Algoritmi di visita

Chiara Stortini

Università degli studi di Roma Tre

Corso di Teoria dei Grafi
23 aprile 2021

Introduzione

Nel corso di questo seminario vedremo alcuni algoritmi efficienti per la visita di un grafo, che sfruttano la possibilità di scomporre un grafo semplice in una foresta.

Introduzione

Sia $G(V, E)$ un grafo, e sia T un albero, sottografo di G.

- Se $V(T) = V(G)$, allora T è un albero generante di G, quindi G è connesso.
- Se $V(T) \subset V(G)$, allora:
  - se $\partial(T) = \emptyset$, allora G non è connesso;
  - se $\partial(T) \neq \emptyset$, per ogni lato $e = xy \in \partial(T)$, con $x \in V(T)$ e $y \in V(G) \setminus V(T)$, allora il sottografo di G ottenuto aggiungendo il vertice y e il lato $e = xy$ a T, è ancora un albero di G.

Introduzione

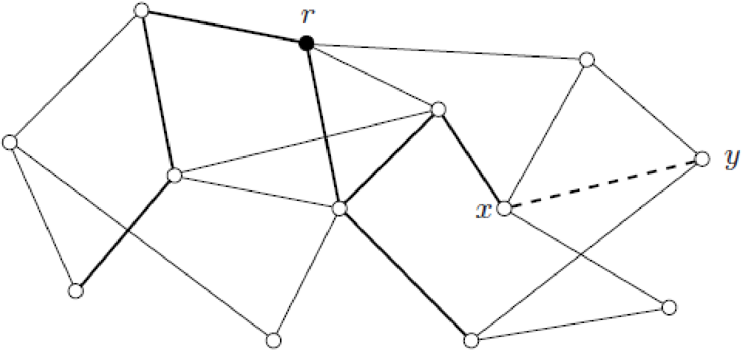
Sia $G(V, E)$ un grafo, e sia T un albero, sottografo di G.

- Se $V(T) = V(G)$, allora T è un albero generante di G, quindi G è connesso.
- Se $V(T) \subset V(G)$, allora:
  - se $\partial(T) = \emptyset$, allora G non è connesso;
  - se $\partial(T) \neq \emptyset$, per ogni lato $e = xy \in \partial(T)$, con $x \in V(T)$ e $y \in V(G) \setminus V(T)$, allora il sottografo di G ottenuto aggiungendo il vertice y e il lato $e = xy$ a T, è ancora un albero di G.

Introduzione

Sia $G(V, E)$ un grafo, e sia T un albero, sottografo di G.

- Se $V(T) = V(G)$, allora T è un albero generante di G, quindi G è connesso.
- Se $V(T) \subset V(G)$, allora:
  - se $\partial(T) = \emptyset$, allora G non è connesso;
  - se $\partial(T) \neq \emptyset$, per ogni lato $e = xy \in \partial(T)$, con $x \in V(T)$ e $y \in V(G) \setminus V(T)$, allora il sottografo di G ottenuto aggiungendo il vertice y e il lato $e = xy$ a T, è ancora un albero di G.


r
x
y

Introduzione

Usando questo metodo si può costruire un albero generante T di G.

Chiameremo tale procedura tree-search.

Qualunque tree-search permette di fare una visita del grafo. Pertanto è in grado di determinare se esso è connesso o meno. Tuttavia, a seconda dello scopo della visita possono essere attuati diversi algoritmi di tree-search.

Di seguito, senza perdita di generalità, assumiamo che tutti i grafi siano connessi e semplici.

Introduzione

Usando questo metodo si può costruire un albero generante T di G.

Chiameremo tale procedura tree-search.

Qualunque tree-search permette di fare una visita del grafo. Pertanto è in grado di determinare se esso è connesso o meno. Tuttavia, a seconda dello scopo della visita possono essere attuati diversi algoritmi di tree-search.

Di seguito, senza perdita di generalità, assumiamo che tutti i grafi siano connessi e semplici.

Introduzione

Usando questo metodo si può costruire un albero generante T di G.

Chiameremo tale procedura tree-search.

Qualunque tree-search permette di fare una visita del grafo. Pertanto è in grado di determinare se esso è connesso o meno. Tuttavia, a seconda dello scopo della visita possono essere attuati diversi algoritmi di tree-search.

Di seguito, senza perdita di generalità, assumiamo che tutti i grafi siano connessi e semplici.

Introduzione

Usando questo metodo si può costruire un albero generante T di G.

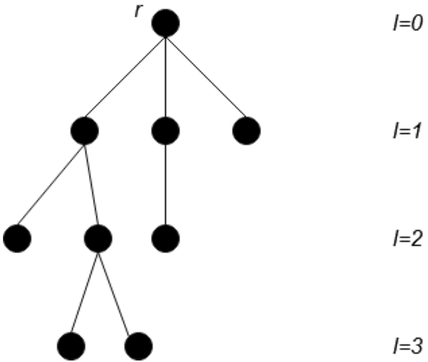
Chiameremo tale procedura tree-search.

Qualunque tree-search permette di fare una visita del grafo. Pertanto è in grado di determinare se esso è connesso o meno. Tuttavia, a seconda dello scopo della visita possono essere attuati diversi algoritmi di tree-search.

