Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Sviluppo:

$$T\left(\frac{n}{2}\right) = 5T\left(\frac{n}{4}\right) + \frac{n}{2}$$
$$T\left(\frac{n}{4}\right) = 5T\left(\frac{n}{8}\right) + \frac{n}{4}$$

Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Sostituzione:

$$T(n) = n + \frac{5}{2}n + \left(\frac{5}{2}\right)^2 n + 5^3 T\left(\frac{n}{8}\right)$$
$$= n \sum_{0 \leq i \leq s} \left(\frac{5}{2}\right)^i$$

Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Terminazione per $n = 1$, ogni volta dimezzo:

$$\frac{n}{2^s} = 1$$
$$2^s = n$$
$$s = \log_2(n)$$

Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Sostituzione:

$$\begin{aligned} T(n) &= n \sum_{0 \leq i \leq \log_2(n)} \left(\frac{5}{2}\right)^i \\ &= n \left(\frac{\left(\frac{5}{2}\right)^{\log_2(n)+1} - 1}{\frac{5}{2} - 1} \right) \end{aligned}$$

Risolvere tramite metodo dello sviluppo l'equazione

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Sostituzione:

$$n\left(\frac{\left(\frac{5}{2}\right)^{\log_2(n)+1} - 1}{\frac{5}{2} - 1}\right) = n\frac{2}{3}\left(\frac{5}{2} \cdot \frac{5^{\log_2(n)}}{2^{\log_2(n)}} - 1\right)$$
$$= n\frac{1}{3}\left(5 \cdot \frac{5^{\log_2(n)}}{n} - 2\right)$$
$$= n\frac{1}{3}\left(5 \cdot \frac{n^{\log_2 5}}{n} - 2\right)$$
$$= \frac{1}{3}\left(5 \cdot n^{\log_2 5} - 2n\right)$$

Equazioni alle ricorrenze

Risolvere tramite metodo dello sviluppo l'equazione

$$T\left(n\right) = 5T\left(\frac{n}{2}\right) + n$$
$$T\left(1\right) = 1$$

Sostituzione:

$$T\left(n\right) = \frac{1}{3}\left(5 \cdot n^{\log_2 5} - 2n\right)$$

Quindi:

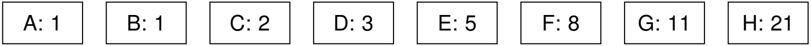
$$T\left(n\right) = O\left(n^{\log_2 5}\right)$$

Codice Huffman

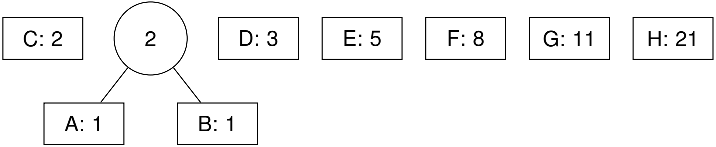
Determinare il codice di Huffman ottimo per i seguenti caratteri con le frequenze specificate:

A: 1   B: 1   C: 2   D: 3   E: 5   F: 8   G: 11   H: 21

Memorizziamo le informazioni in una coda a priorità

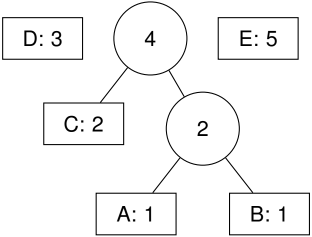


Estraiamo i due elementi a frequenza minore, A e B, e creiamo un nuovo albero che fonde i due nodi. Il nuovo albero ha frequenza 2, e va reinserito nella coda a priorità



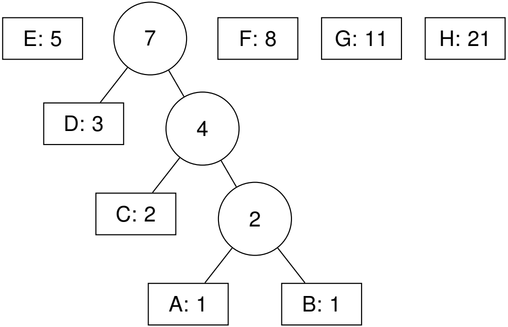
Codice Huffman

Estraiamo nuovamente i due elementi a frequenza minore, C e l'albero che contiene A, B, e li fondiamo in un novo albero con frequenza 4. Reinseriamo il nuovo albero nella coda

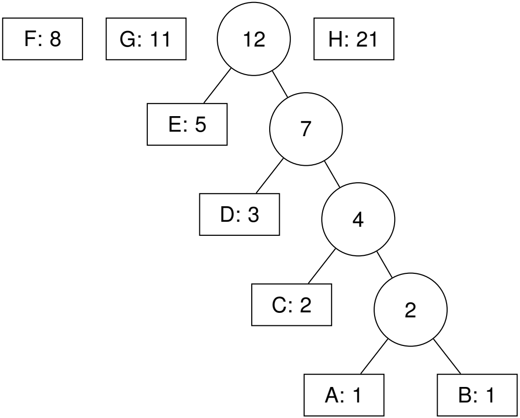


Codice Huffman

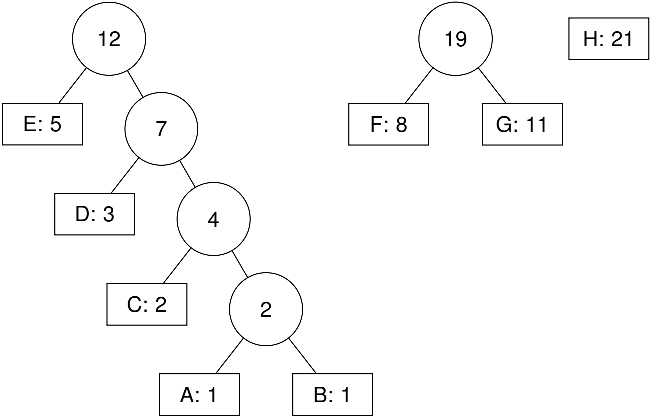
Iterando, si ottiene:



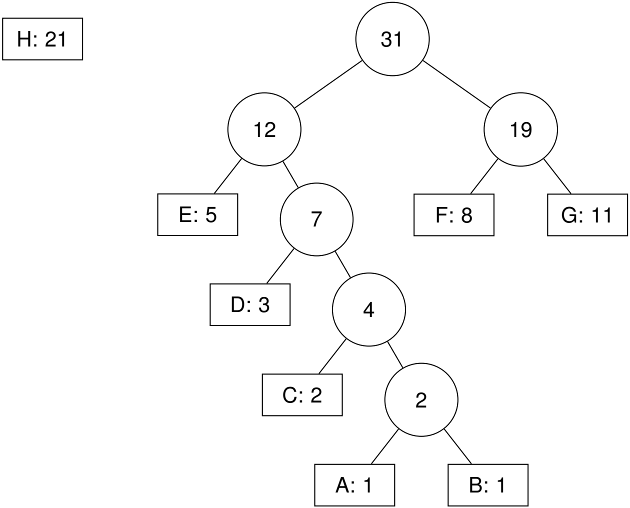
Codice Huffman



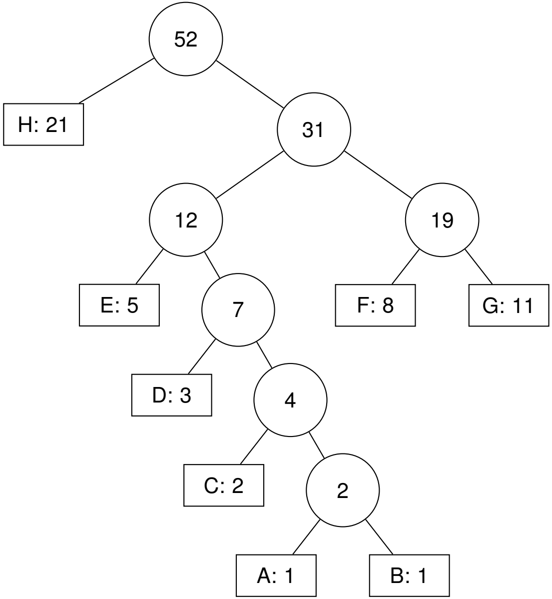
Codice Huffman

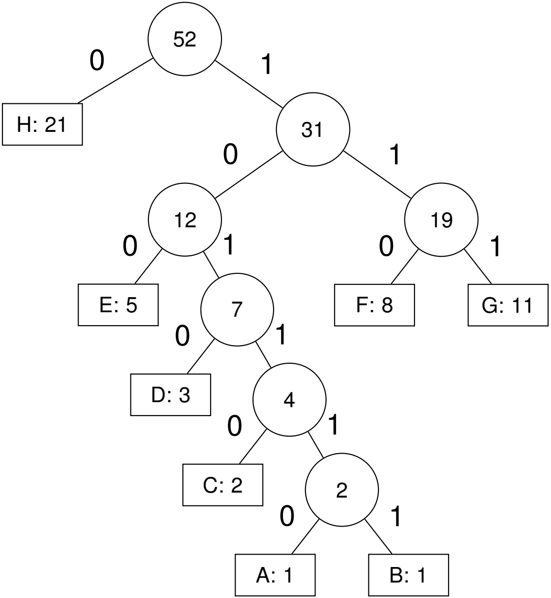


Codice Huffman



Codice Huffman

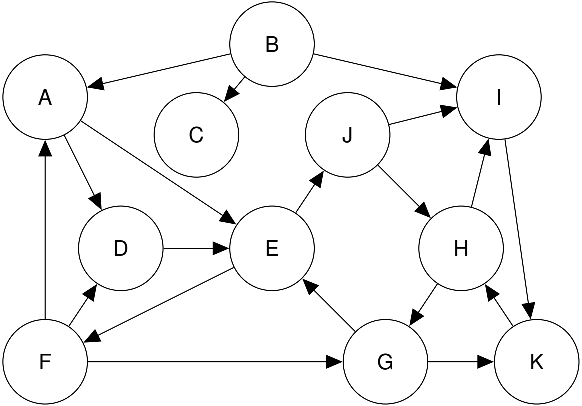




A	101110
B	101111
C	10110
D	1010
E	100
F	110
G	111
H	0

Visite grafi

Consideriamo il seguente grafo



Considerando B come nodo iniziale:

- Si effettui una visita in ampiezza
- Si effettui una visita in profondità, considerando i vertici in ordine alfabetico e etichettando i vertici con tempo di inizio/fine elaborazione
- Si etichettino gli archi come T (tree), B (back), F (forward), C (cross)

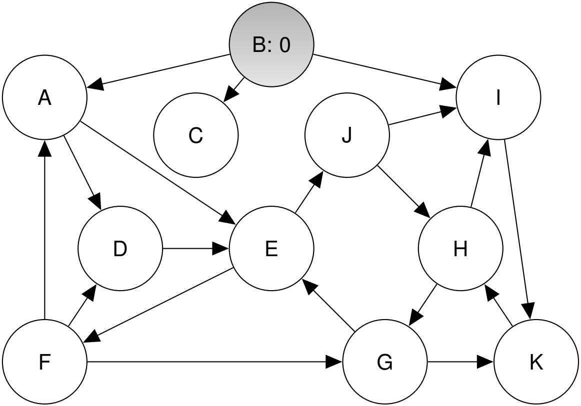
Visita in ampiezza

Usiamo una coda per contenere i nodi scoperti e non ancora visitati

Partiamo dal nodo B, lo inseriamo nella coda e lo etichettiamo come nodo a distanza 0 da B stesso

Proseguiamo estraendo dalla coda il primo nodo X, andando quindi a inserire utti i nodi adiacenti a X non ancora scoperti, che saranno etichettati come a distanza $D + 1$, dove D è la distanza di X da B

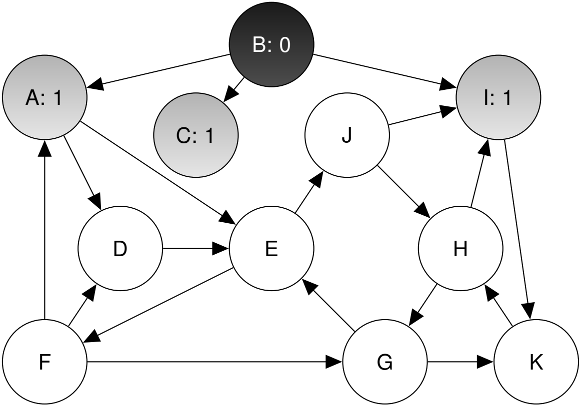
Visita in ampiezza



Coda:

B: 0

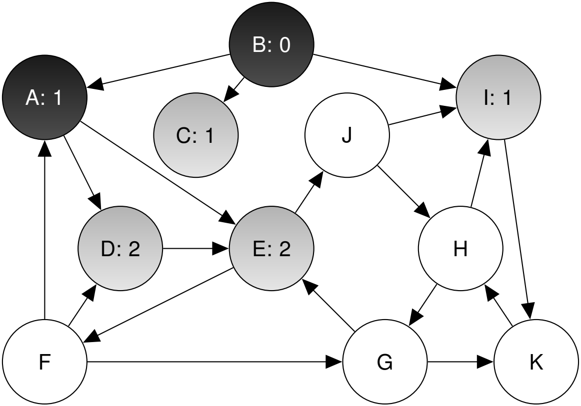
Visita in ampiezza



Coda:

A: 1
C: 1
I: 1

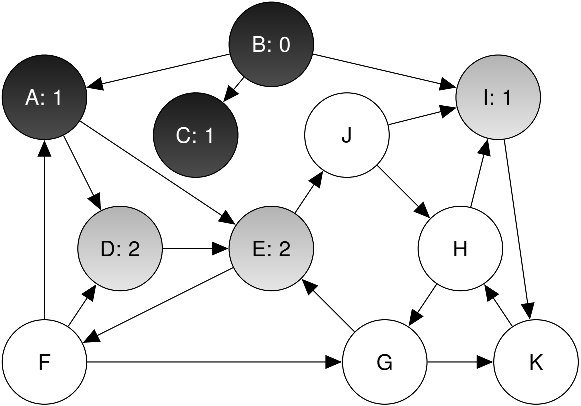
Visita in ampiezza



Coda:

C: 1
I: 1
D: 2
E: 2

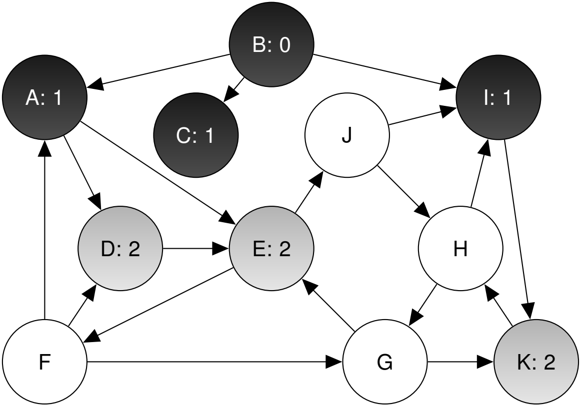
Visita in ampiezza



Coda:

I: 1
D: 2
E: 2

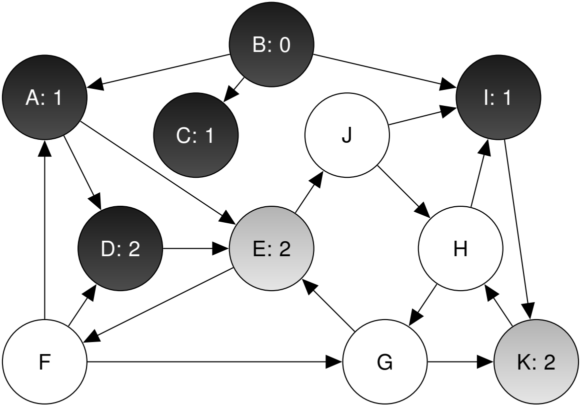
Visita in ampiezza



Coda:

D: 2
E: 2
K: 2

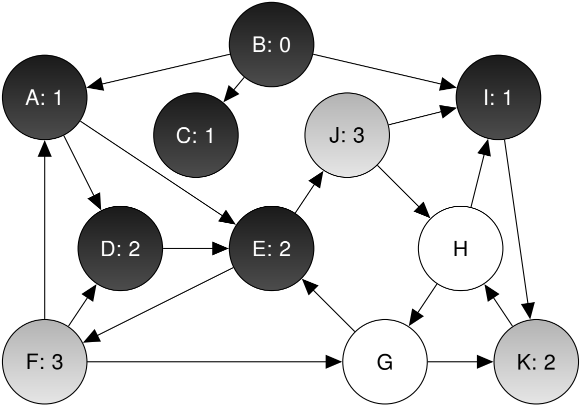
Visita in ampiezza



Coda:

E: 2
K: 2

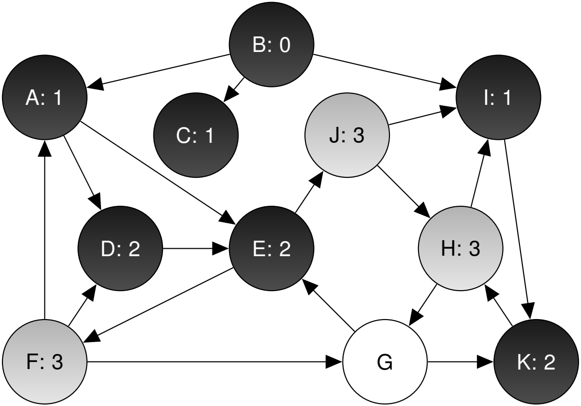
Visita in ampiezza



Coda:

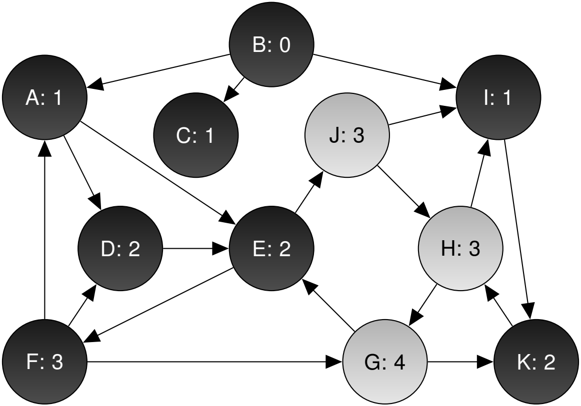
K: 2
F: 3
J: 3

Visita in ampiezza



Coda:

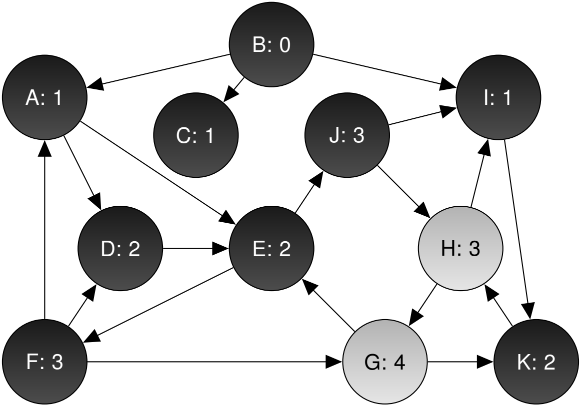
F: 3
J: 3
H: 3



Coda:

J: 3
H: 3
G: 4

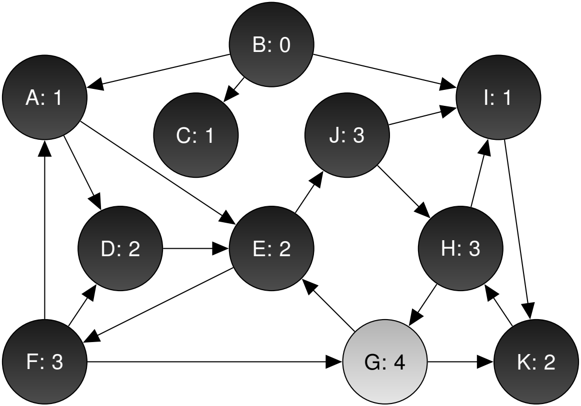
Visita in ampiezza



Coda:

H: 3
G: 4

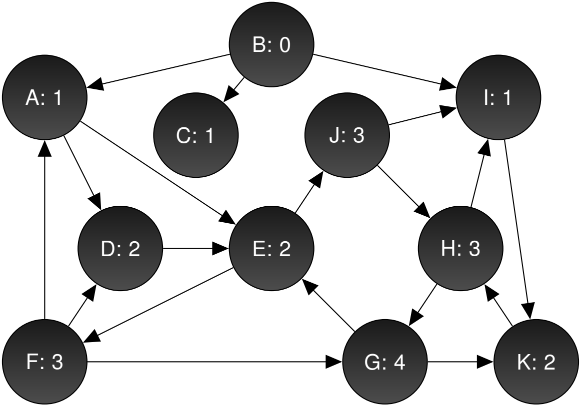
Visita in ampiezza



Coda:

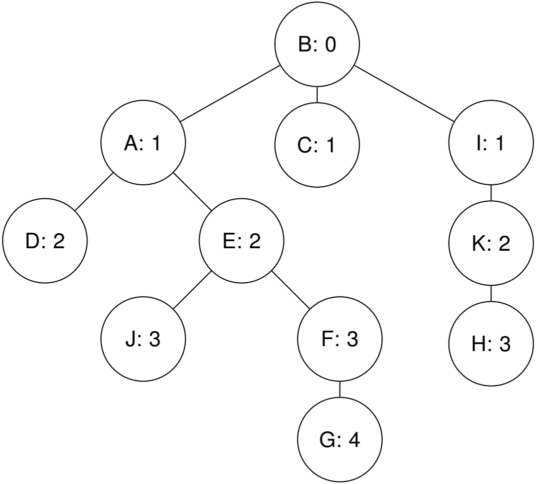
G: 4

Visita in ampiezza



Coda:

Albero della visita in ampiezza:



Visita in profondità

Partiamo dal nodo B e iniziamo a percorrere cammini finchè possibile senza ripassare da nodi già visitati.

