BFS}(G, s)$ costruisce i livelli di nodi $L_1, L_2, \ldots$, dove L_i =insieme dei nodi a distanza i da s.

Per rappresentare ogni livello L_i useremo una *lista* $L[i]$

Quando esaminiamo un nodo $u \in L_i$, esamineremo tutti i gli archi della forma (u, v) e *se* $v \notin L_i$ allora occorrerà mettere v in L_{i+1}, quindi ci occorre un metodo veloce per stabilire questo fatto

Useremo un array $\texttt{Scoperto}[1 \ldots n]$ e porremo $\texttt{Scoperto}[i] = \texttt{true}$ ogni qualvolta la nostra visita trova il vertice i.

Per ogni nodo v calcoleremo il parametro $d[v]$ e porremo $d[v] = i$ se e solo se il nodo i viene messo nel livello L_i (ovvero se il nodo i si trova esattamente a distanza i dal nodo di partenza s)

Abbiamo descritto l'algoritmo $\texttt{BFS}$ facente uso di n liste separate $L[i]$, una per ogli livello L_i. Se ne può fare a meno ed usare una sola lista, che gestiremo come una coda (*first-in, first-out*).

In questo modo i nodi sono processati nell'ordine in cui essi vengono scoperti: ogni volta che un nodo viene scoperto per la prima volta viene aggiunto alla fine della coda e l'algoritmo processa sempre il nodo che si trova alla testa della coda.

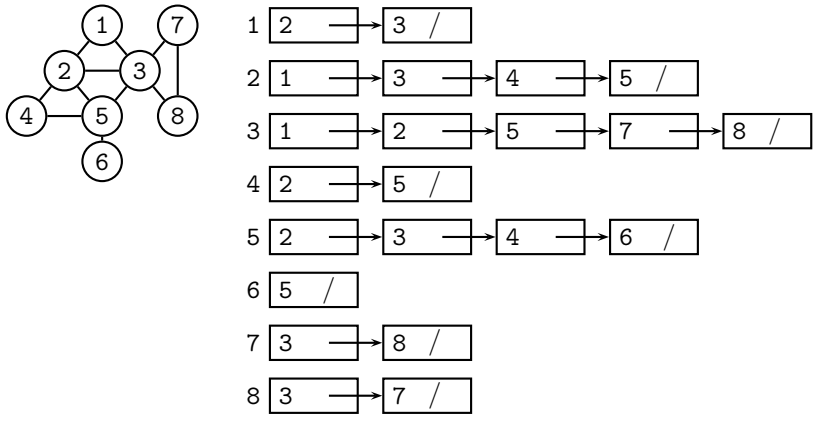
Se manteniamo i nodi scoperti in quest'ordine, allora i nodi nel livello L_i vengono esaminati chiaramente prima dei nodi del livello L_{i+1}

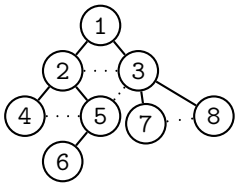
Quindi i nodi del livello L_i formano una sequenza di nodi consecutivi nella coda, e sono seguiti dalla sequenza consecutiva dei nodi nel livello L_{i+1}, e così via...

Pertanto quest'implementazione con un'unica coda produrrà lo stesso risultato dell'implementazione della $\texttt{BFS}$ prima descritta.

$\texttt{BFS}(G, s)$
1. Poni $\texttt{Scoperto}[s] = \texttt{true}$ e $\texttt{Scoperto}[v] = \texttt{false}$ $\forall v \neq s$
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) incidente su 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; $d[v] \leftarrow d[u] + 1$





La coda Q conterrà, in sequenza: $Q = \{1\}, Q = \emptyset, Q = \{2\}, Q = \{2, 3\}, Q = \{3\}, Q = \{3, 4\}, Q = \{3, 4, 5\}, Q = \{4, 5\}, Q = \{4, 5, 7\}, Q = \{4, 5, 7, 8\}, Q = \{5, 7, 8\}, Q = \{7, 8, 6\}, Q = \{8, 6\}, Q = \{6\}, Q = \emptyset$