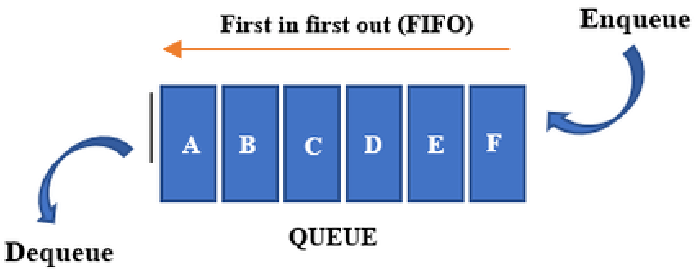
Di seguito, senza perdita di generalità, assumiamo che tutti i grafi siano connessi e semplici.

Livello di un vertice

Dato un albero T, definiamo: livello "ℓ" di un vertice di un albero, la sua profondità nell'albero.
In particolare, ponendo il livello della radice uguale a 0, avremo che il livello di un vertice è pari a 1 più il livello del vertice precedente.

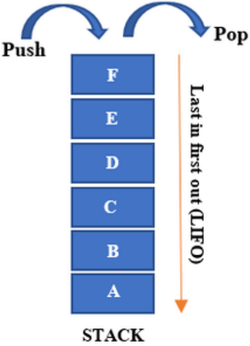


Definiamo fila (queue), un vettore Q in cui il primo elemento ad essere aggiunto è il primo che viene estratto (first in first out).



Pila

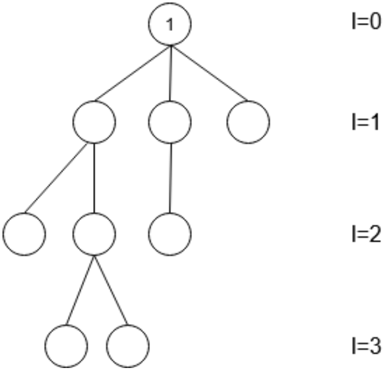
Definiamo pila (stack), un vettore S in cui l'ultimo elemento ad essere aggiunto è il primo che viene estratto (last in first out).



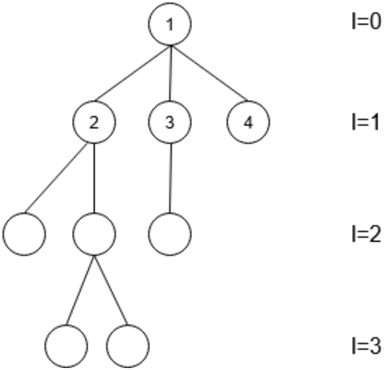
Ricerca in ampiezza

La ricerca in ampiezza (breadth-first search), è un tipo di visita in cui i vertici di un albero T di G vengono visitati partendo dal livello più basso, fino a quello più alto.

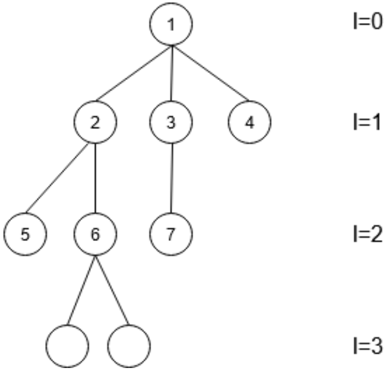
Visita in ampiezza



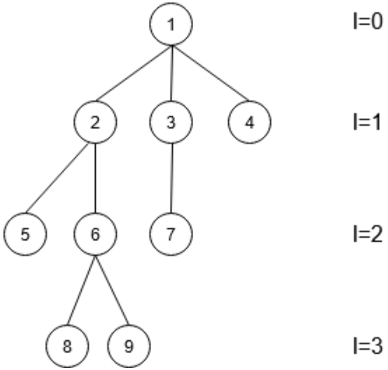
Visita in ampiezza



Visita in ampiezza



Visita in ampiezza



Visita in ampiezza

INPUT: una grafo connesso G;

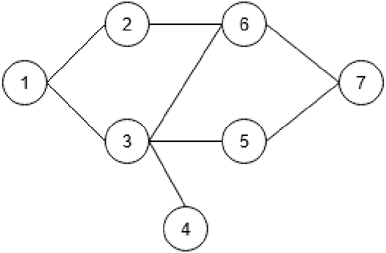
inizializza $i = 0$ e $Q = \emptyset$;
incrementa i di 1;
colora r di nero;
inizializza $\ell(r) = 0$ e $t(r) = 1$;
aggiungi r a Q;
while Q è non vuota **do**:
    considera la testa di Q, che chiameremo x;
    if x ha un vicino non colorato, che chiamiamo y, **then**:
        incrementa i di 1;
        colora y di nero;
        poni $p(y) = x$, $\ell(y) = \ell(x) + 1$ e $t(y) = i$;
        aggiungi y a Q;

Visita in ampiezza

    else:
        rimuovi x da Q;
    end if;
end while;
restituisci (p, ℓ, t).

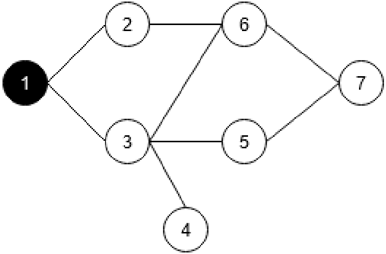
OUTPUT: un albero T di G con radice r, con p una funzione predecessore, ℓ una funzione di livello tale che $\ell(v) = d_G(r, v)$ per ogni $v \in V(G)$, e una funzione tempo t.