Quando ciò non è possibile, torniamo indietro fino al primo nodo da cui partono ancora cammini inesplorati e proseguiamo la visita

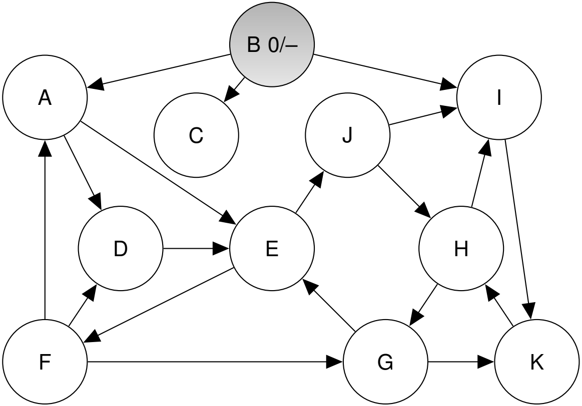
Etichettiamo i nodi con tempo di inizio e fine elaborazione

La visita genera un albero (o una foresta)

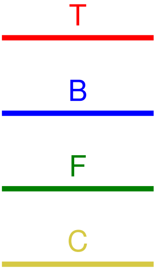
Etichettiamo gli archi che percorriamo come archi T (tree, in rosso nelle figure)

Per gli altri archi, distinguiamo tra archi F (forward, in verde), archi B (back, in blu) e archi C (cross, in giallo)

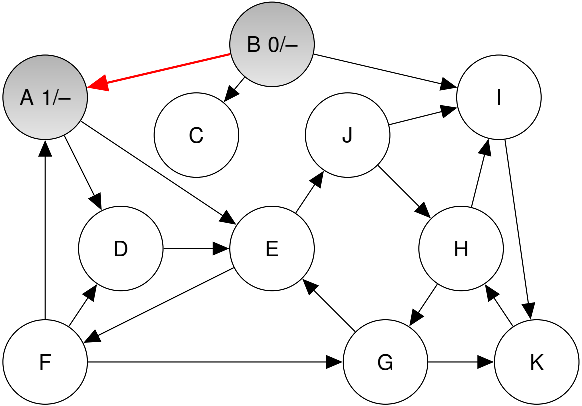
Visita in profondità



Archi:



Visita in profondità



Archi:

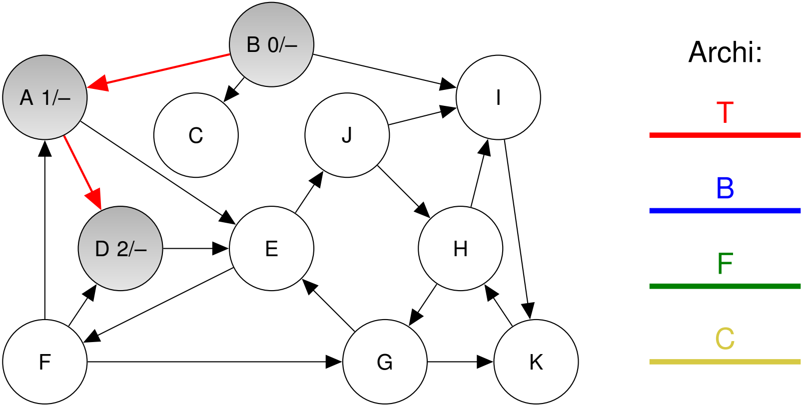
T

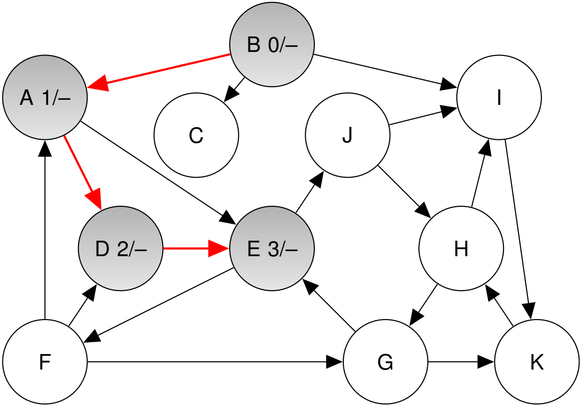
B

F

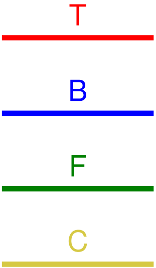
C

Visita in profondità

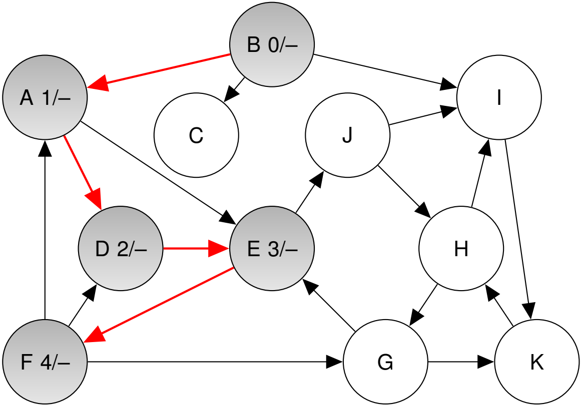




Archi:



Visita in profondità



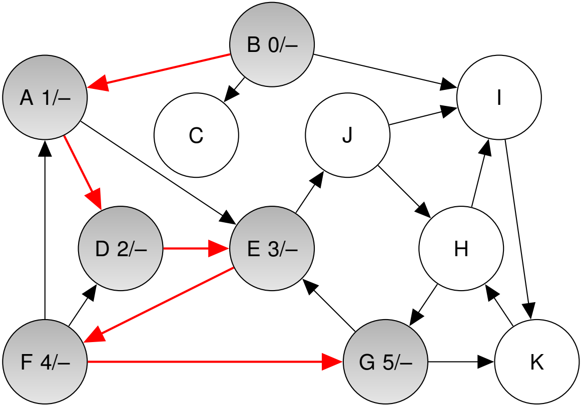
Archi:

T

B

F

C



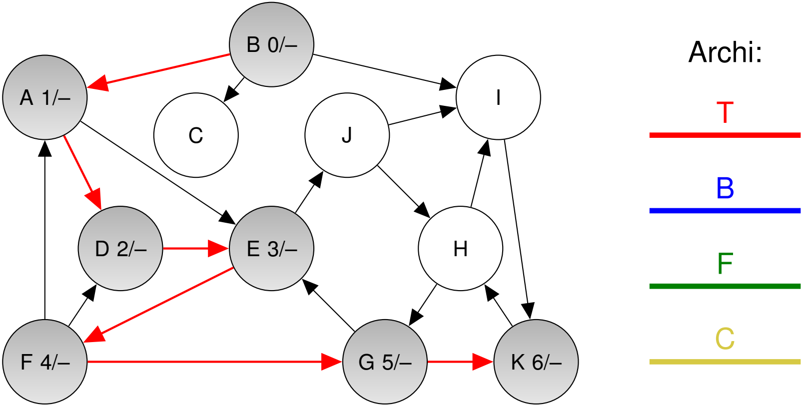
Archi:

T

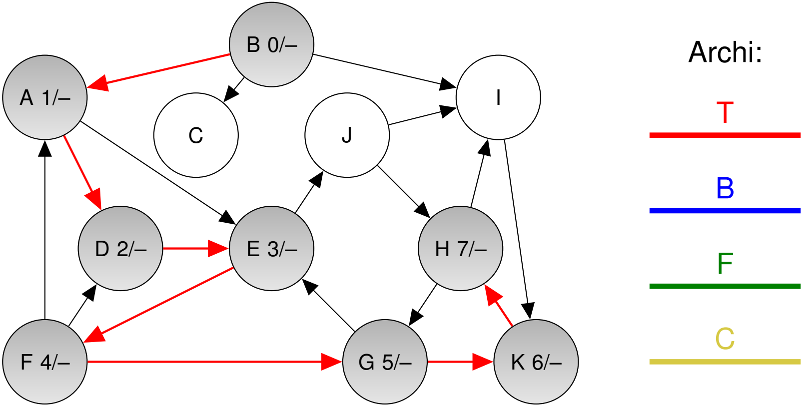
B

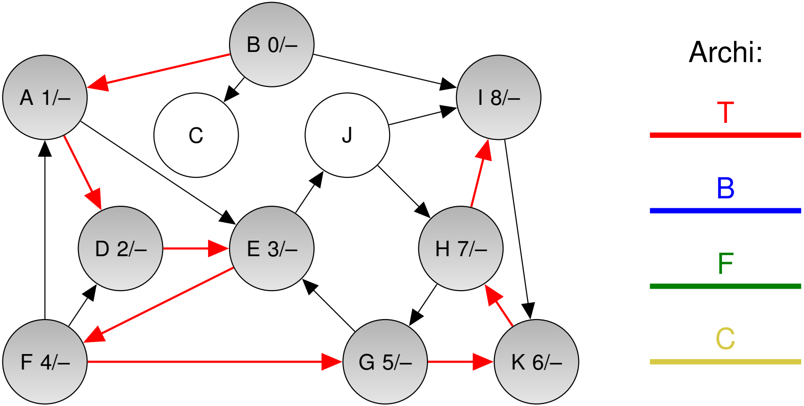
F

C

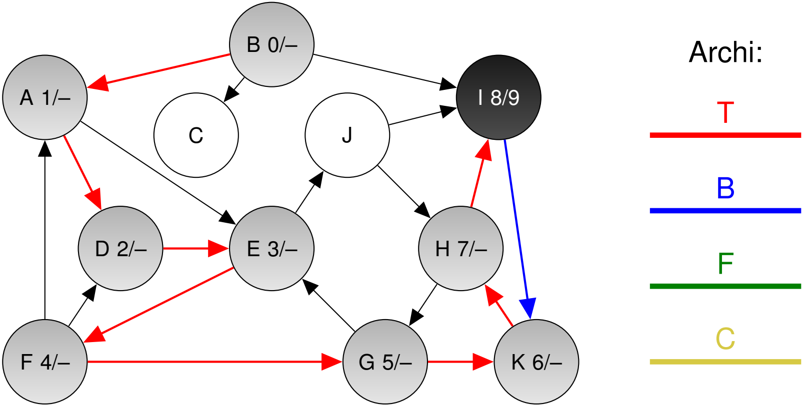

B 0/–
A 1/–
C
J
I
D 2/–
E 3/–
H
F 4/–
G 5/–
K 6/–
Archi:
T
B
F
C

Visita in profondità

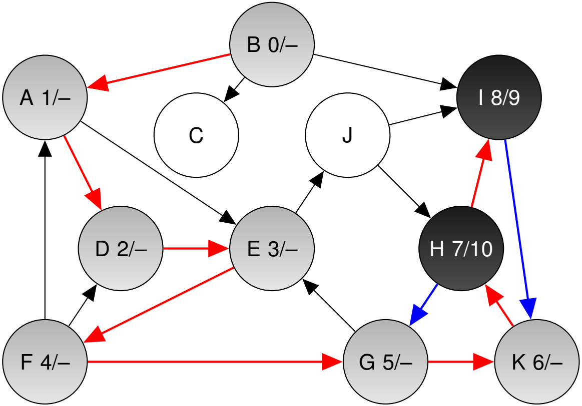



B 0/–
A 1/–
C
J
I 8/–
D 2/–
E 3/–
H 7/–
F 4/–
G 5/–
K 6/–
Archi:
T
B
F
C

Visita in profondità



Visita in profondità



Archi:

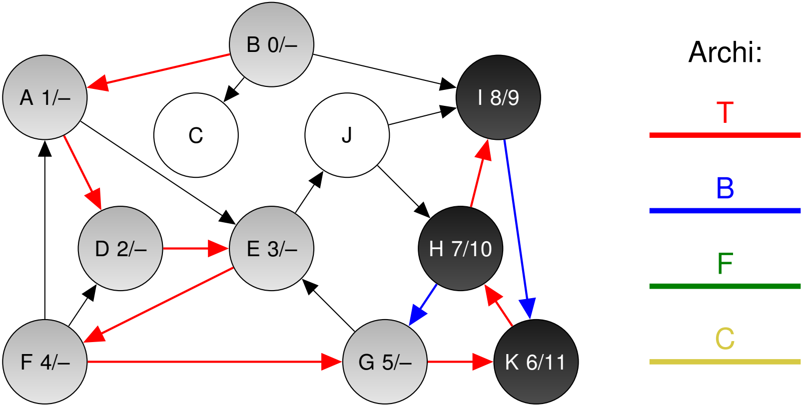
T

B

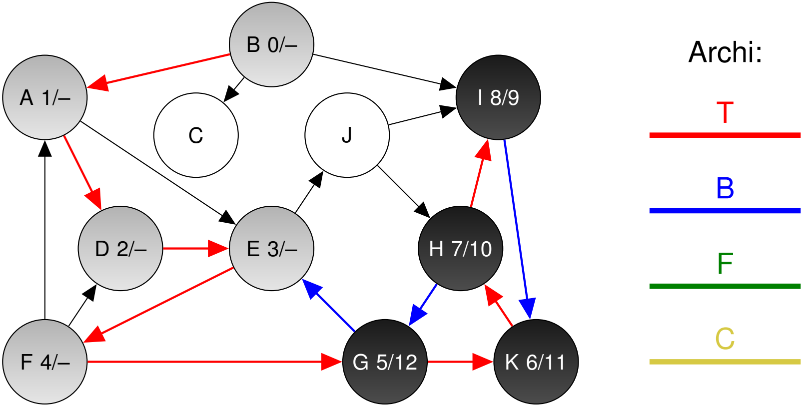
F

C

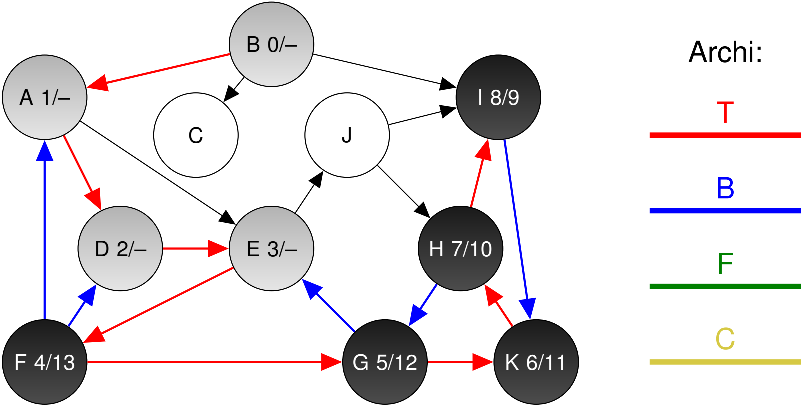
Visita in profondità



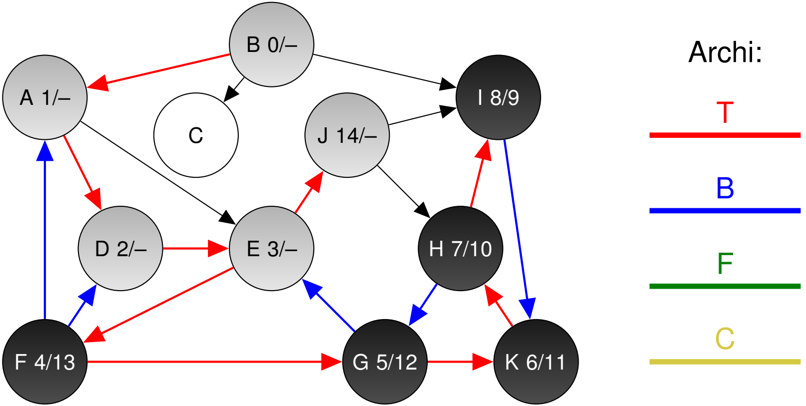
Visita in profondità



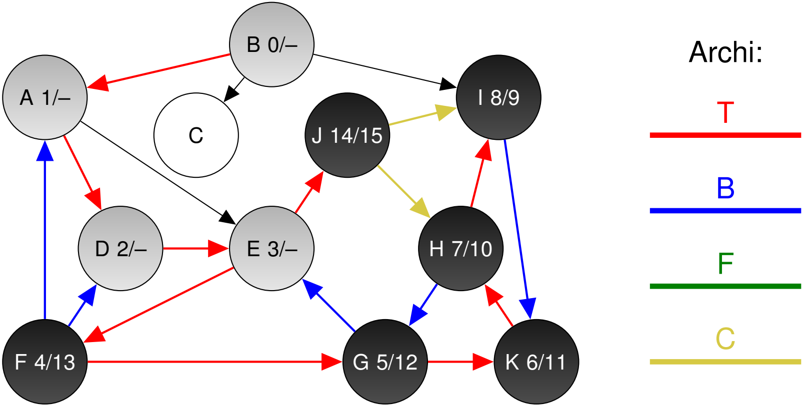
Visita in profondità

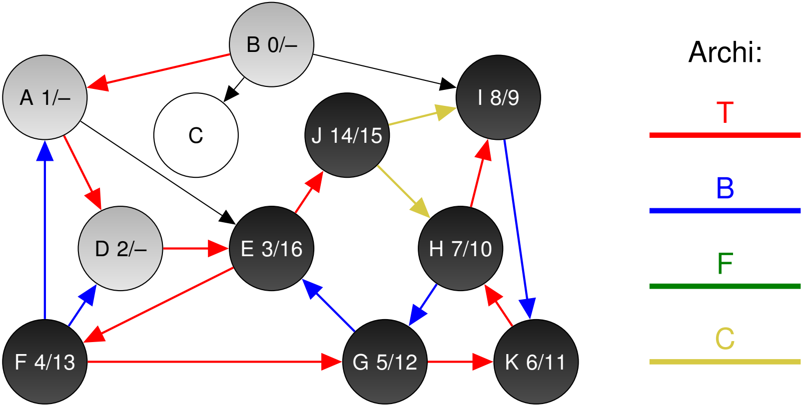


Visita in profondità

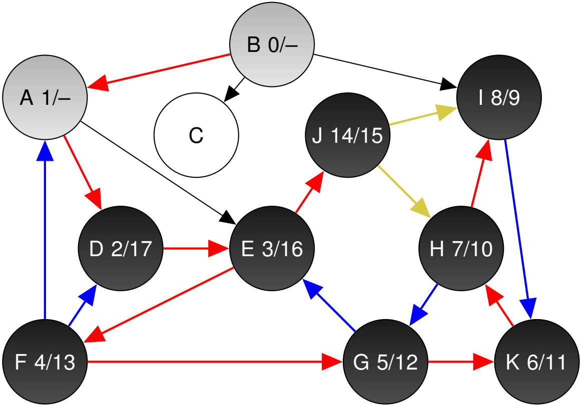


Visita in profondità




B 0/–
A 1/–
C
J 14/15
I 8/9
D 2/–
E 3/16
H 7/10
F 4/13
G 5/12
K 6/11
Archi:
T
B
F
C