Esempio



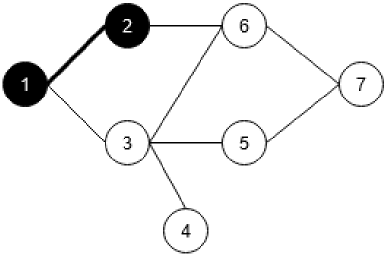
$Q = \varnothing$
$i = 0$

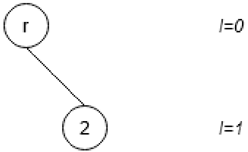
Esempio



$Q = (1)$
$i = 1,\ \ell(r) = 0,\ t(r) = 1$

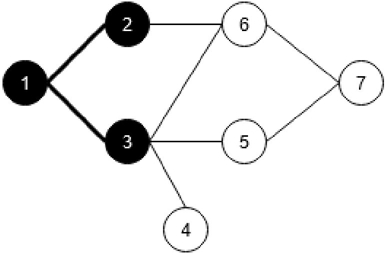
Esempio

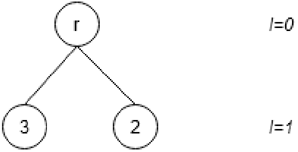




$Q = (1, 2)$
$i = 2,\ p(2) = 1,\ \ell(2) = 1,\ t(2) = 2$

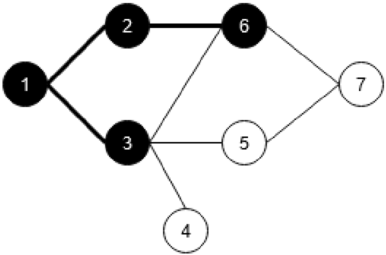
Esempio

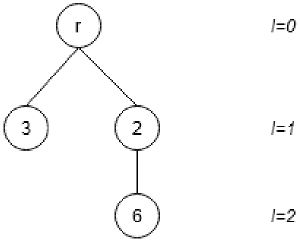




$Q = (1, 2, 3)$
$i = 3,\ p(3) = 1,\ \ell(3) = 1,\ t(3) = 3$

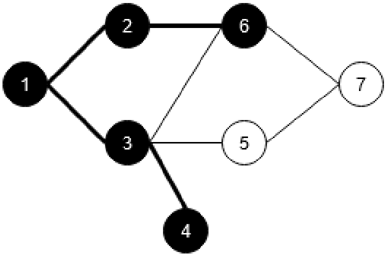
Esempio

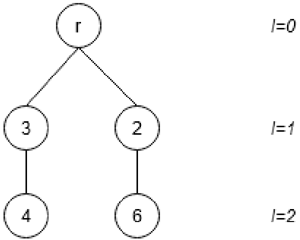




$Q = (2, 3, 6)$
$i = 4,\ p(6) = 2,\ \ell(6) = 2,\ t(6) = 4$

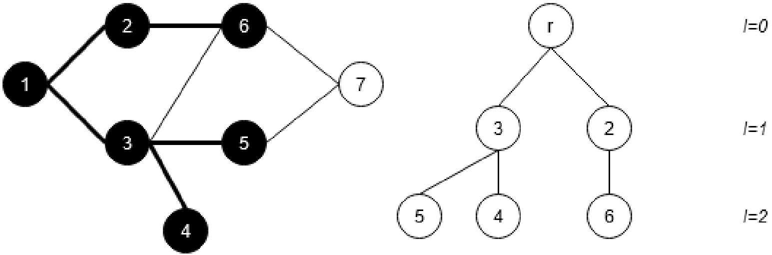
Esempio





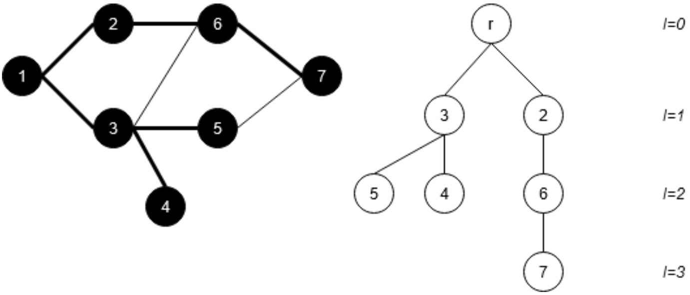
$$Q = (3, 6, 4)$$
$$i = 5,\ p(4) = 3,\ \ell(4) = 2,\ t(4) = 5$$

Esempio



$Q = (3, 6, 4, 5)$
$i = 6,\ p(5) = 3,\ \ell(5) = 2,\ t(5) = 6$

Esempio



$Q = (6, 4, 5, 7)$

$i = 7,\ p(7) = 6,\ \ell(7) = 3,\ t(7) = 7$

Teorema

Sia G un grafo connesso, e sia T l'albero ottenuto da G tramite l'algoritmo di ricerca in ampiezza, con radice r. Allora:

a) *per ogni vertice v di G, il livello di v in T è dato da: $\ell(v) = d_T(r, v)$;*

b) *per ogni arco di G che unisce vertici di T che si trovano allo stesso livello o su livelli consecutivi, allora:*

$$|\ell(u) - \ell(v)| \leq 1, \text{ per ogni } uv \in E(G).$$

Visita in ampiezza

Teorema

Sia G un grafo connesso. I valori della funzione di livello ℓ dati dall'algoritmo di ricerca in ampiezza, sono le distanze dei vertici $v \in V(G)$ dalla radice r:

$$\ell(v) = d_G(r, v), \text{ per ogni } v \in V(G).$$

Dimostrazione.