Visita in profondità



Archi:

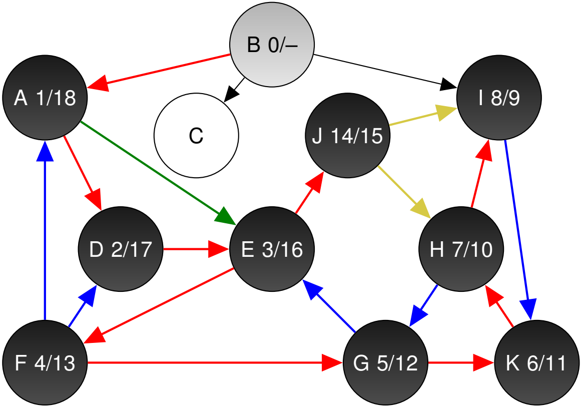
T

B

F

C

Visita in profondità



Archi:

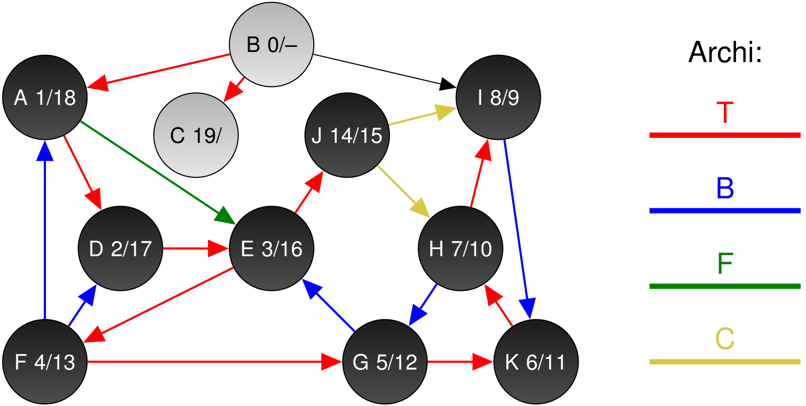
T

B

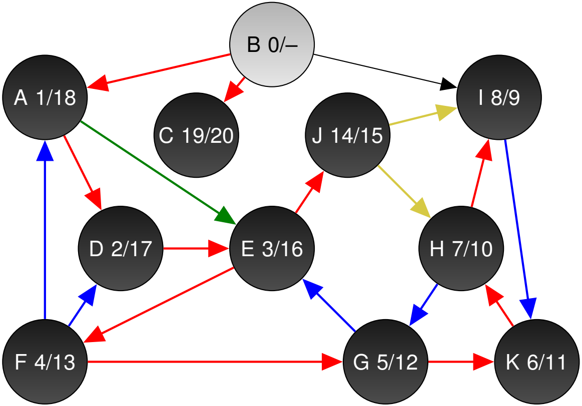
F

C

Visita in profondità



Visita in profondità



Archi:

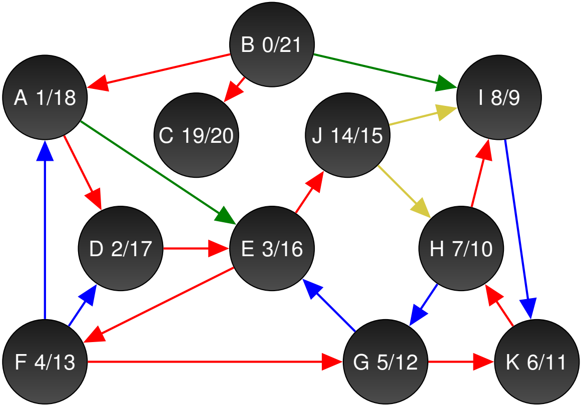
T

B

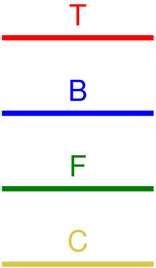
F

C

Visita in profondità

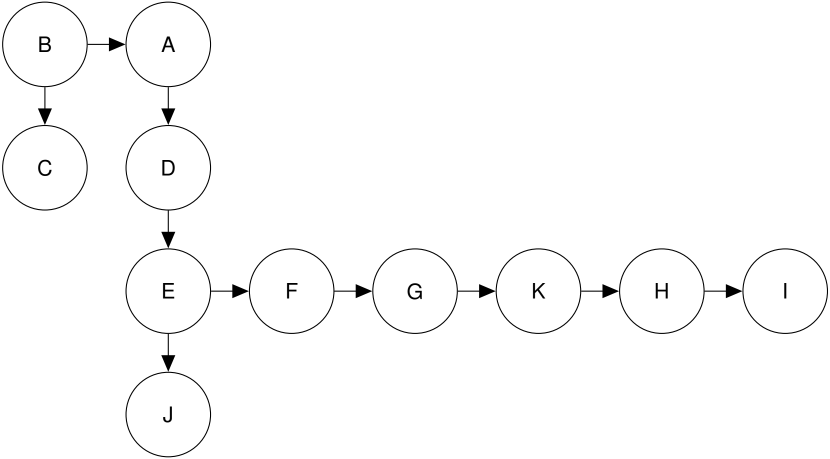


Archi:



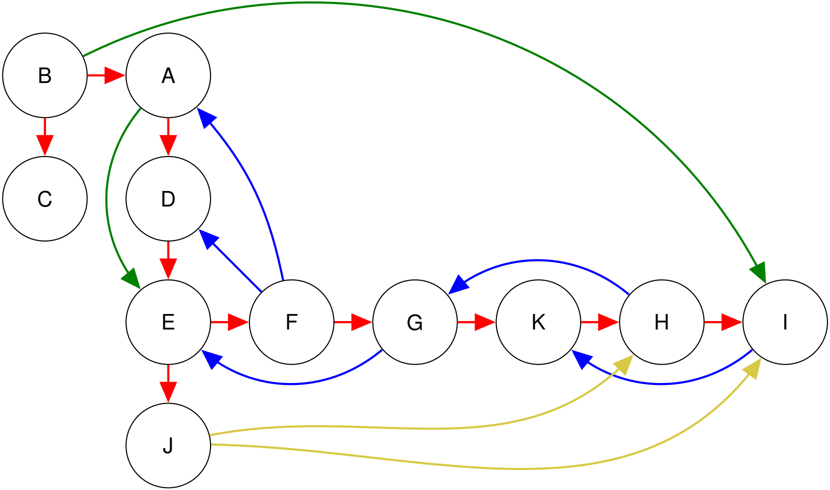
Visita in profondità

Albero della visita in profondità:

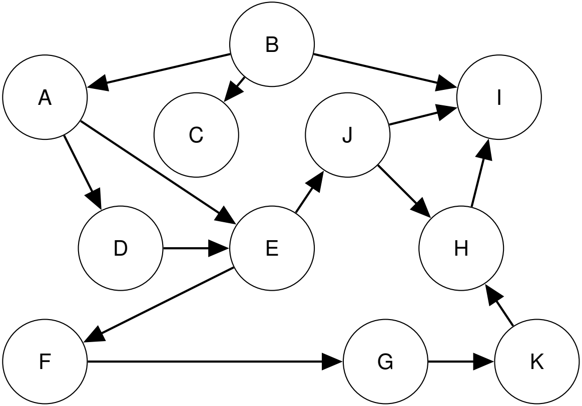


Visita in profondità

Completando il grafo:



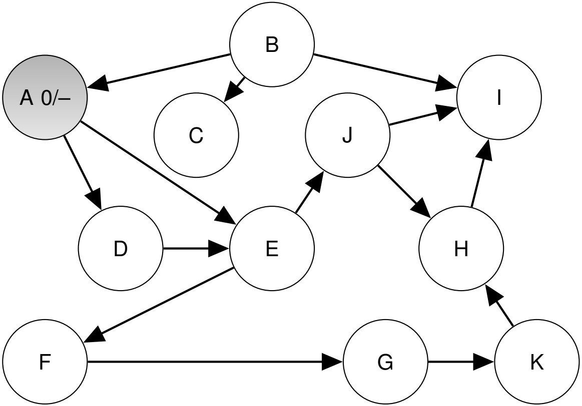
Consideriamo il grafo orientato aciclico



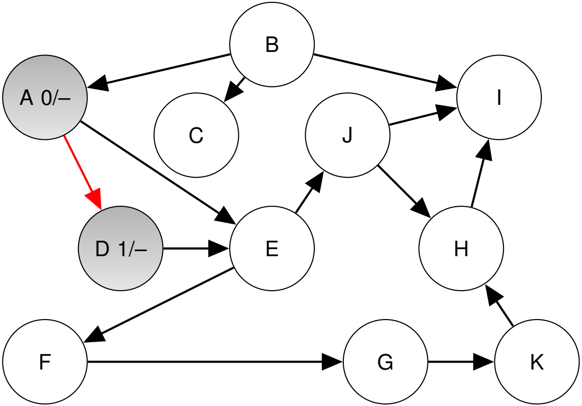
Per trovare un ordinamento topologico utilizziamo una visita DFS

Consideriamo i nodi ordinati lessicograficamente

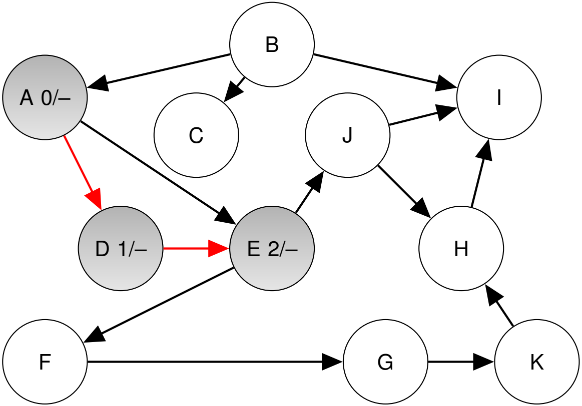
DAG - Ordinamento topologico



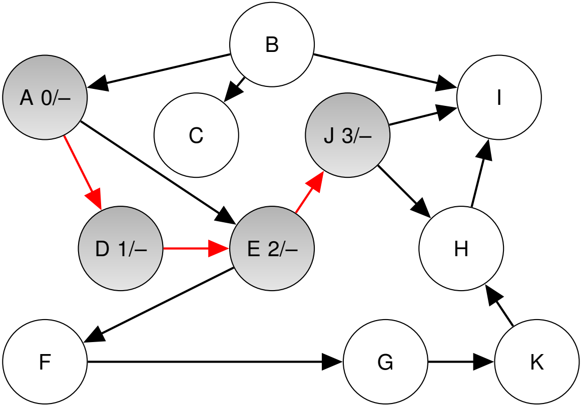
DAG - Ordinamento topologico

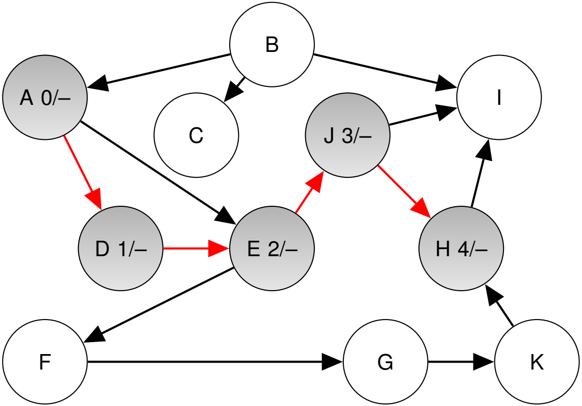


DAG - Ordinamento topologico

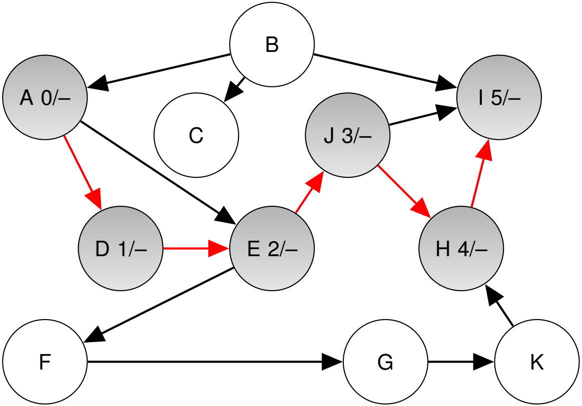


DAG - Ordinamento topologico

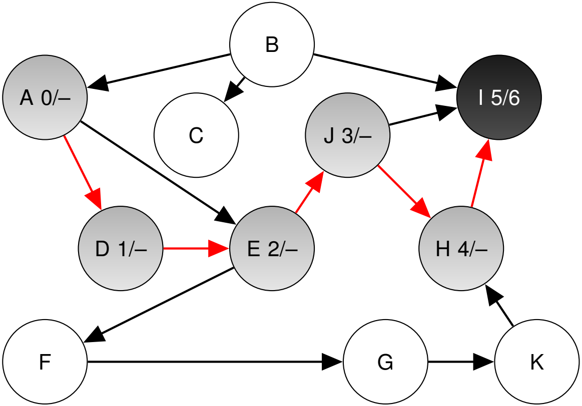


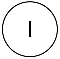

B
A 0/–
C
J 3/–
I
D 1/–
E 2/–
H 4/–
F
G
K

DAG - Ordinamento topologico

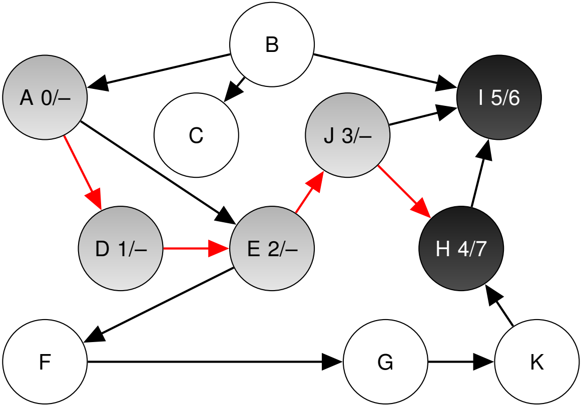


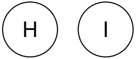
DAG - Ordinamento topologico



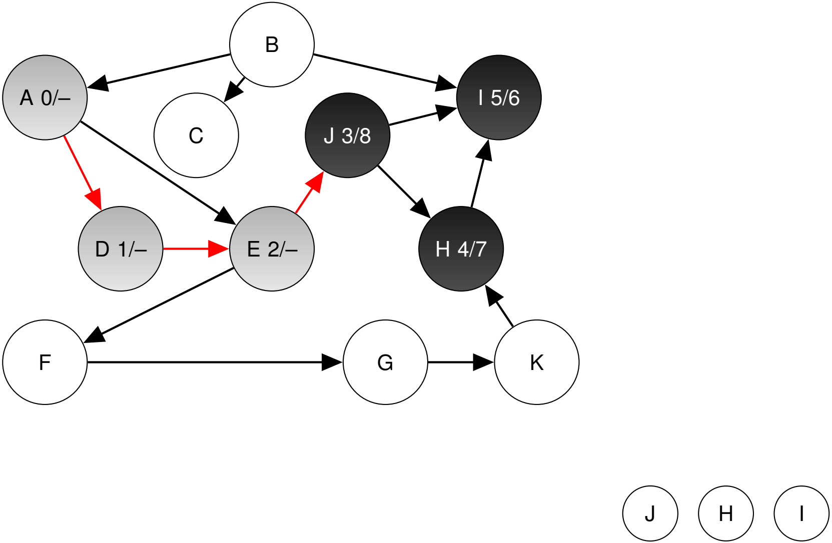


DAG - Ordinamento topologico

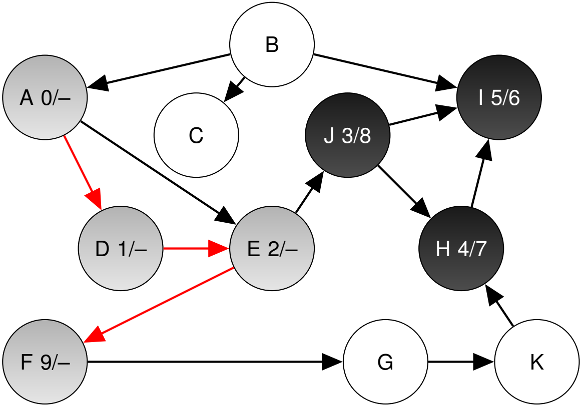


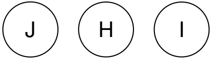


DAG - Ordinamento topologico

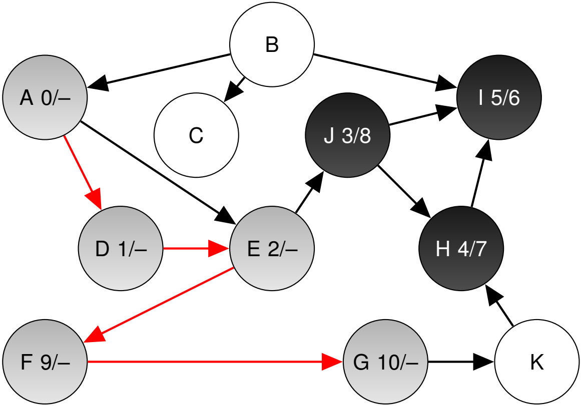


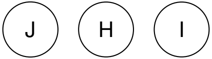
DAG - Ordinamento topologico



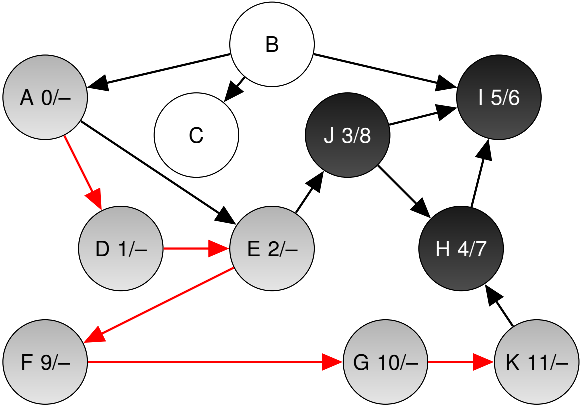


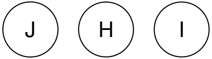
DAG - Ordinamento topologico



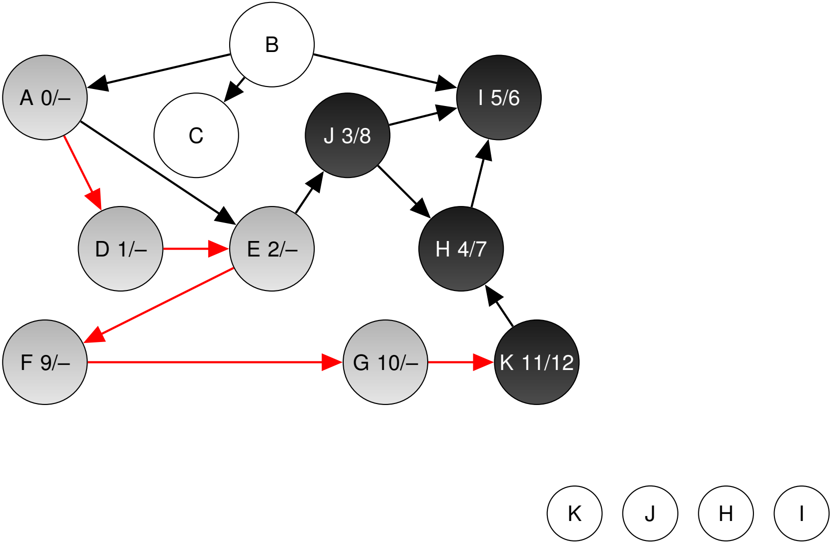


DAG - Ordinamento topologico

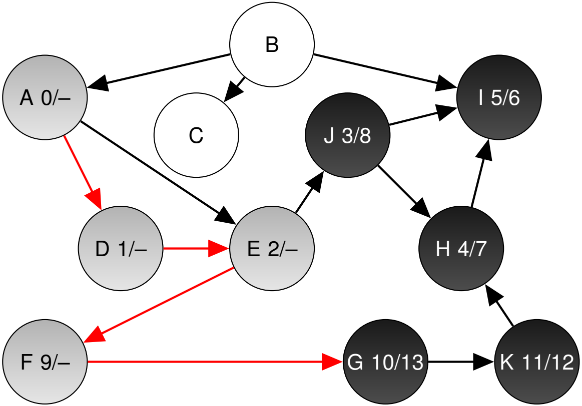


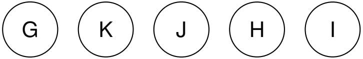


DAG - Ordinamento topologico

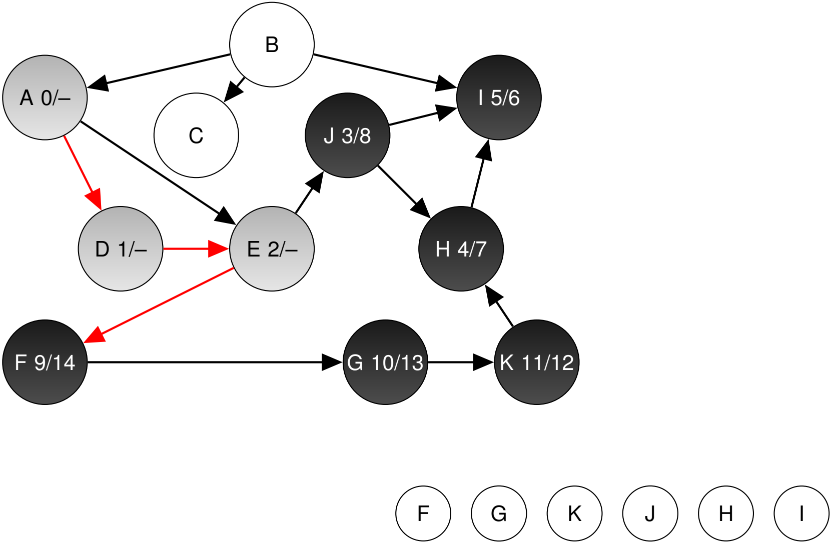


DAG - Ordinamento topologico

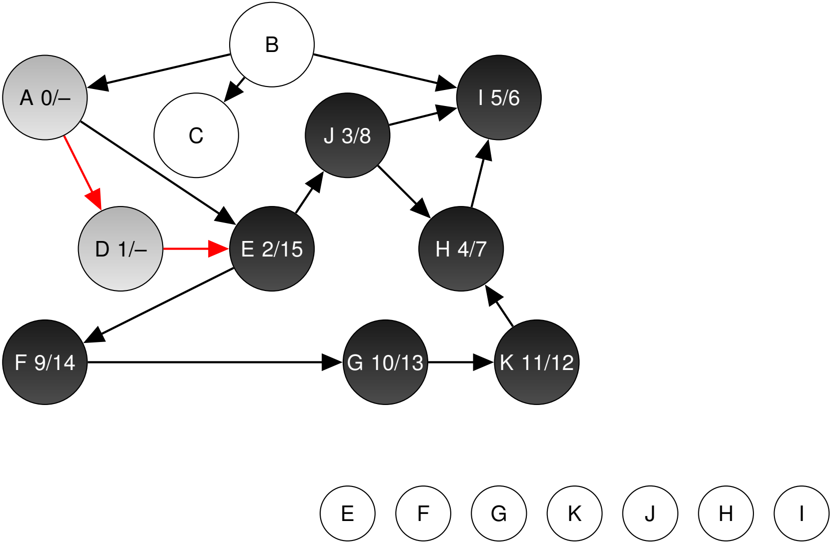




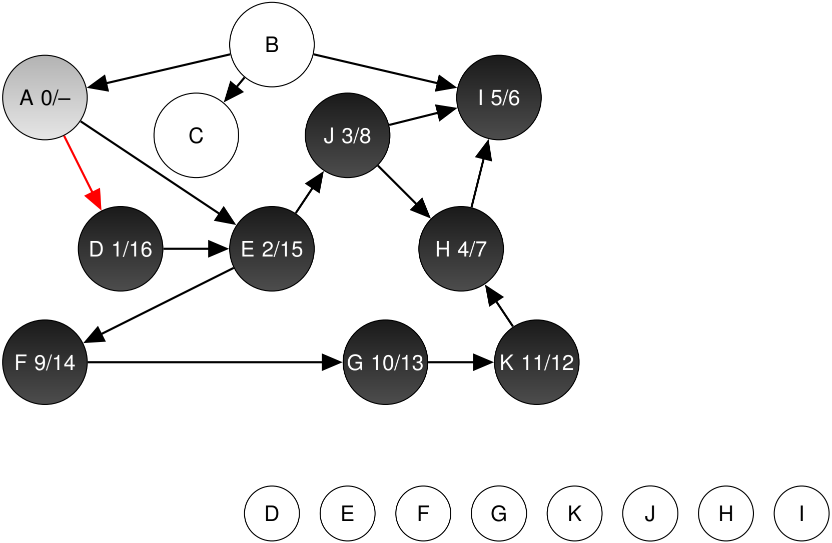
DAG - Ordinamento topologico



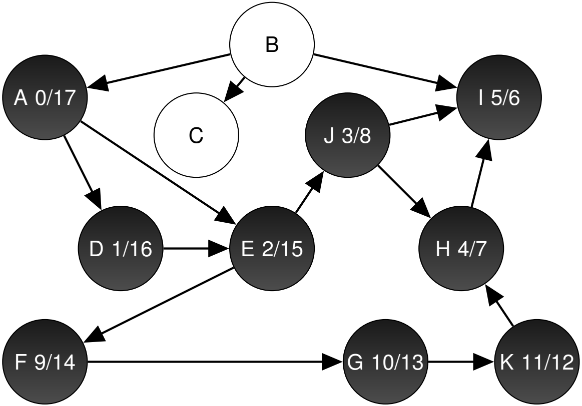
DAG - Ordinamento topologico

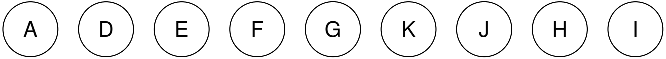


DAG - Ordinamento topologico

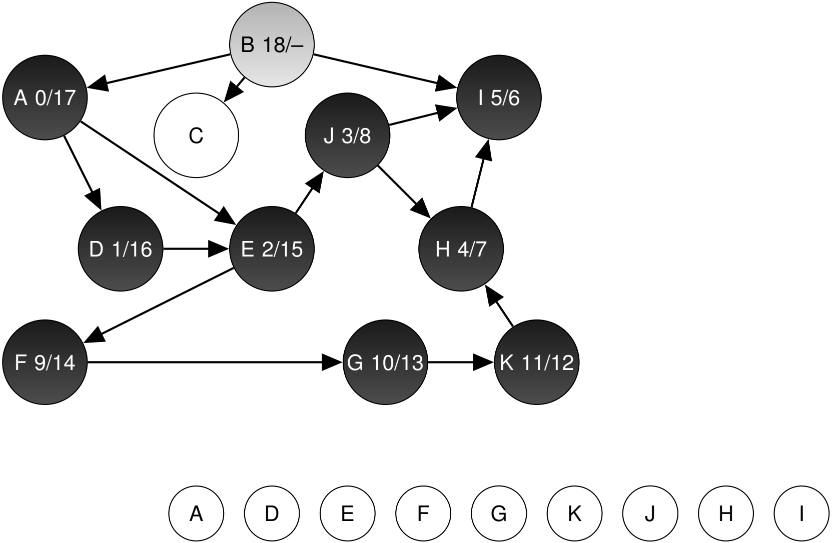


DAG - Ordinamento topologico

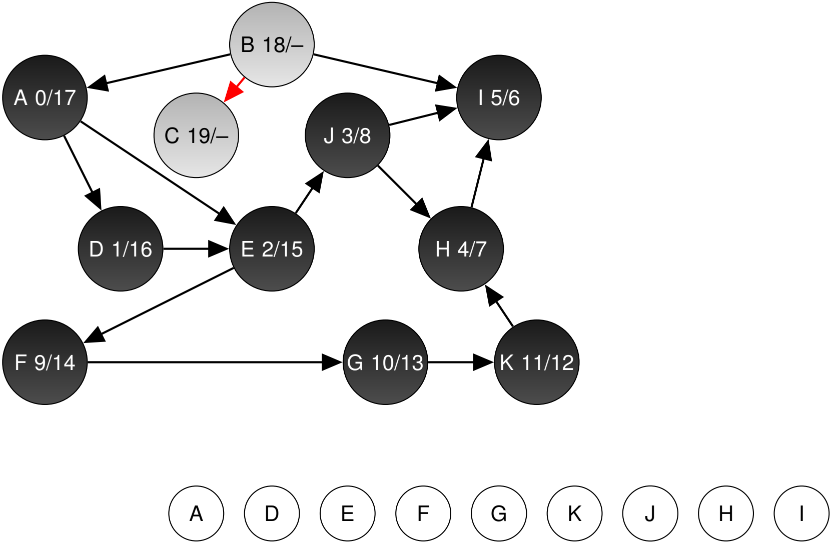


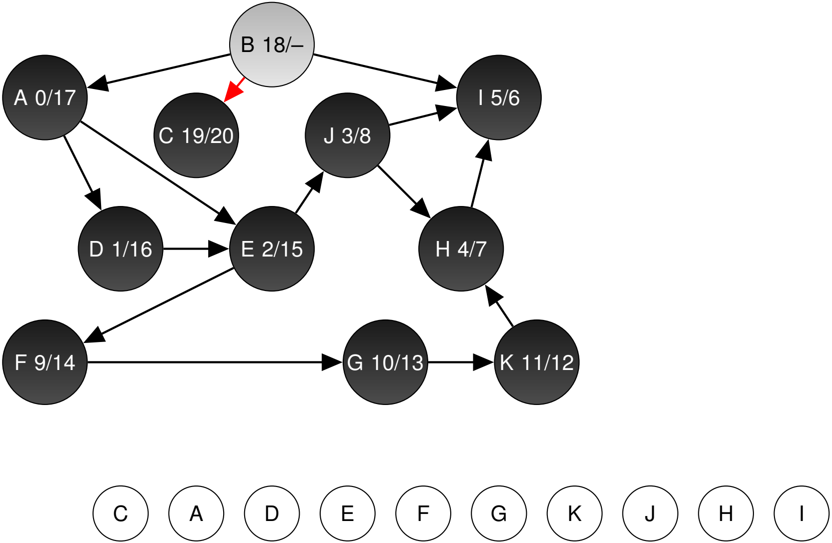


DAG - Ordinamento topologico

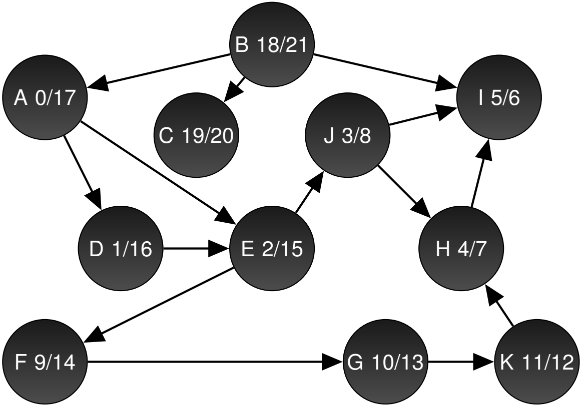


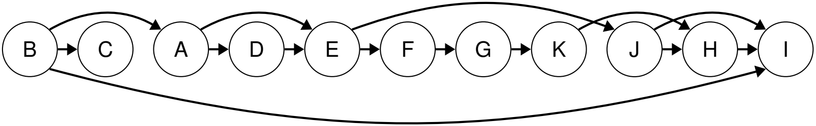
DAG - Ordinamento topologico




B 18/–
A 0/17
C 19/20
J 3/8
I 5/6
D 1/16
E 2/15
H 4/7
F 9/14
G 10/13
K 11/12
C
A
D
E
F
G
K
J
H
I

DAG - Ordinamento topologico

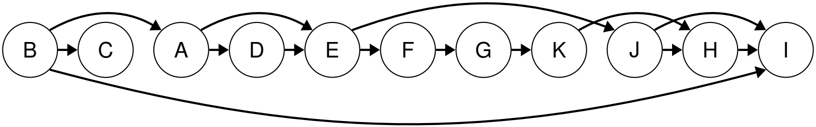




DAG - Cammini massimi

Vogliamo calcolare un cammino massimo

Partiamo dall'ordinamento topologico calcolato in precedenza:

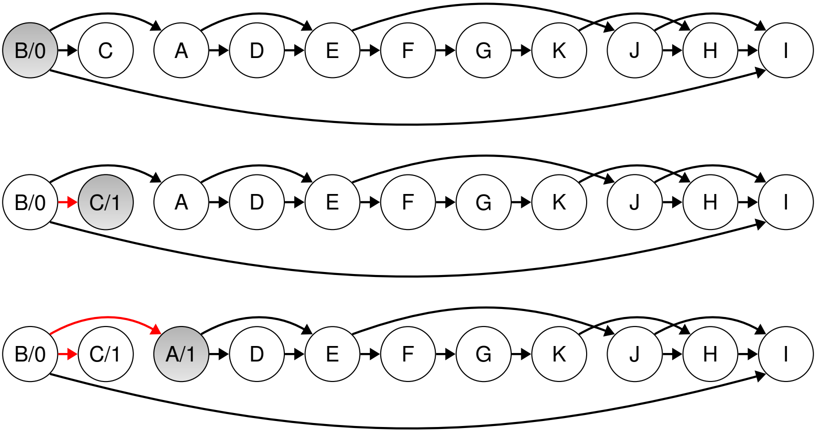


Percorriamo l'ordinamento topologico, calcolando, per ogni nodo:

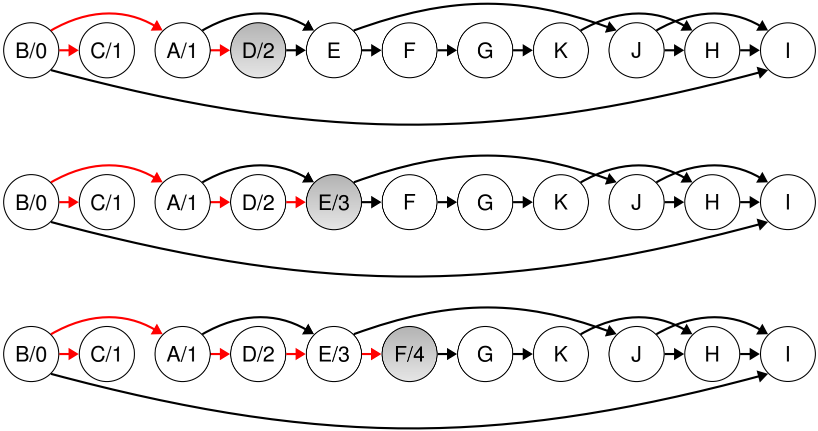
$$d(v) = \max_{u|(u,v)\in E} (d(u) + 1) \tag{1}$$

Se il nodo v non ha predecessori, mettiamo $d(v) = 0$

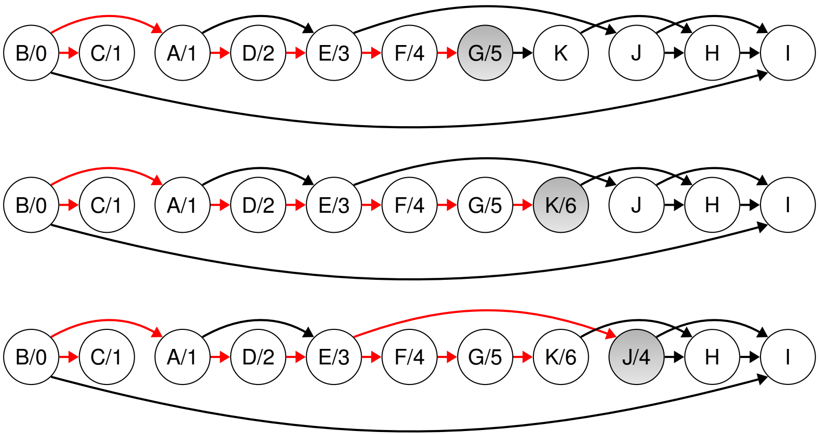
DAG - Cammini massimi



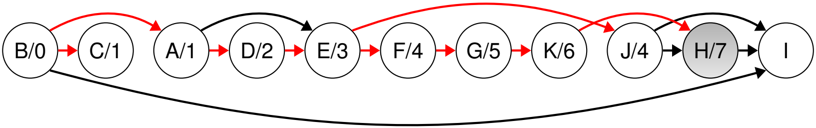
DAG - Cammini massimi

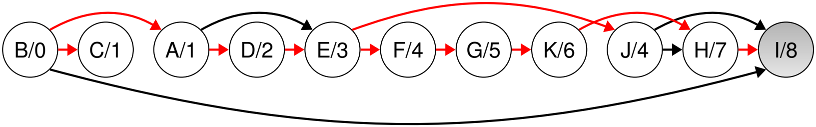


DAG - Cammini massimi

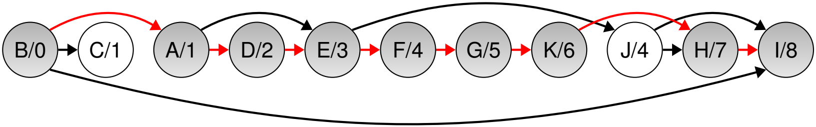


DAG - Cammini massimi





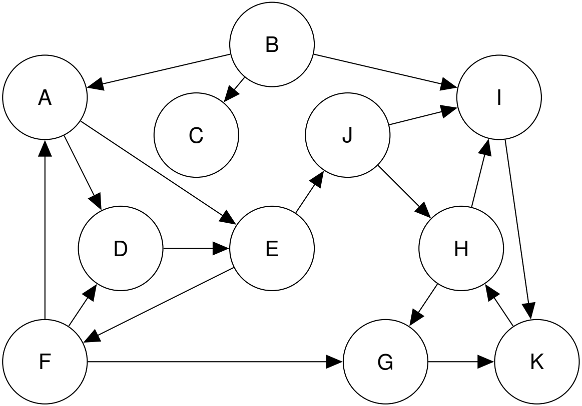
Cammino di lunghezza massima: nodo finale I ($\arg\max_v d(v)$)