Dal punto a) del teorema precedente: $\ell(v) = d_T(r, v)$.

Inoltre $d_T(r, v) \geq d_G(r, v)$, essendo T un sottografo di G.

Allora $\ell(v) \geq d_G(r, v)$.

Dimostriamo che vale anche l'altra disuguaglianza procedendo per induzione sulla lunghezza del cammino più breve tra r e v.

Sia P il cammino più breve tra r e v su G, con $v \neq r$, e sia u il predecessore di v su P. Allora rPu è il cammino più breve tra r e u, e $d_G(r, u) = d_G(r, v) - 1$.

Per induzione $\ell(u) \leq d_G(r, u)$, per il punto b) del teorema precedente: $\ell(v) - \ell(u) \leq 1$.

Allora: $\ell(v) \leq \ell(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v)$.

Dalle due disuguaglianze, segue la tesi. □

Visita in ampiezza

Dimostrazione.
Dal punto a) del teorema precedente: $\ell(v) = d_T(r, v)$.
Inoltre $d_T(r, v) \geq d_G(r, v)$, essendo T un sottografo di G.
Allora $\ell(v) \geq d_G(r, v)$.
Dimostriamo che vale anche l'altra disuguaglianza procedendo per induzione sulla lunghezza del cammino più breve tra r e v.
Sia P il cammino più breve tra r e v su G, con $v \neq r$, e sia u il predecessore di v su P. Allora rPu è il cammino più breve tra r e u, e $d_G(r, u) = d_G(r, v) - 1$.
Per induzione $\ell(u) \leq d_G(r, u)$, per il punto b) del teorema precedente: $\ell(v) - \ell(u) \leq 1$.
Allora: $\ell(v) \leq \ell(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v)$.
Dalle due disuguaglianze, segue la tesi. $\square$

Dimostrazione.

Dal punto a) del teorema precedente: $\ell(v) = d_T(r, v)$.
Inoltre $d_T(r, v) \geq d_G(r, v)$, essendo T un sottografo di G.
Allora $\ell(v) \geq d_G(r, v)$.
Dimostriamo che vale anche l'altra disuguaglianza procedendo per induzione sulla lunghezza del cammino più breve tra r e v.
Sia P il cammino più breve tra r e v su G, con $v \neq r$, e sia u il predecessore di v su P. Allora rPu è il cammino più breve tra r e u, e $d_G(r, u) = d_G(r, v) - 1$.
Per induzione $\ell(u) \leq d_G(r, u)$, per il punto b) del teorema precedente: $\ell(v) - \ell(u) \leq 1$.
Allora: $\ell(v) \leq \ell(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v)$.
Dalle due disuguaglianze, segue la tesi. $\square$

Dimostrazione.

Dal punto a) del teorema precedente: $\ell(v) = d_T(r, v)$.
Inoltre $d_T(r, v) \geq d_G(r, v)$, essendo T un sottografo di G.
Allora $\ell(v) \geq d_G(r, v)$.
Dimostriamo che vale anche l'altra disuguaglianza procedendo per induzione sulla lunghezza del cammino più breve tra r e v.
Sia P il cammino più breve tra r e v su G, con $v \neq r$, e sia u il predecessore di v su P. Allora rPu è il cammino più breve tra r e u, e $d_G(r, u) = d_G(r, v) - 1$.
Per induzione $\ell(u) \leq d_G(r, u)$, per il punto b) del teorema precedente: $\ell(v) - \ell(u) \leq 1$.
Allora: $\ell(v) \leq \ell(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v)$.
Dalle due disuguaglianze, segue la tesi. □

Dimostrazione.

Dal punto a) del teorema precedente: $\ell(v) = d_T(r, v)$.
Inoltre $d_T(r, v) \geq d_G(r, v)$, essendo T un sottografo di G.
Allora $\ell(v) \geq d_G(r, v)$.
Dimostriamo che vale anche l'altra disuguaglianza procedendo per induzione sulla lunghezza del cammino più breve tra r e v.
Sia P il cammino più breve tra r e v su G, con $v \neq r$, e sia u il predecessore di v su P. Allora rPu è il cammino più breve tra r e u, e $d_G(r, u) = d_G(r, v) - 1$.
Per induzione $\ell(u) \leq d_G(r, u)$, per il punto b) del teorema precedente: $\ell(v) - \ell(u) \leq 1$.
Allora: $\ell(v) \leq \ell(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v)$.
Dalle due disuguaglianze, segue la tesi. $\square$

Visita in profondità

La ricerca in profondità (depth-first search), è un tipo di visita in cui viene esaminato per primo l'elenco di adiacenza dell'ultimo vertice aggiunto all'albero T, che chiamiamo x.
Se tale vertice ha un vicino non in T, lo aggiungiamo.
In caso contrario torniamo indietro fino all'ultimo vertice aggiunto prima di x ed esaminiamo i suoi vicini. E così via.

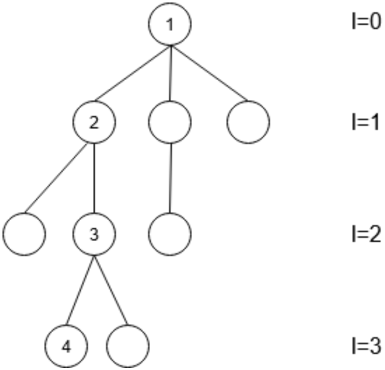
Visita in profondità

La ricerca in profondità (depth-first search), è un tipo di visita in cui viene esaminato per primo l'elenco di adiacenza dell'ultimo vertice aggiunto all'albero T, che chiamiamo x. Se tale vertice ha un vicino non in T, lo aggiungiamo. In caso contrario torniamo indietro fino all'ultimo vertice aggiunto prima di x ed esaminiamo i suoi vicini. E così via.