Percorrendo all'indietro gli archi che portano a I:

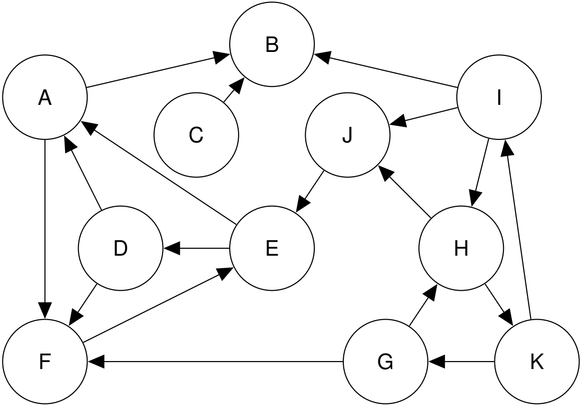
B -> A -> D -> E -> F -> G -> K -> H -> I

Consideriamo il seguente grafo (nota — il grafo è leggermente diverso da quello considerato nella prima parte):



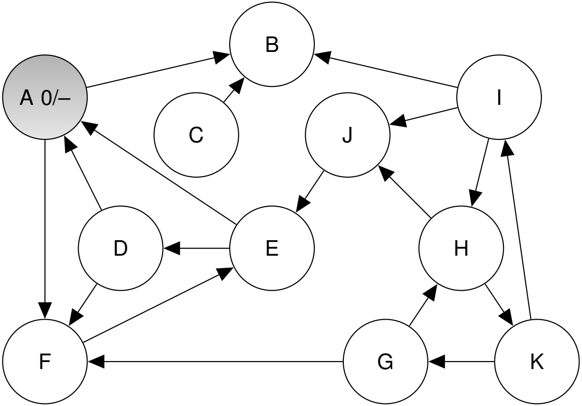
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

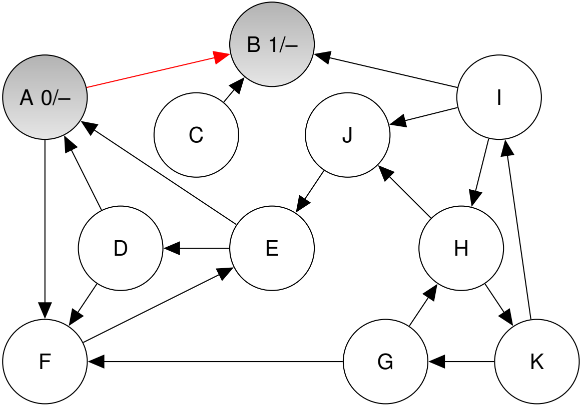


Componenti fortemente connesse

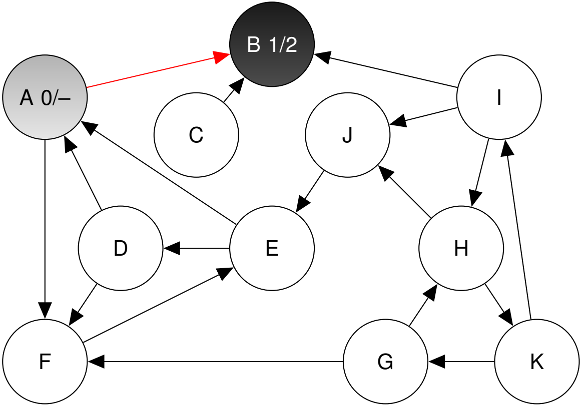
Primo passo: DFS sul grafo trasposto



Primo passo: DFS sul grafo trasposto

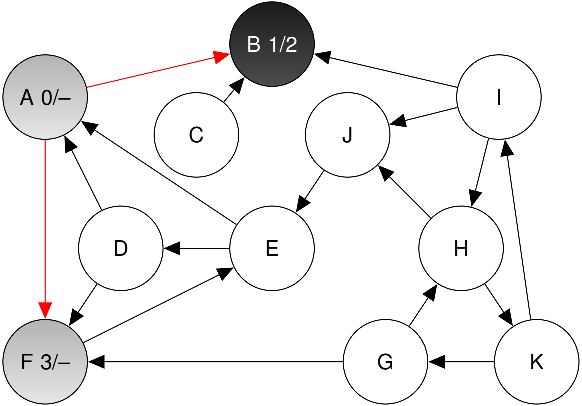


Primo passo: DFS sul grafo trasposto



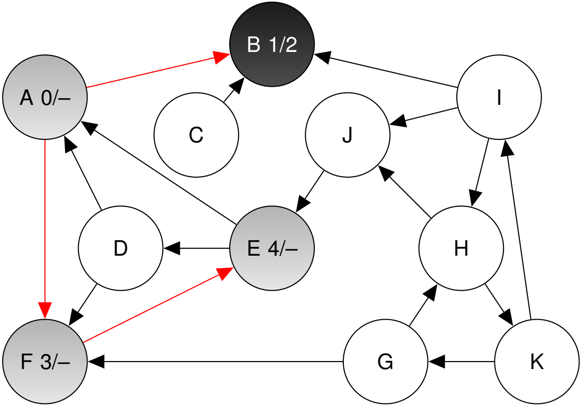
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

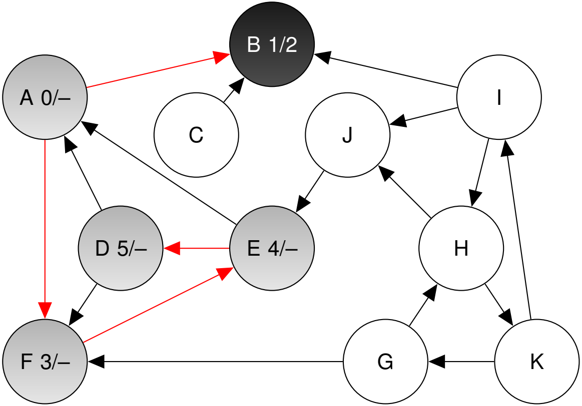


Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

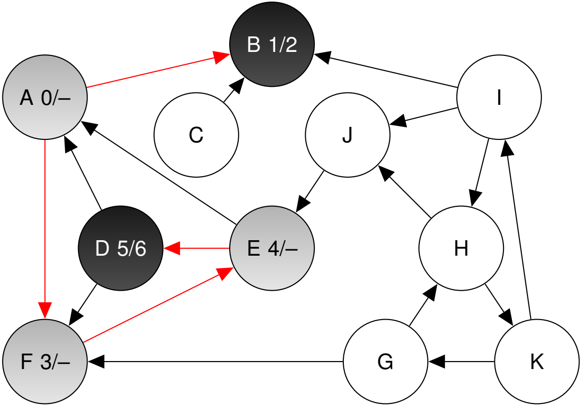


Primo passo: DFS sul grafo trasposto

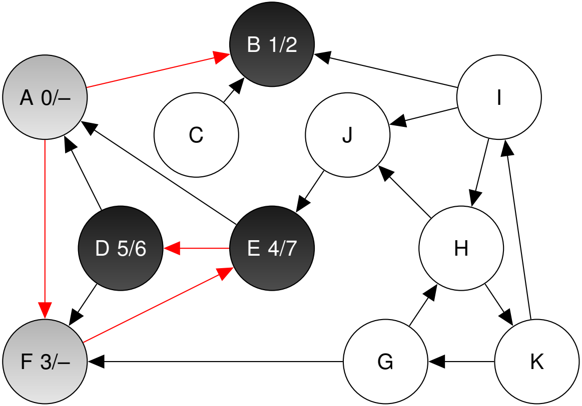


Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

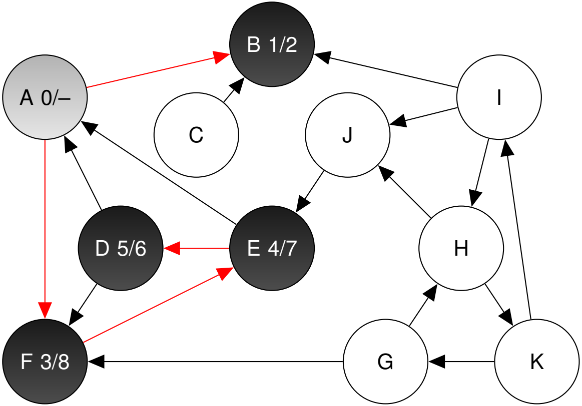


Primo passo: DFS sul grafo trasposto



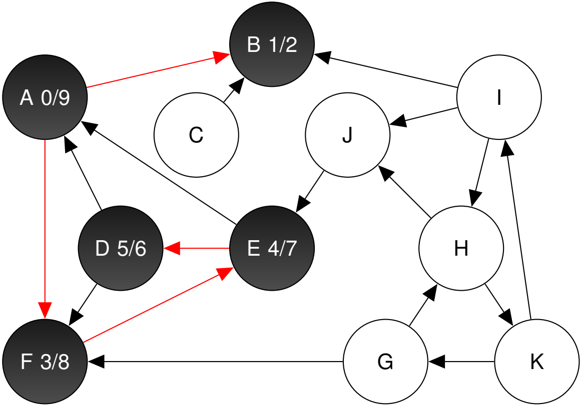
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto



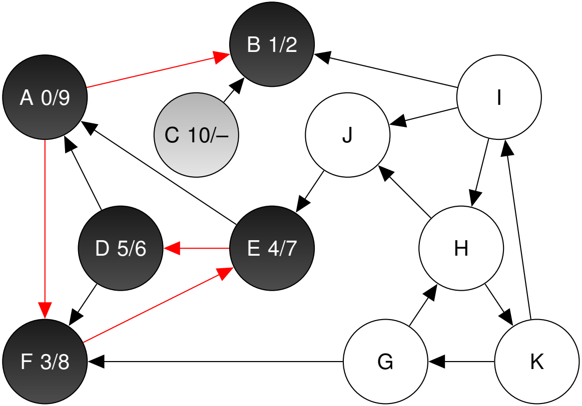
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

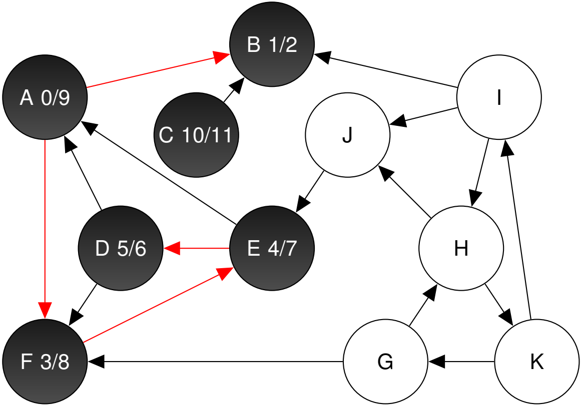


Componenti fortemente connesse

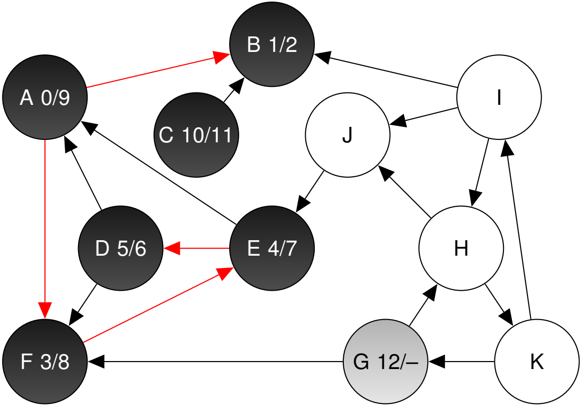
Primo passo: DFS sul grafo trasposto



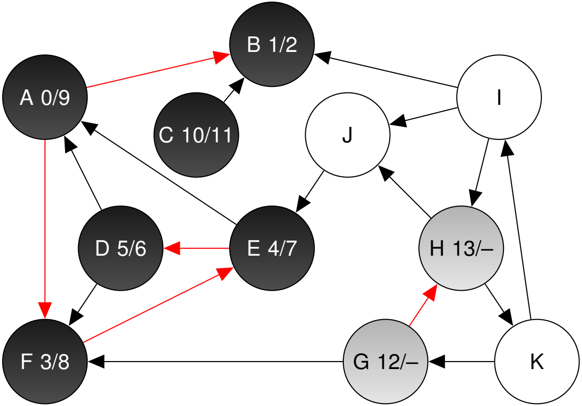
Primo passo: DFS sul grafo trasposto



Primo passo: DFS sul grafo trasposto

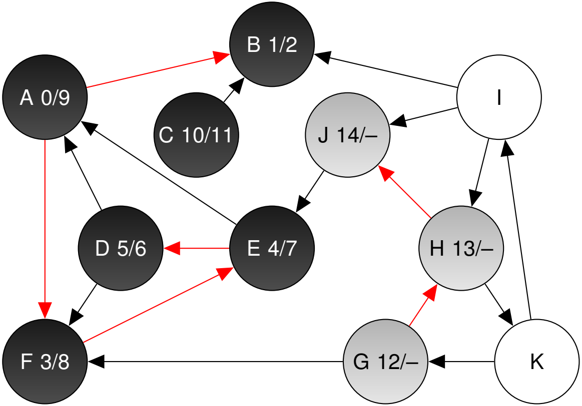


Primo passo: DFS sul grafo trasposto

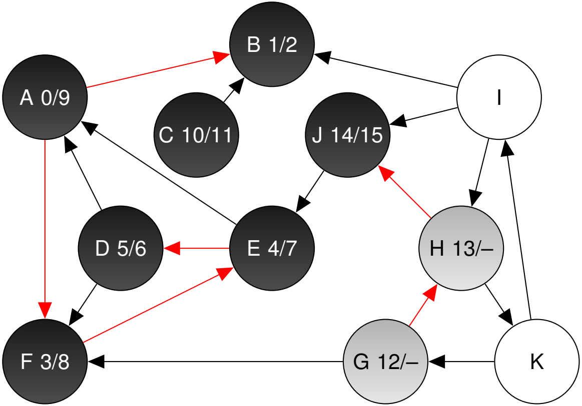


Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

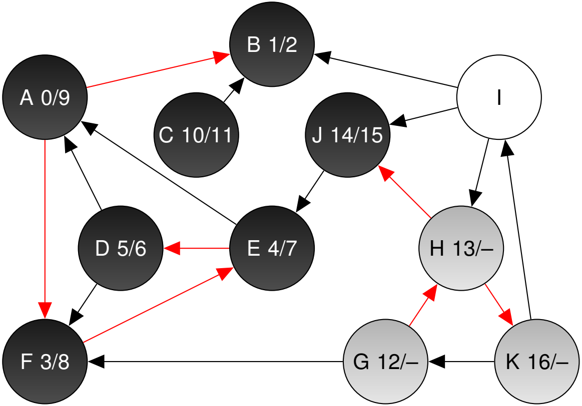


Primo passo: DFS sul grafo trasposto

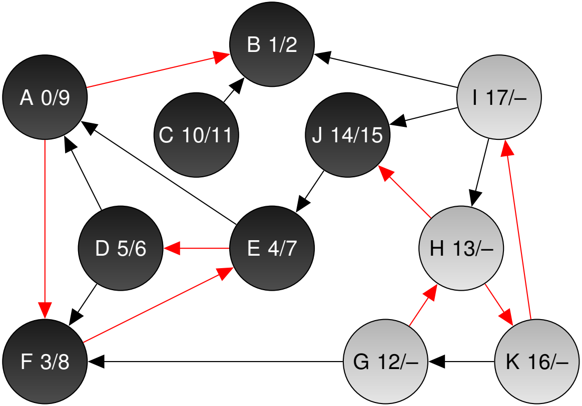


Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

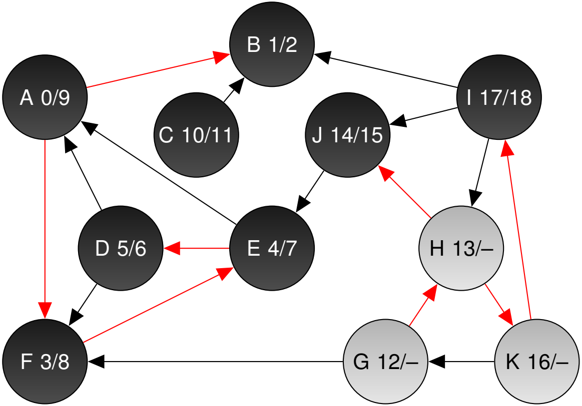


Primo passo: DFS sul grafo trasposto



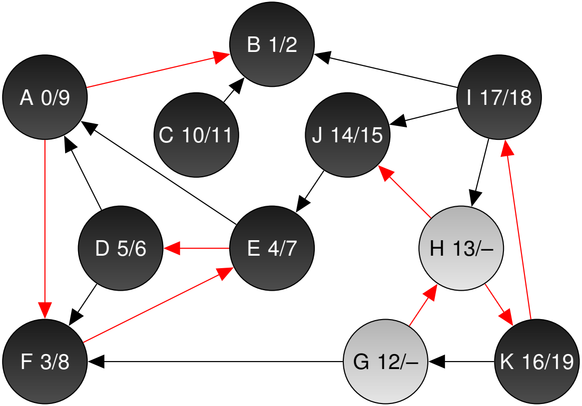
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto



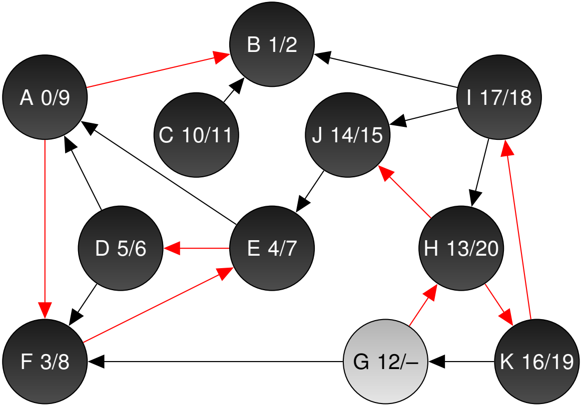
Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto