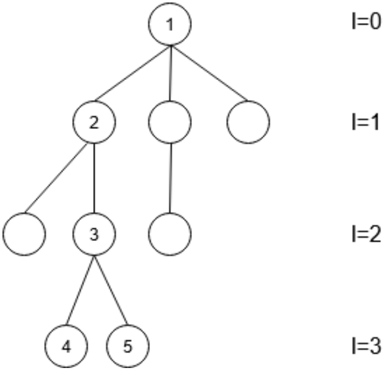
Visita in profondità

La ricerca in profondità (depth-first search), è un tipo di visita in cui viene esaminato per primo l'elenco di adiacenza dell'ultimo vertice aggiunto all'albero T, che chiamiamo x. Se tale vertice ha un vicino non in T, lo aggiungiamo. In caso contrario torniamo indietro fino all'ultimo vertice aggiunto prima di x ed esaminiamo i suoi vicini. E così via.

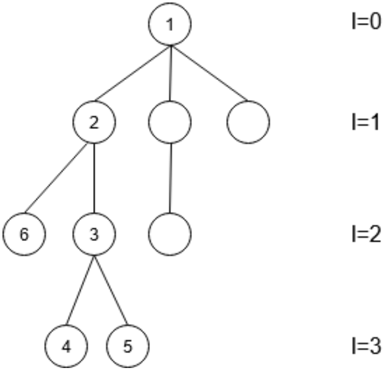
Visita in profondità



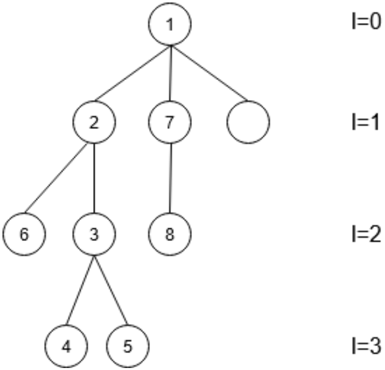
Visita in profondità



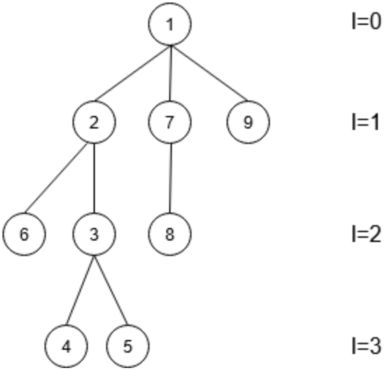
Visita in profondità



Visita in profondità



Visita in profondità



Visita in profondità

INPUT: un grafo connesso G

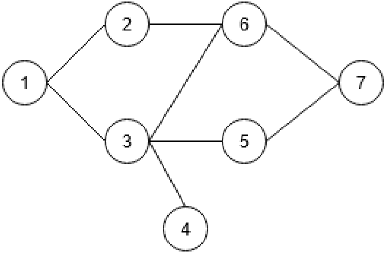
inizializza $i = 0$ e $S = \varnothing$;
scegli un vertice e chiamalo r (sarà la nostra radice);
incrementa i di 1;
colora r di nero;
inizializza $f_1(r) = i$;
aggiungi r ad S;
while S è non vuoto **do**:
    considera la cima di S, che chiameremo x;
    incrementa i di 1;
    if x ha un vicino non colorato, che chiamiamo y, **then**:
        colora y di nero;
        poni $p(y) = x$ e $f_1(y) = i$;
        aggiungi y ad S;

Visita in profondità

    else:
        poni $f_2(x) = i$;
        rimuovi x da S;
    end if;
end while;
restituisci (p, f_1, f_2).

OUTPUT: un albero generante di G con radice r, una funzione predecessore p, due funzioni tempo: f_1 e f_2.

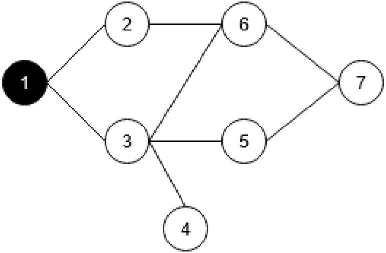
Visita in profondità



$S = \varnothing$

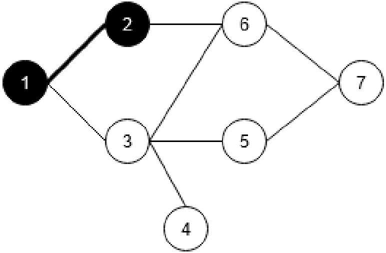
$i = 0$

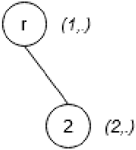
Visita in profondità



$S = (1)$
$i = 1,\ f_1(1) = 1$

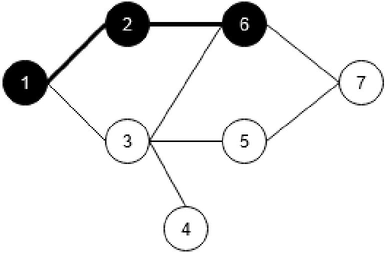
Visita in profondità

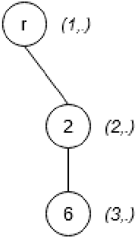




$S = (1, 2)$
$i = 2,\ p(2) = r,\ f_1(2) = 2$

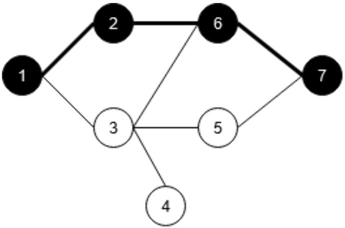
Visita in profondità

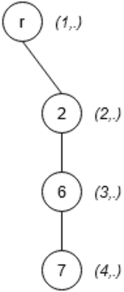




$$S = (1, 2, 6)$$
$$i = 3,\ p(6) = 2,\ f_1(6) = 3$$

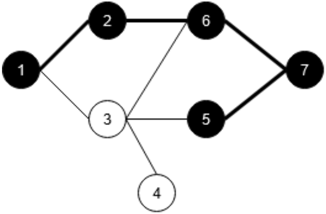
Visita in profondità

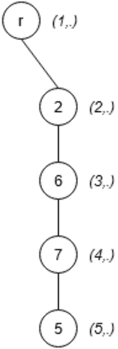




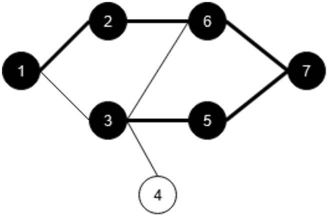
$S = (1, 2, 6, 7)$
$i = 4, \; p(7) = 6, \; f_1(7) = 4$

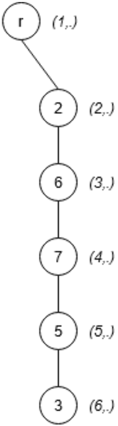
Visita in profondità



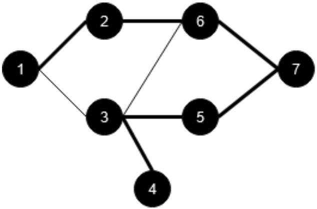


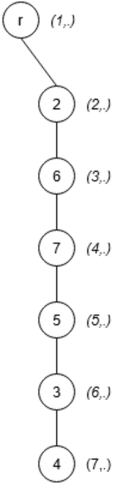
$S = (1, 2, 6, 7, 5)$
$i = 5,\ p(5) = 7,\ f_1(5) = 5$





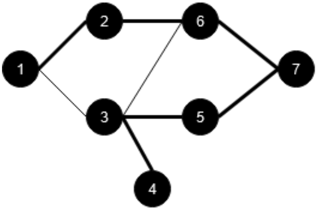
$S = (1, 2, 6, 7, 5, 3)$
$i = 6,\ p(3) = 5,\ f_1(3) = 6$