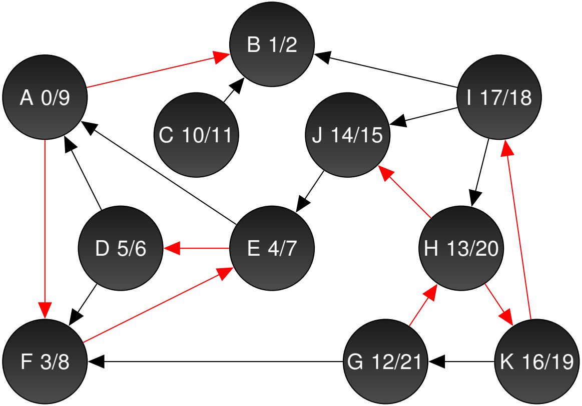


Componenti fortemente connesse

Primo passo: DFS sul grafo trasposto



Primo passo: DFS sul grafo trasposto

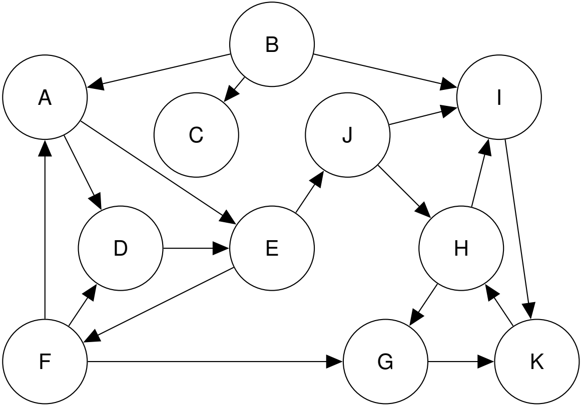


Componenti fortemente connesse

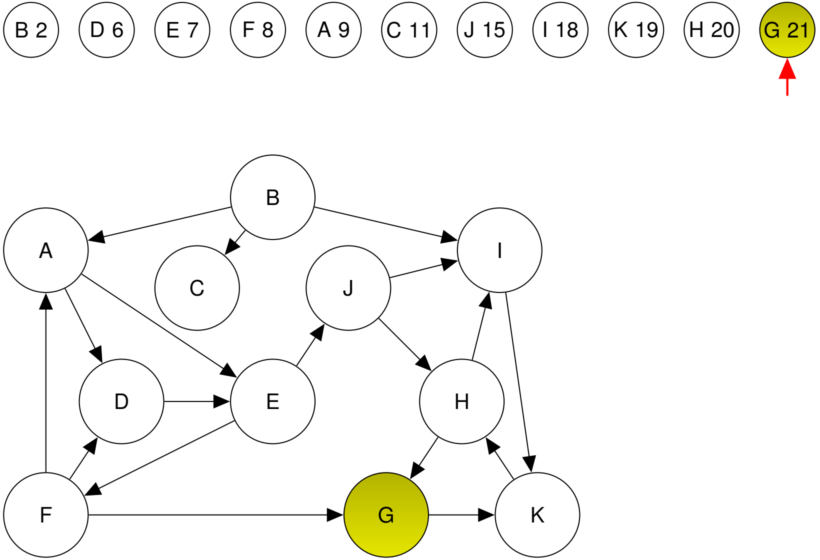
Ordinamento dei nodi per tempo di fine:

B 2 D 6 E 7 F 8 A 9 C 11 J 15 I 18 K 19 H 20 G 21

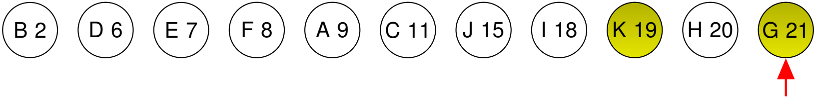
DFS su grafo originale secondo tempi di fine elaborazione decrescenti

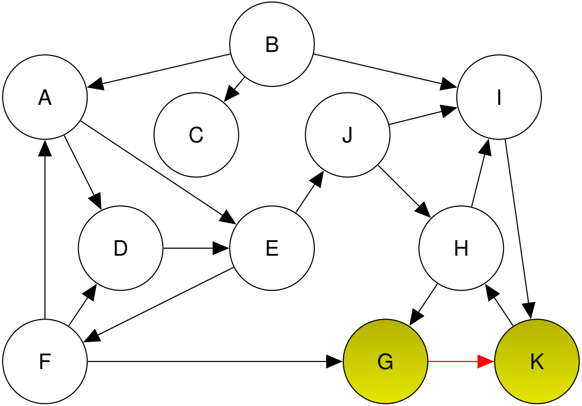


Componenti fortemente connesse

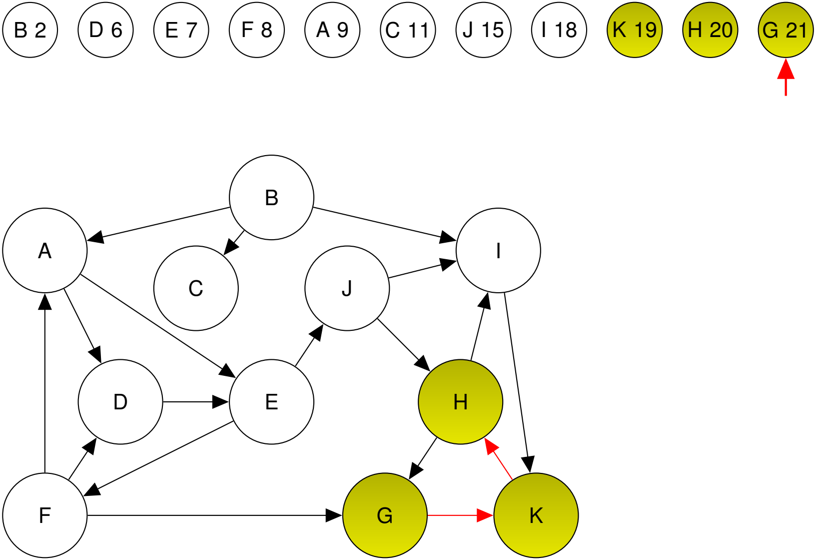


Componenti fortemente connesse

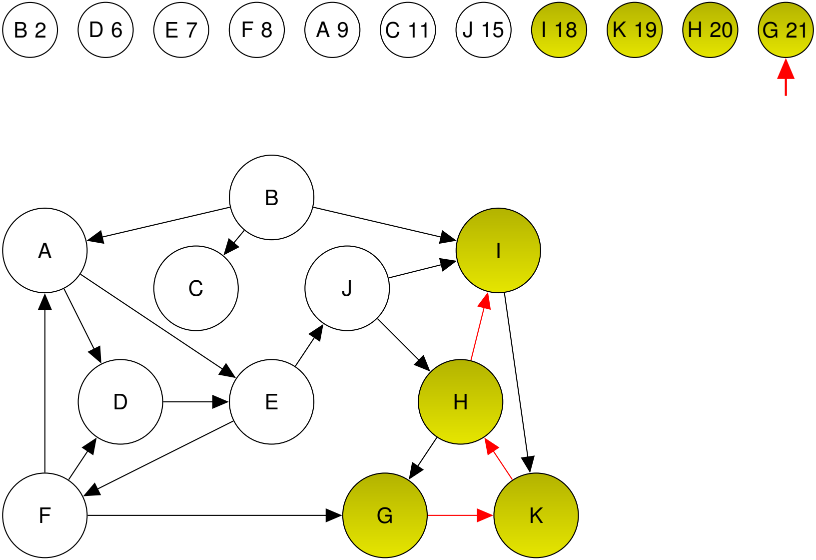




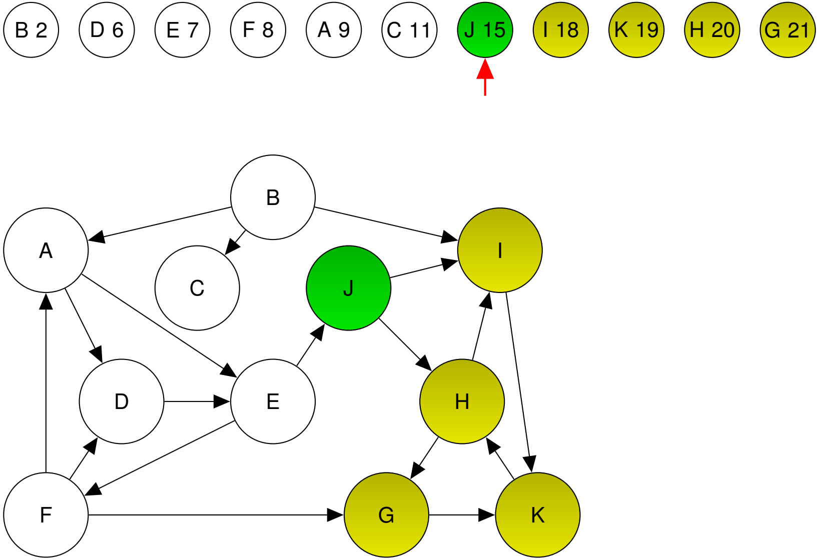
Componenti fortemente connesse



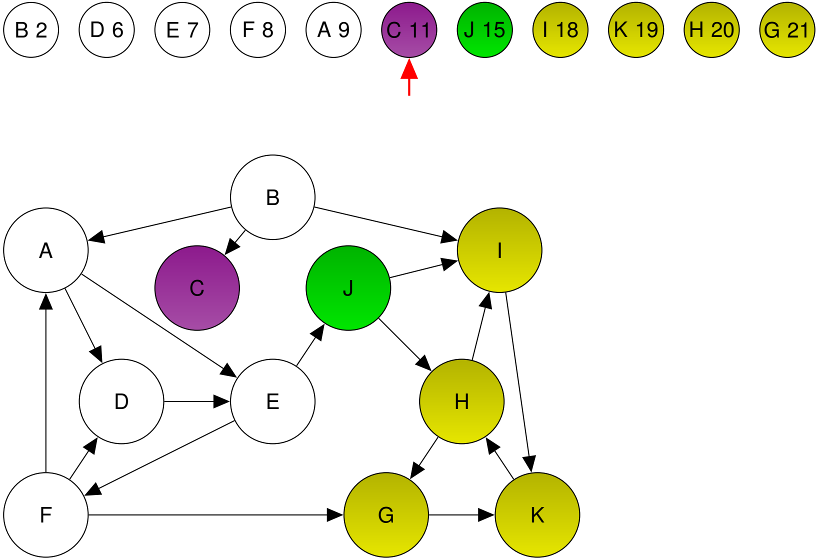
Componenti fortemente connesse



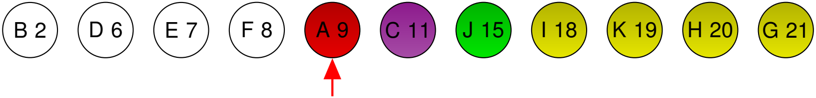
Componenti fortemente connesse

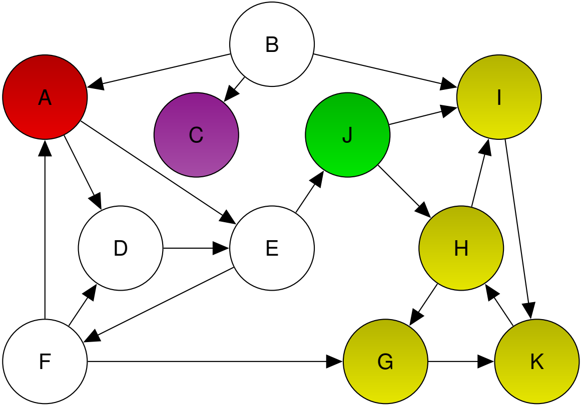


Componenti fortemente connesse

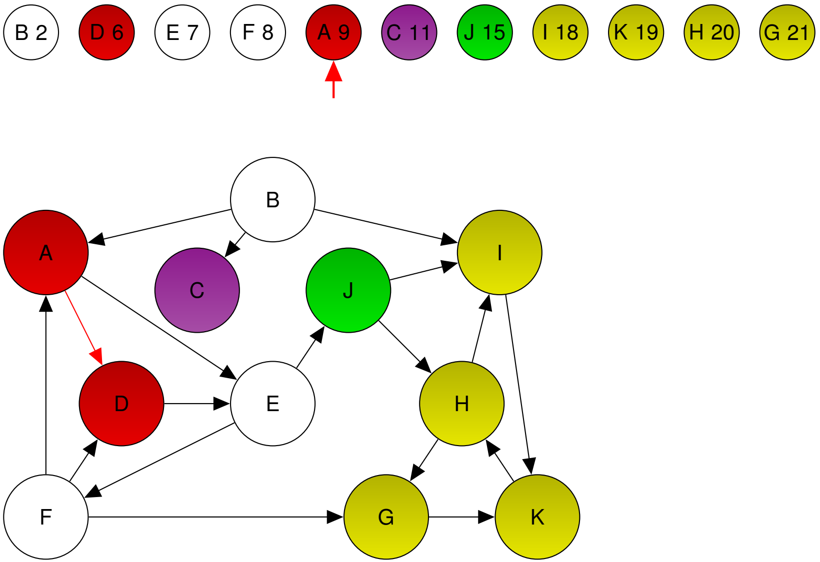


Componenti fortemente connesse

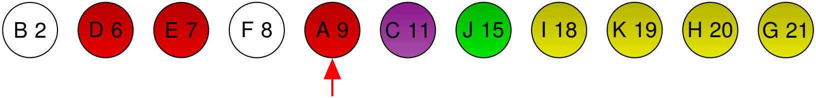


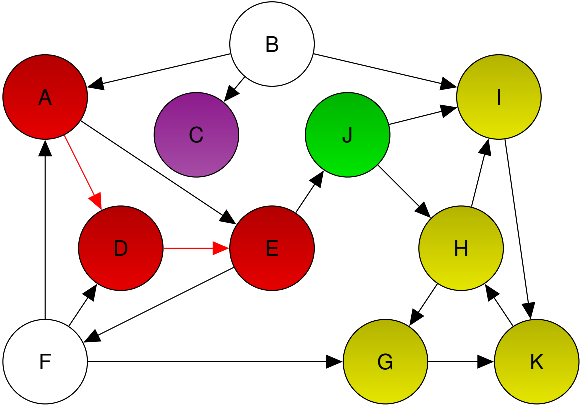


Componenti fortemente connesse

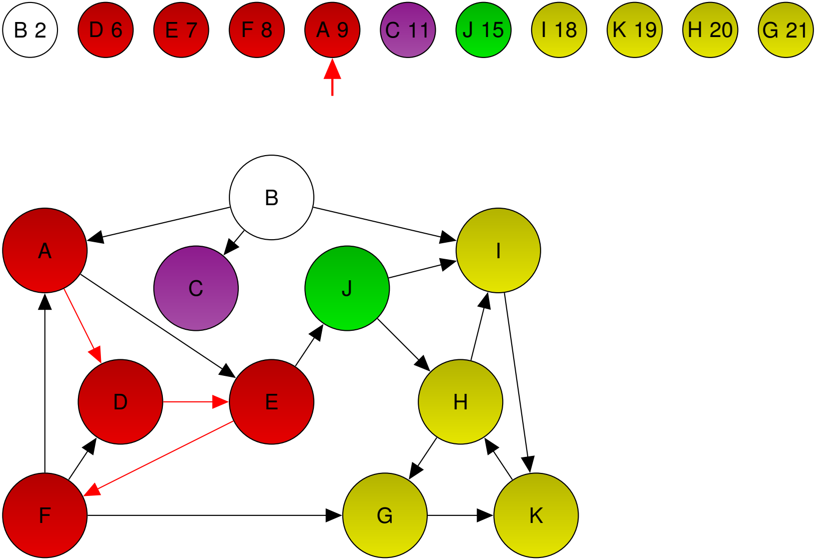


Componenti fortemente connesse

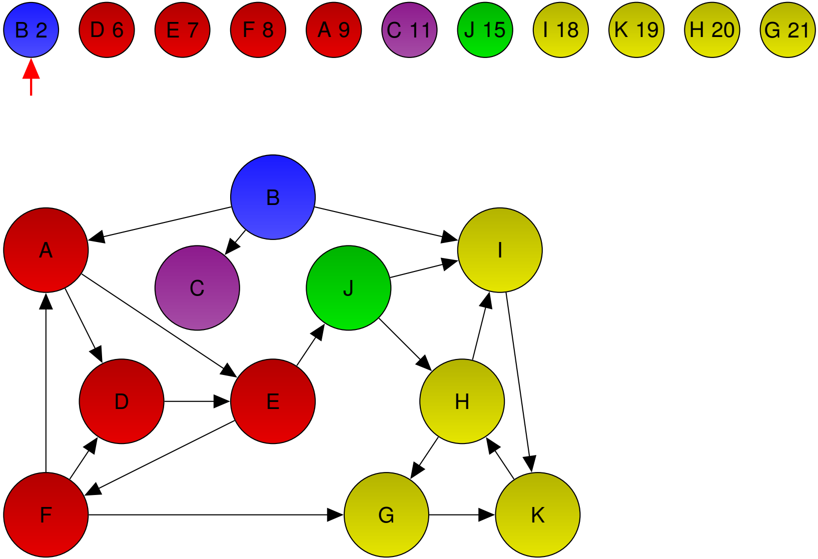




Componenti fortemente connesse



Componenti fortemente connesse



Longest increasing sequence

Data la sequenza

6	3	4	12	8	9	14	7	10	8

calcolare una sottosequenza crescente di lunghezza massima

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1									
Parent	-1									

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1								
Parent	-1	-1								

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2							
Parent	-1	-1	1							

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3						
Parent	-1	-1	1	2						

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3					
Parent	-1	-1	1	2	2					

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4				
Parent	-1	-1	1	2	2	4				

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5			
Parent	-1	-1	1	2	2	4	5			

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3		
Parent	-1	-1	1	2	2	4	5	2		

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3	5	
Parent	-1	-1	1	2	2	4	5	2	5	

Longest increasing sequence

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3	5	4
Parent	-1	-1	1	2	2	4	5	2	5	7

Longest increasing sequence

LIS: ultimo elemento in posizione 8 (massimo del vettore LIS)

Completiamo la sequenza seguendo all'indietro i riferimenti nel vettore Parent:

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3	5	4
Parent	-1	-1	1	2	2	4	5	2	5	7

Longest increasing sequence

LIS: ultimo elemento in posizione 8 (massimo del vettore LIS

Completiamo la sequenza seguendo all'indietro i riferimenti nel vettore Parent:

	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3	5	4
Parent	-1	-1	1	2	2	4	5	2	5	7

In alternativa, una seconda sequenza termina con l'elemento in posizione 6

Longest increasing sequence

Equivalente a ricerca cammino di lunghezza massima su DAG

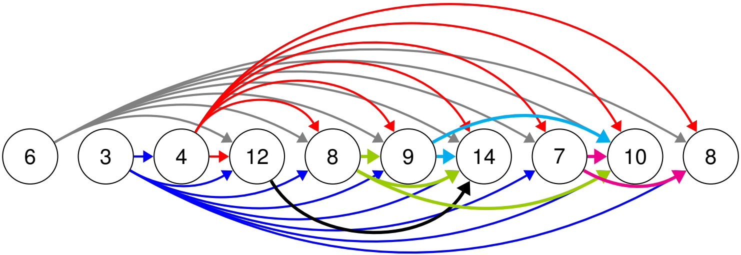
Ogni elemento della sequenza è un nodo

I nodi sono (topologicamente) ordinati secondo l'ordine della sequenza

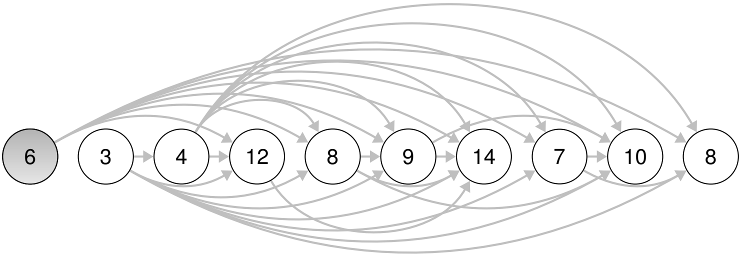
Un arco (u, v) collega i nodi u e v se e solo se v segue u nell'ordinamento (topologico) e il valore di v è maggiore del valore di u

Applichiamo l'algoritmo per i cammini massimi, mettendo $d(v) = 1$ per i nodi senza predecessori

6	3	4	12	8	9	14	7	10	8

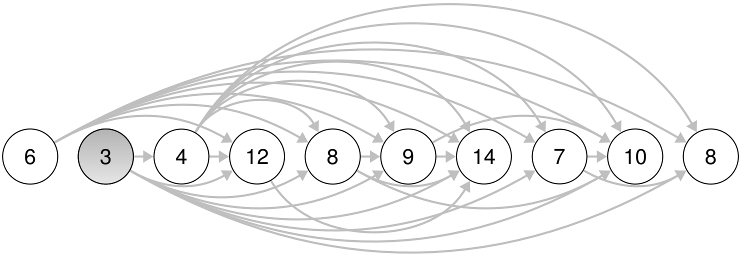


Longest increasing sequence



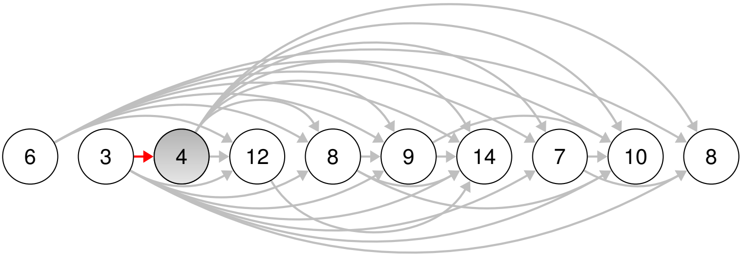
1

Longest increasing sequence



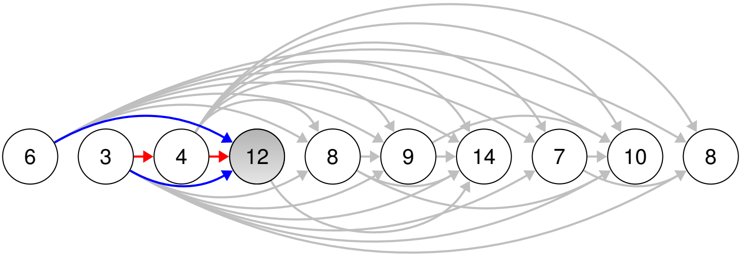
1 1

Longest increasing sequence



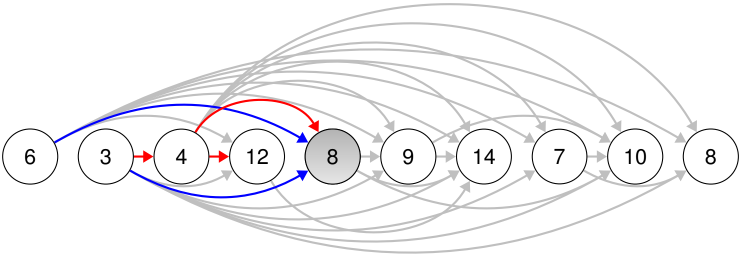
1 1 2

Longest increasing sequence



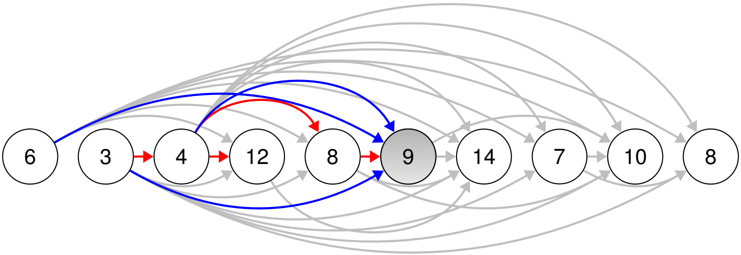
1 1 2 3

Longest increasing sequence



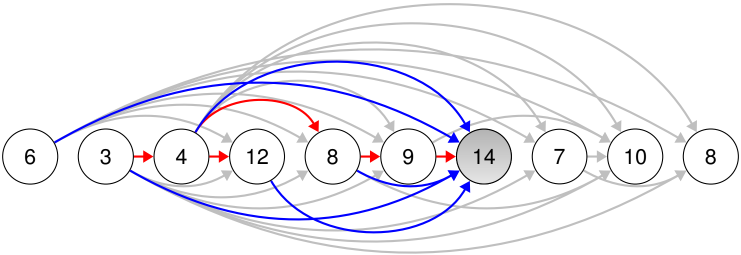
1 1 2 3 3

Longest increasing sequence



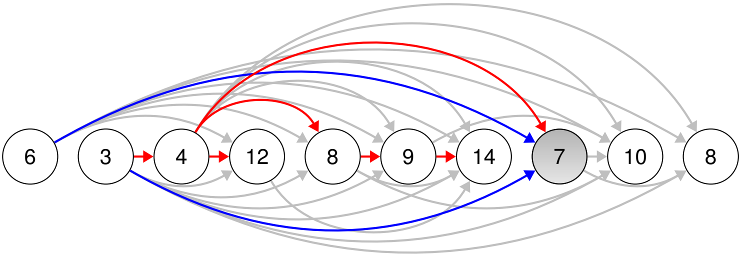
1 1 2 3 3 4

Longest increasing sequence



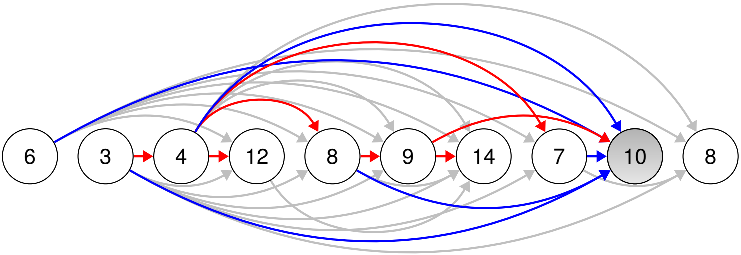
1 1 2 3 3 4 5

Longest increasing sequence



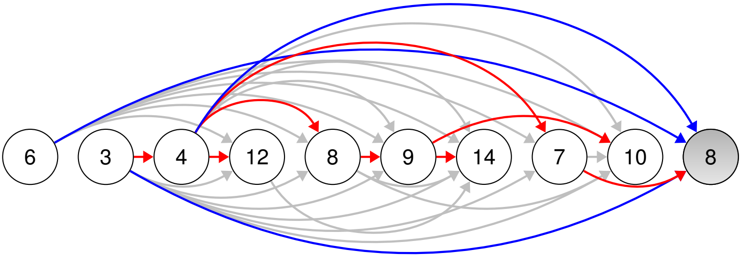
1 1 2 3 3 4 5 3

Longest increasing sequence



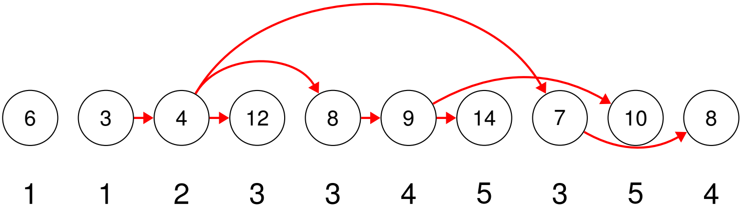
1 1 2 3 3 4 5 3 5

Longest increasing sequence



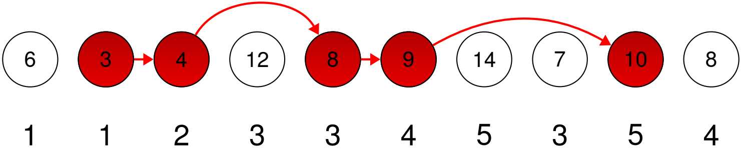
1	1	2	3	3	4	5	3	5	4

Longest increasing sequence



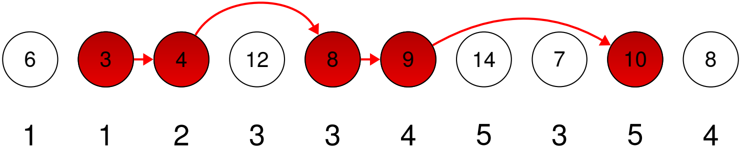
Una LIS termina in nodo 10 (alternativamente, in nodo 14)

Seguendo gli archi a ritroso possiamo ricostruire la sequenza

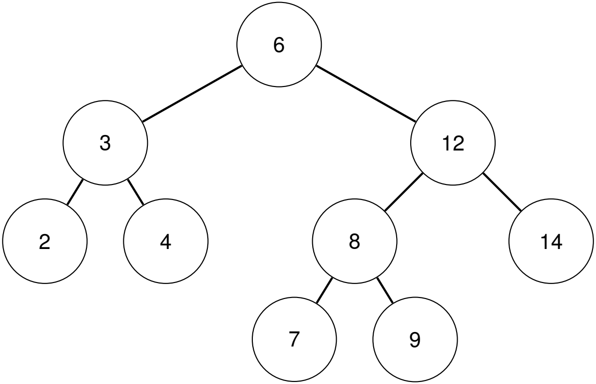


Longest increasing sequence

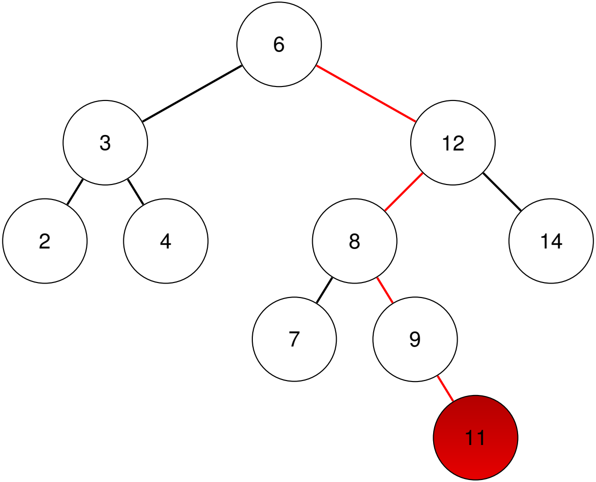
	0	1	2	3	4	5	6	7	8	9
	6	3	4	12	8	9	14	7	10	8
LIS	1	1	2	3	3	4	5	3	5	4
Parent	-1	-1	1	2	2	4	5	2	5	7



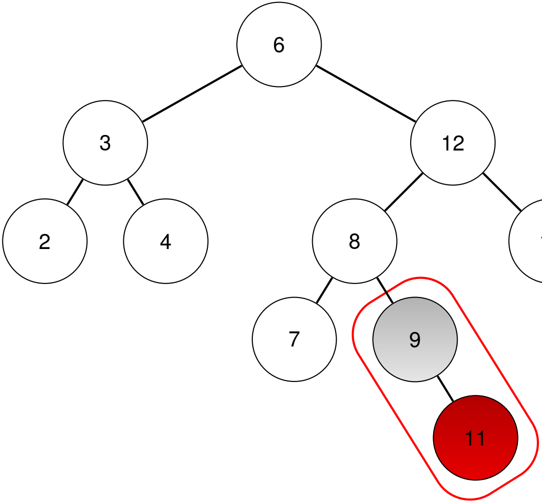
Inserimento in radice: dato il seguente BST, inserire il valore 11 in radice



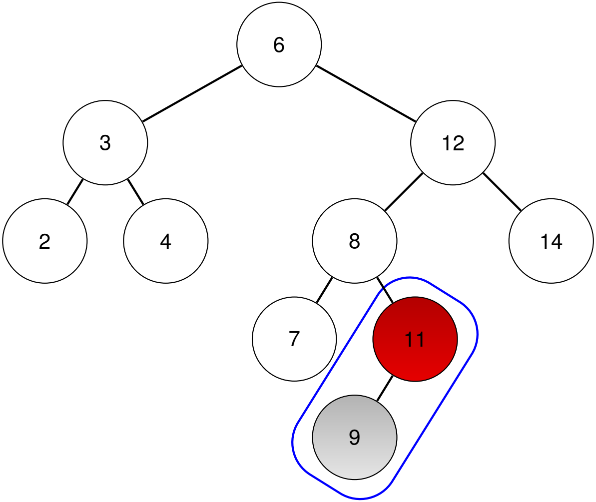
Inseriamo il nodo come foglia



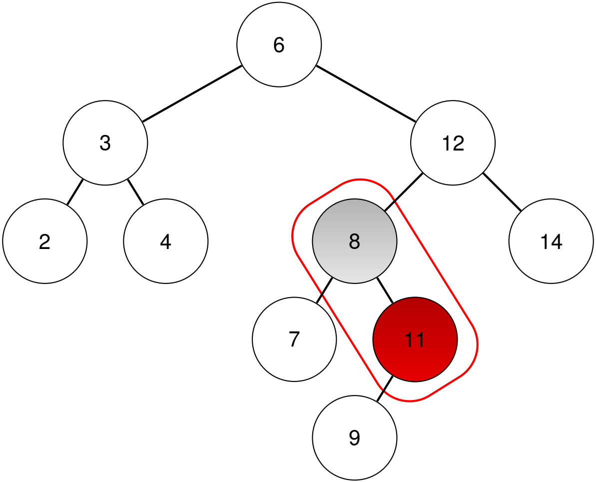
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



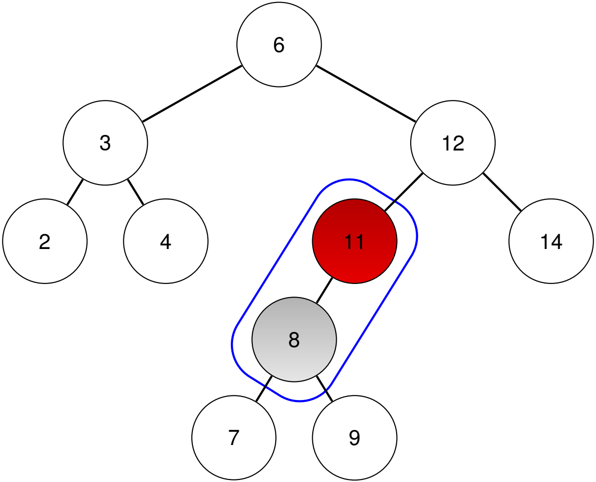
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell’albero



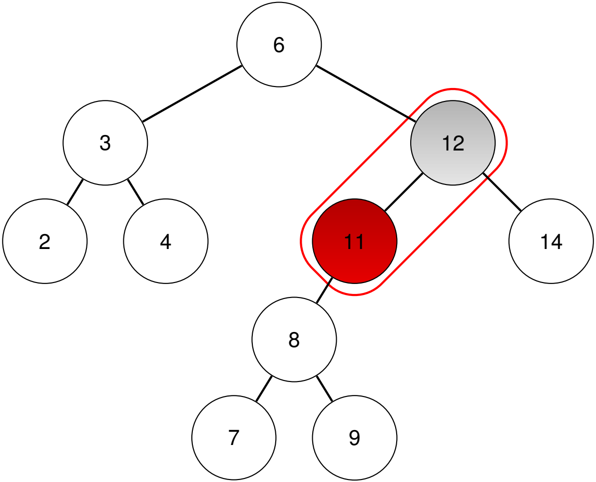
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



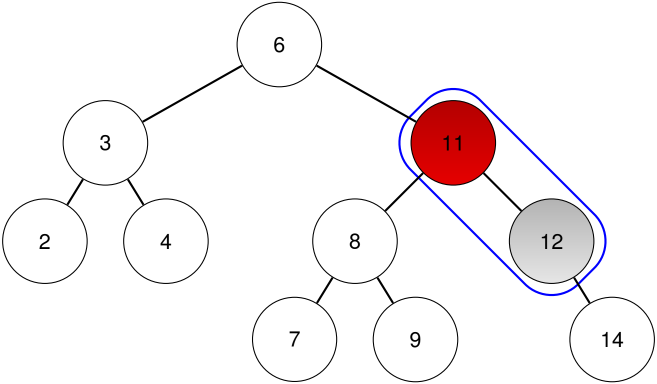
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



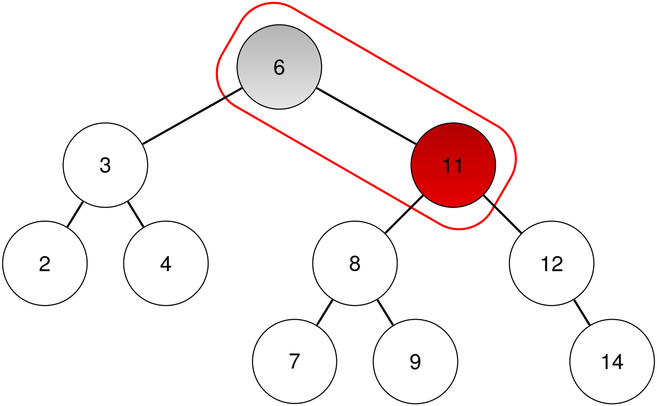
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell’albero



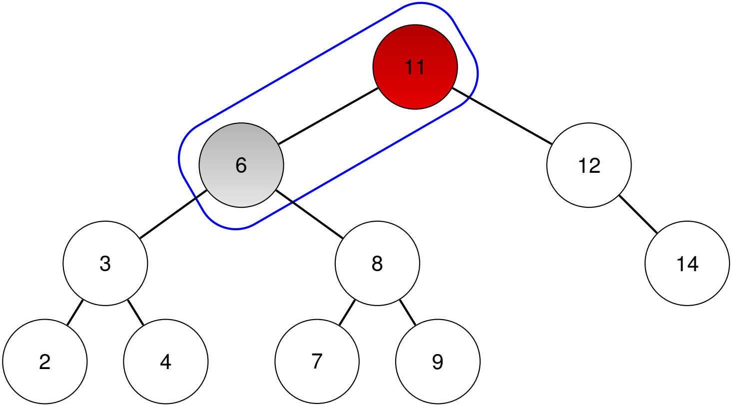
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



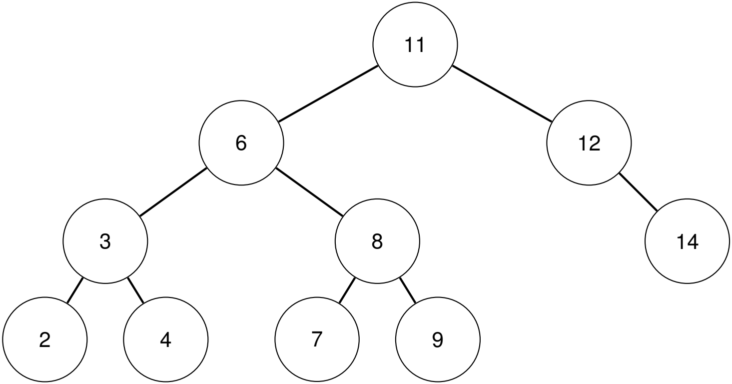
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



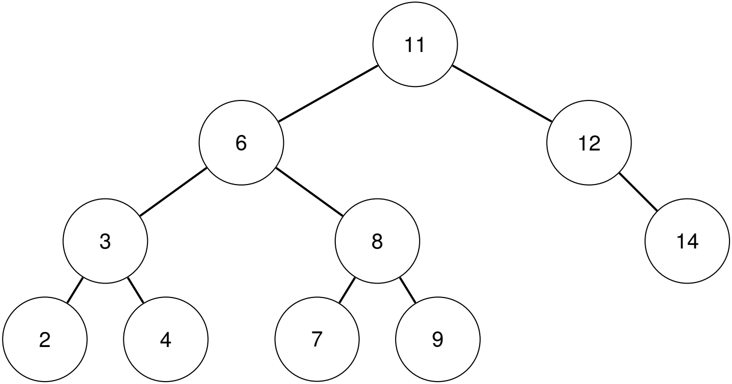
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



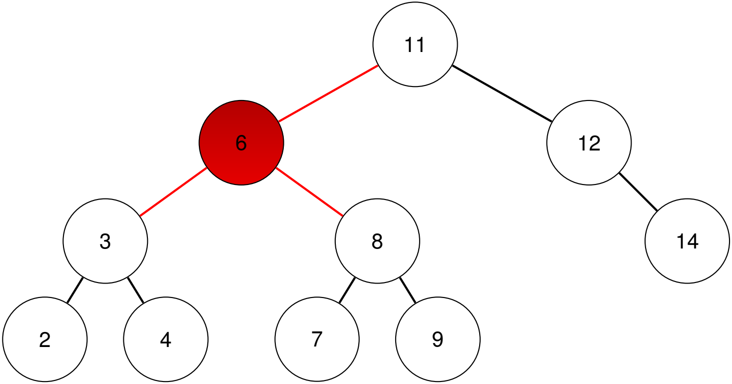
Effettuiamo le rotazioni per far diventare il nodo inserito la radice dell'albero