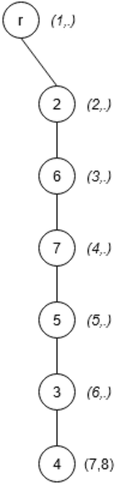




$S = (1, 2, 6, 7, 5, 3, 4)$
$i = 7,\ p(4) = 3,\ f_1(4) = 7$

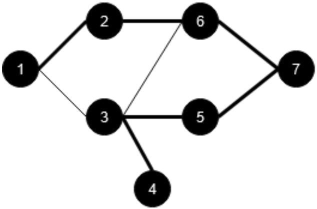
Visita in profondità

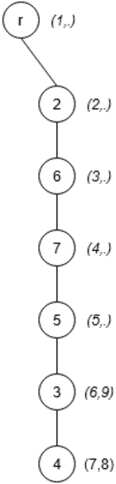




$$S = (1, 2, 6, 7, 5, 3, 4)$$
$$i = 8, \; f_2(4) = 8$$

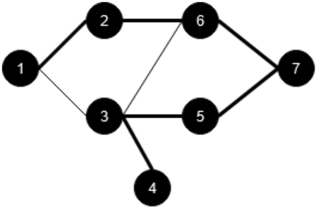
Visita in profondità

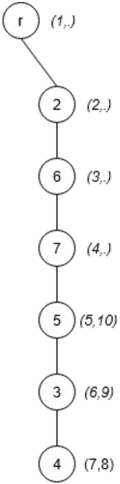




$S = (1, 2, 6, 7, 5, 3)$
$i = 9, \; f_2(3) = 9$

Visita in profondità

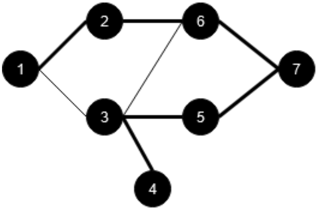


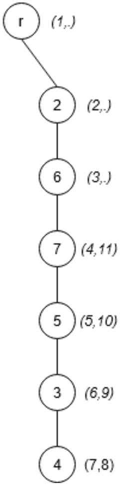


$S = (1, 2, 6, 7, 5)$

$i = 10,\ f_2(5) = 10$

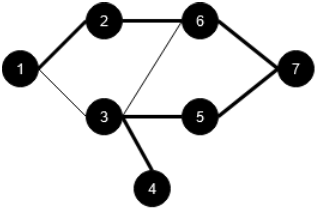
Visita in profondità

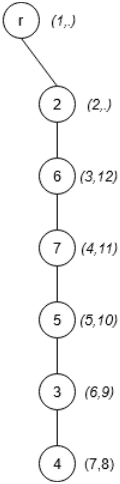




$S = (1, 2, 6, 7)$
$i = 11, \; f_2(7) = 11$

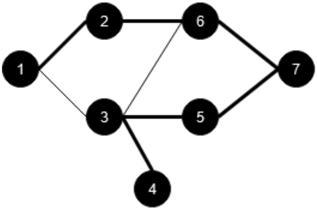
Visita in profondità

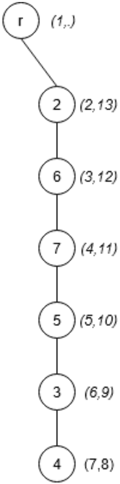




$S = (1, 2, 6)$
$i = 12, \ f_2(6) = 12$

Visita in profondità

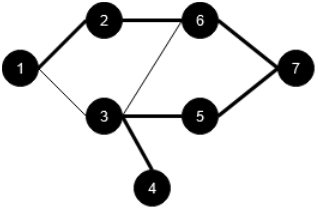


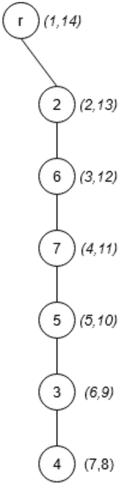


$S = (1, 2)$

$i = 13,\ f_2(2) = 13$

Visita in profondità





$S = (1)$
$i = 14,\ f_2(1) = 14$

Teorema

Siano u e v due vertici di G, con $f_1(u) < f_1(v)$.

a) *Se u e v sono adiacenti in G, allora $f_2(v) < f_2(u)$.*

b) *u è un predecessore di v in T se e solo se $f_2(v) < f_2(u)$.*

Dimostrazione.

a) Seguendo i passi $8-12$ dell'algoritmo di visita in profondità, il vertice u viene rimosso da S solo quando tutti i suoi vicini sono stati aggiunti alla pila S. Poichè $f_1(u) < f_1(v)$, allora v viene aggiunto ad S quando u è ancora presente in S; e u non può essere rimosso da S prima di v. Allora $f_2(v) < f_2(u)$.

Dimostrazione.

b) Supponiamo che u sia un predecessore di v in T. Seguendo i passi 9 e 12 dell'algoritmo di visita in profondità, il valore di f_1 aumenta ad ogni passo lungo il cammino uTv. Applicando il punto a) ad ogni ponte di tale cammino, segue $f_2(v) < f_2(u)$.

Ora supponiamo che u non sia un predecessore di v in T. Poichè $f_1(u) < f_1(v)$, allora neanche v è un predecessore di u. Segue che u non fa parte del cammino rTv e v non fa parte del cammino rTu.

Dimostrazione.

b) Supponiamo che u sia un predecessore di v in T. Seguendo i passi 9 e 12 dell'algoritmo di visita in profondità, il valore di f_1 aumenta ad ogni passo lungo il cammino uTv. Applicando il punto a) ad ogni ponte di tale cammino, segue $f_2(v) < f_2(u)$.
Ora supponiamo che u non sia un predecessore di v in T. Poichè $f_1(u) < f_1(v)$, allora neanche v è un predecessore di u. Segue che u non fa parte del cammino rTv e v non fa parte del cammino rTu.

Dimostrazione.

Sia n, l'ultimo vertice in comune tra i cammini rTv e rTu. Essendo $f_1(u) < f_1(v)$, allora il discendente di n nel cammino rTv, viene aggiunto alla pila S solo dopo che il discendente di n sul cammino rTu è stato rimosso. In particolare u viene addiunto ad S, solo dopo che v è stato rimosso, allora $f_2(u) < f_1(v)$. Poichè $f_1(v) < f_2(v)$, allora $f_2(u) < f_2(v)$. □

Teorema

Sia T un albero di visita di un grafo G trovato tramite algoritmo di visita in profondità. Allora ogni ponte di G unisce vertici adiacenti in T.

Vertici di taglio e blocchi

In un grafo che rappresenta una rete di comunicazioni, i vertici di taglio di un grafo connesso, rappresentano i centri in cui un guasto interromperebbe le comunicazioni.

È quindi importante identificare questi siti, in modo che possano essere prese precauzioni per ridurre la vulnerabilità della rete.

Tarjan, nel 1972 mostrò come questo problema potesse essere risolto efficientemente con una ricerca in profondità.

Vertici di taglio e blocchi

In un grafo che rappresenta una rete di comunicazioni, i vertici di taglio di un grafo connesso, rappresentano i centri in cui un guasto interromperebbe le comunicazioni.

È quindi importante identificare questi siti, in modo che possano essere prese precauzioni per ridurre la vulnerabilità della rete.

Tarjan, nel 1972 mostrò come questo problema potesse essere risolto efficientemente con una ricerca in profondità.

Vertici di taglio e blocchi

In un grafo che rappresenta una rete di comunicazioni, i vertici di taglio di un grafo connesso, rappresentano i centri in cui un guasto interromperebbe le comunicazioni.

È quindi importante identificare questi siti, in modo che possano essere prese precauzioni per ridurre la vulnerabilità della rete.

Tarjan, nel 1972 mostrò come questo problema potesse essere risolto efficientemente con una ricerca in profondità.

Vertici di taglio e blocchi

Vertici di taglio e blocchi

Vediamo ora come una ricerca in profondità può essere usata per trovare i vertici di taglio e i blocchi di un grafo connesso.

Sia T un albero trovato tramite l'algoritmo di ricerca in profondità di G, e sia B un blocco di G, allora $T \cap B$ è un sotto-albero di G. Inoltre, essendo T un albero radicato, possiamo associare a B un vertice unico: la radice dell'albero $T \cap B$. Chiameremo tale vertice radice di B rispetto a T. Esso è il primo vertice di B ad essere incorporato a T.

Notiamo che i vertici di taglio di G sono proprio le radici dei blocchi. Allora per identificare i vertici di taglio e i blocchi di G è sufficiente individuare tali radici. Questo può essere fatto durante l'esecuzione della ricerca in profondità.

Vertici di taglio e blocchi

Vediamo ora come una ricerca in profondità può essere usata per trovare i vertici di taglio e i blocchi di un grafo connesso.

Sia T un albero trovato tramite l'algoritmo di ricerca in profondità di G, e sia B un blocco di G, allora $T \cap B$ è un sotto-albero di G. Inoltre, essendo T un albero radicato, possiamo associare a B un vertice unico: la radice dell'albero $T \cap B$. Chiameremo tale vertice radice di B rispetto a T. Esso è il primo vertice di B ad essere incorporato a T.

Notiamo che i vertici di taglio di G sono proprio le radici dei blocchi. Allora per identificare i vertici di taglio e i blocchi di G è sufficiente individuare tali radici. Questo può essere fatto durante l'esecuzione della ricerca in profondità.

Vertici di taglio e blocchi

Vediamo ora come una ricerca in profondità può essere usata per trovare i vertici di taglio e i blocchi di un grafo connesso.

Sia T un albero trovato tramite l'algoritmo di ricerca in profondità di G, e sia B un blocco di G, allora $T \cap B$ è un sotto-albero di G. Inoltre, essendo T un albero radicato, possiamo associare a B un vertice unico: la radice dell'albero $T \cap B$. Chiameremo tale vertice radice di B rispetto a T. Esso è il primo vertice di B ad essere incorporato a T.
Notiamo che i vertici di taglio di G sono proprio le radici dei blocchi. Allora per identificare i vertici di taglio e i blocchi di G è sufficiente individuare tali radici. Questo può essere fatto durante l'esecuzione della ricerca in profondità.

Vertici di taglio e blocchi

Definiamo la funzione $f_1^* : V \to N$ definita nel seguente modo:

- Se un predecessore di v può essere raggiunto partendo da v attraverso un cammino formato unicamente da lati di T e al più il lato che unisce v al suo predecessore non appartiene a T, $f_1^*(v)$ è l'ultimo valore f_1 di tale predecessore.
- In caso contrario $f_1^*(v) = f_1(v)$.

Osserviamo che il vertice v è la radice di un blocco B di G se e solo se v ha un figlio w tale che $f_1^*(w) \geq f_1(v)$.

Vertici di taglio e blocchi

Definiamo la funzione $f_1^* : V \to \mathbb{N}$ definita nel seguente modo:

- Se un predecessore di v può essere raggiunto partendo da v attraverso un cammino formato unicamente da lati di T e al più il lato che unisce v al suo predecessore non appartiene a T, $f_1^*(v)$ è l'ultimo valore f_1 di tale predecessore.
- In caso contrario $f_1^*(v) = f_1(v)$.

Osserviamo che il vertice v è la radice di un blocco B di G se e solo se v ha un figlio w tale che $f_1^*(w) \geq f_1(v)$.

Vertici di taglio e blocchi

Definiamo la funzione $f_1^* : V \to \mathbb{N}$ definita nel seguente modo:

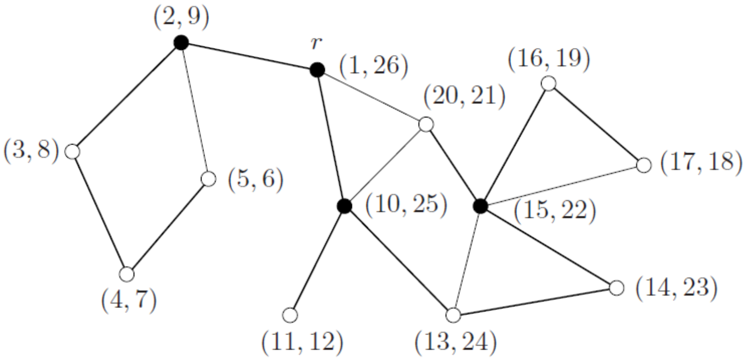
- Se un predecessore di v può essere raggiunto partendo da v attraverso un cammino formato unicamente da lati di T e al più il lato che unisce v al suo predecessore non appartiene a T, $f_1^*(v)$ è l'ultimo valore f_1 di tale predecessore.
- In caso contrario $f_1^*(v) = f_1(v)$.

Osserviamo che il vertice v è la radice di un blocco B di G se e solo se v ha un figlio w tale che $f_1^*(w) \geq f_1(v)$.

Vertici di taglio e blocchi

La funzione f_1^* può essere calcolata durante l'esecuzione dell'algoritmo di visita in profondità, e contemporaneamente si possono trovare le radici dei blocchi. Pertanto le radici dei blocchi B di G possono essere trovate in tempo lineare.

Vertici di taglio e blocchi



Grazie per l'attenzione!

Bibliografia

[1] J. A. Bondy, U. S. R. Murty, *Graph Theory*, Graduate Text in Mathematics 244, 2008, Capitolo 6.1.