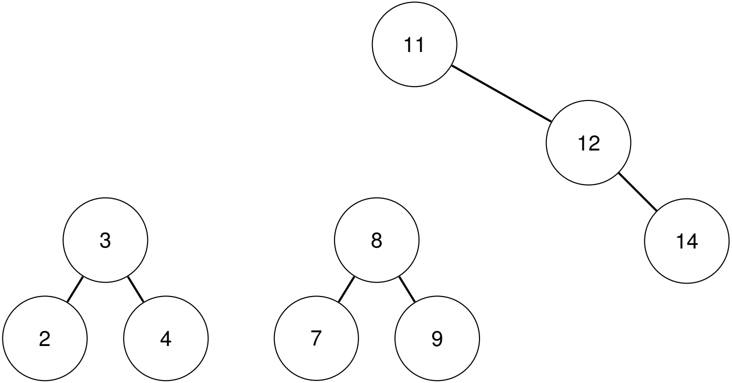
Cancellazione: vogliamo rimuovere il nodo 6



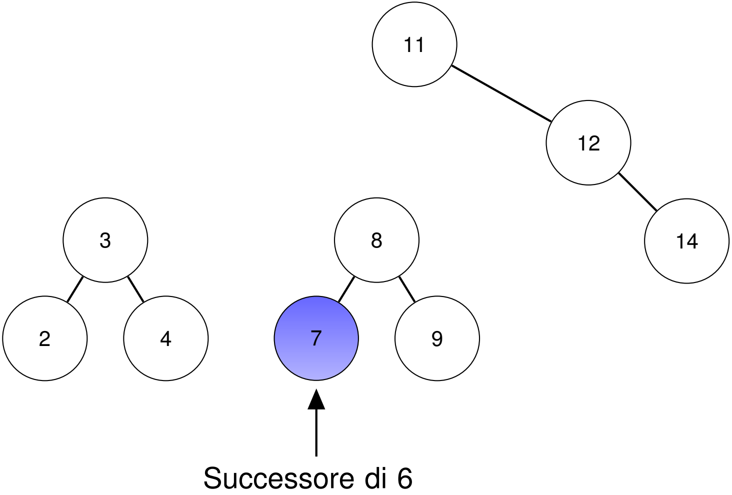
Cancellazione: vogliamo rimuovere il nodo 6



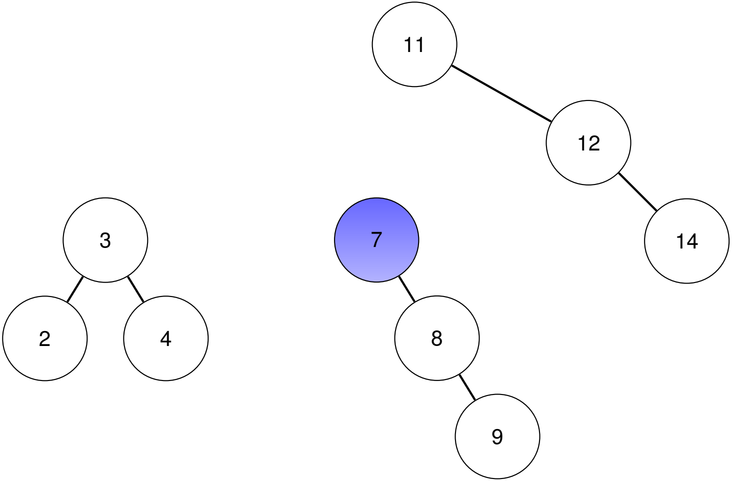
Cancellazione: vogliamo rimuovere il nodo 6



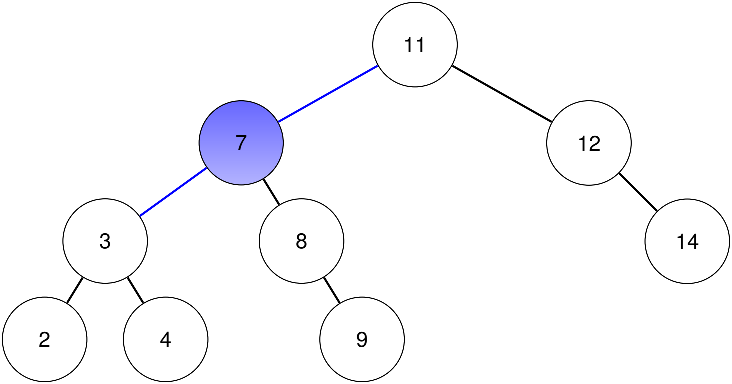
Cancellazione: vogliamo rimuovere il nodo 6



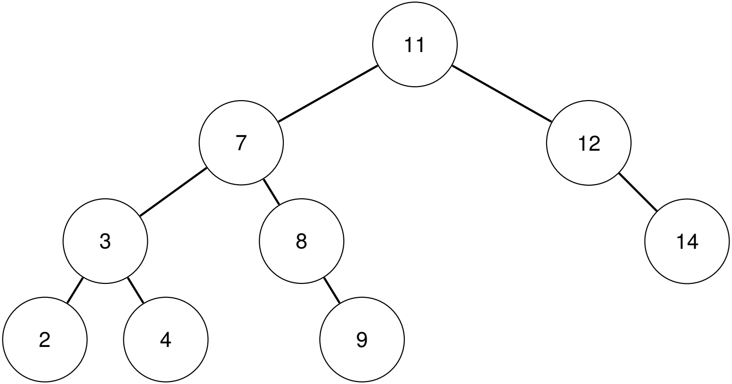
Partizionamento del sottoalbero DX rispetto al nodo 7



Ricostruzione dell'albero



Ricostruzione dell'albero

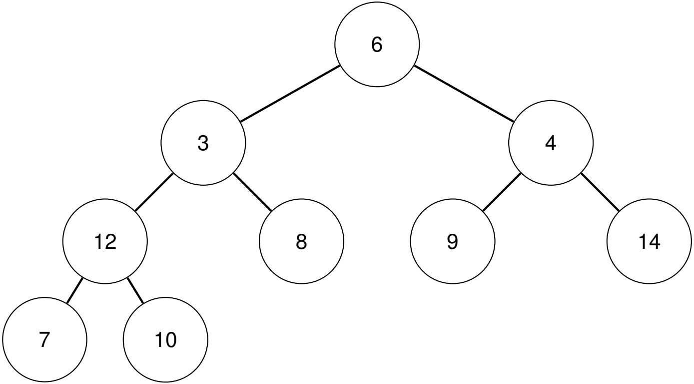


Ordinare tramite Heapsort il vettore

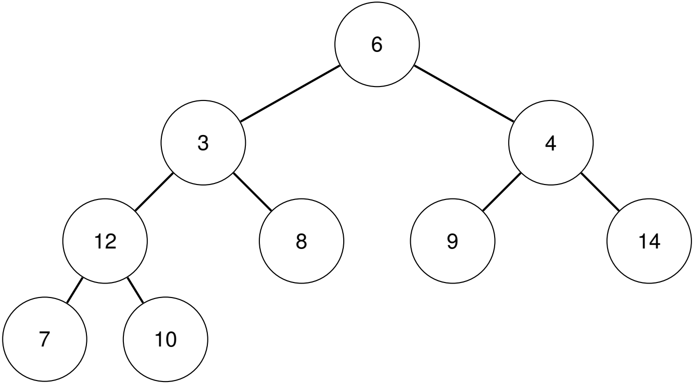
6	3	4	12	8	9	14	7	10

Ordinare tramite Heapsort il vettore

6	3	4	12	8	9	14	7	10



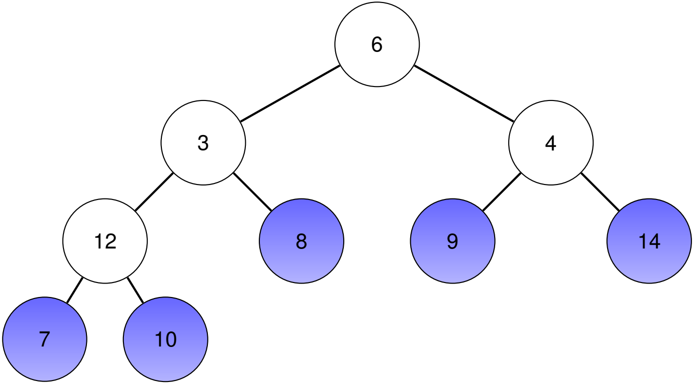
Heapbuild: trasformazione dell'albero in un heap



6	3	4	12	8	9	14	7	10

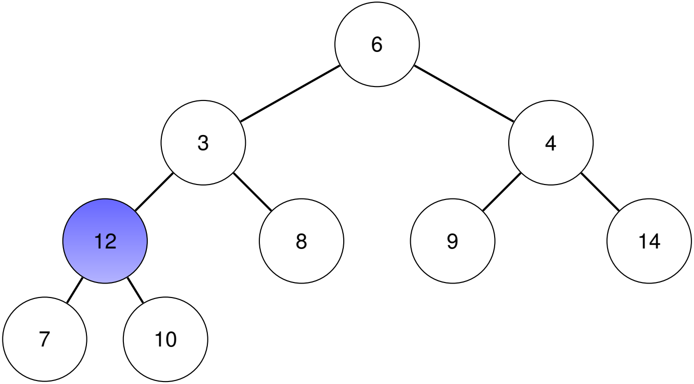
Heapsort

Le foglie sono già heap



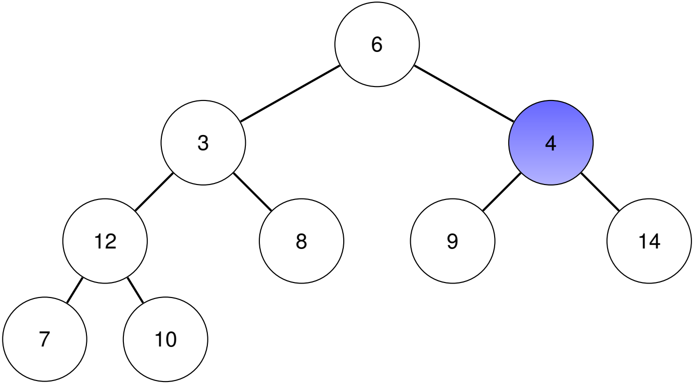
6	3	4	12	8	9	14	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



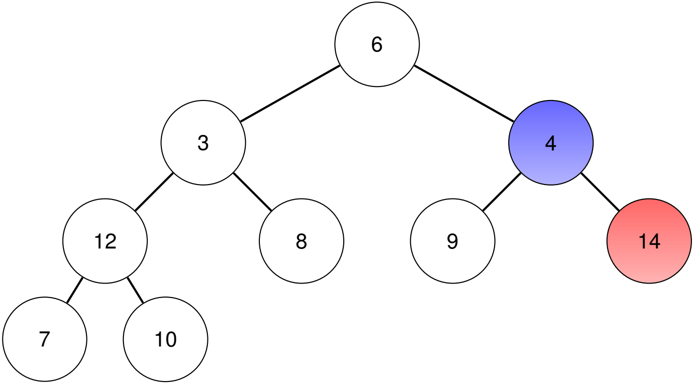
6	3	4	12	8	9	14	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



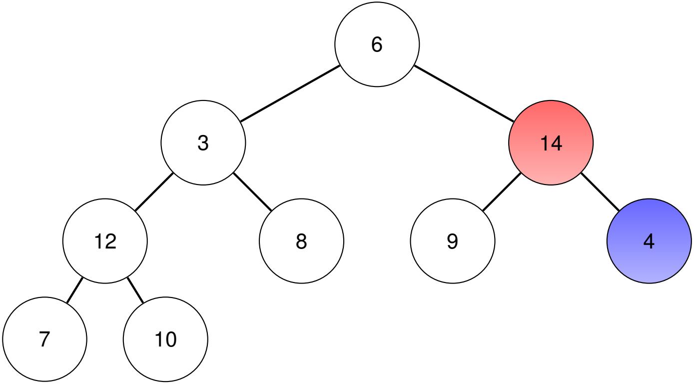
6	3	4	12	8	9	14	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



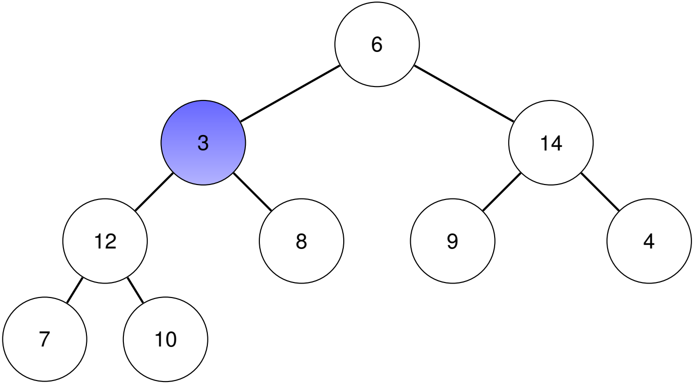
6	3	4	12	8	9	14	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



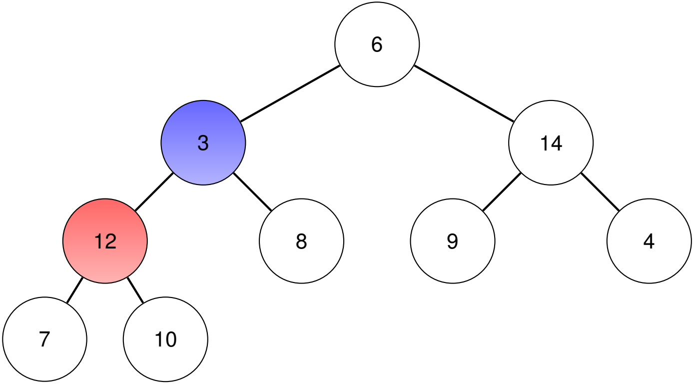
6	3	14	12	8	9	4	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



6	3	14	12	8	9	4	7	10

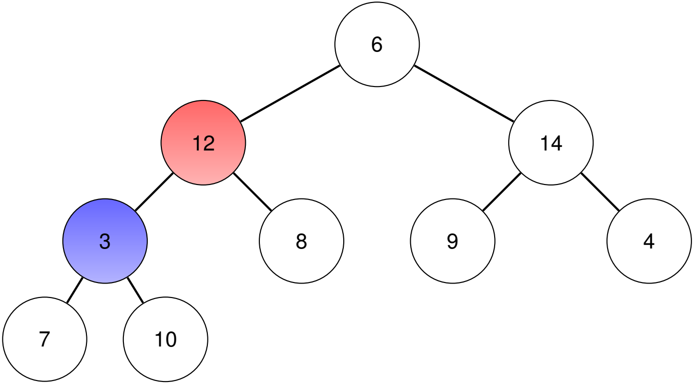
Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



6	3	14	12	8	9	4	7	10

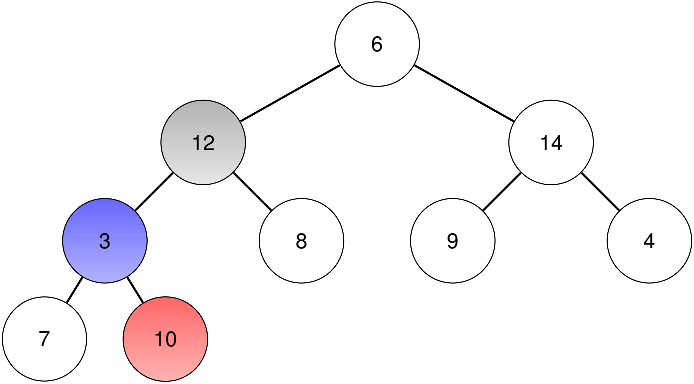
Heapsort

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



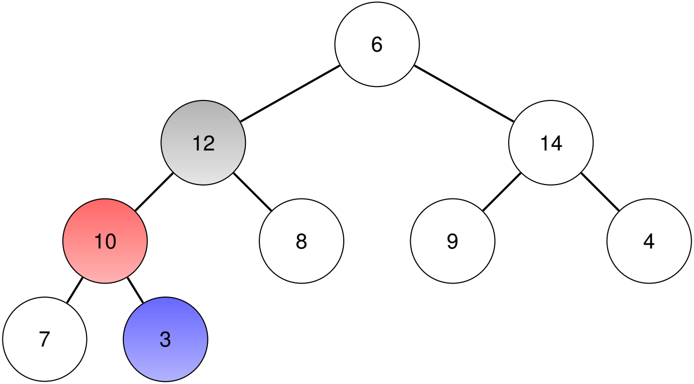
6	12	14	3	8	9	4	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



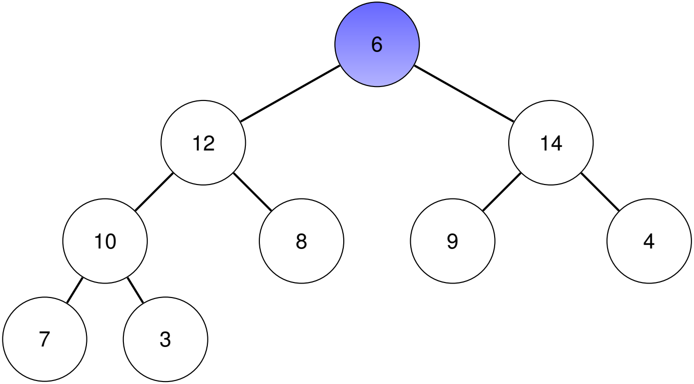
6	12	14	3	8	9	4	7	10

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



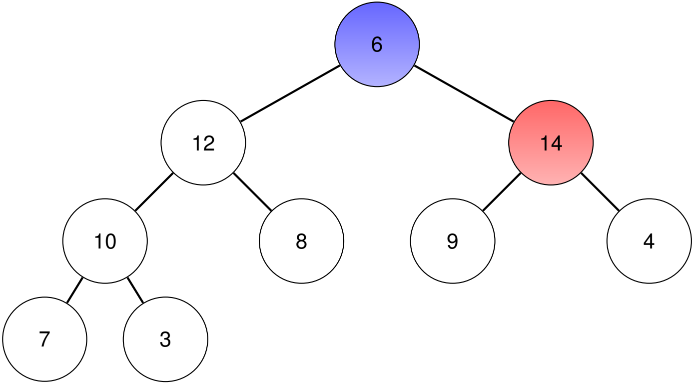
6	12	14	10	8	9	4	7	3

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



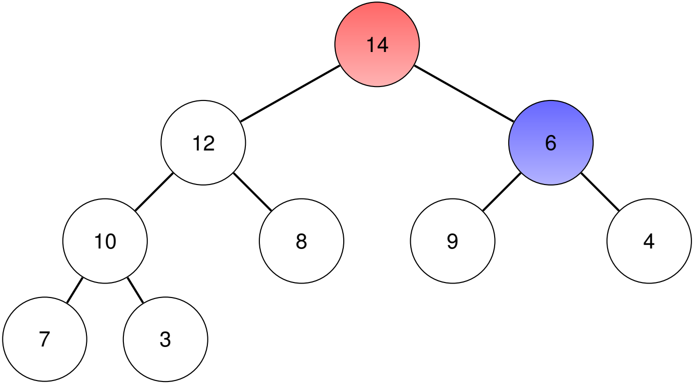
6	12	14	10	8	9	4	7	3

Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



6	12	14	10	8	9	4	7	3

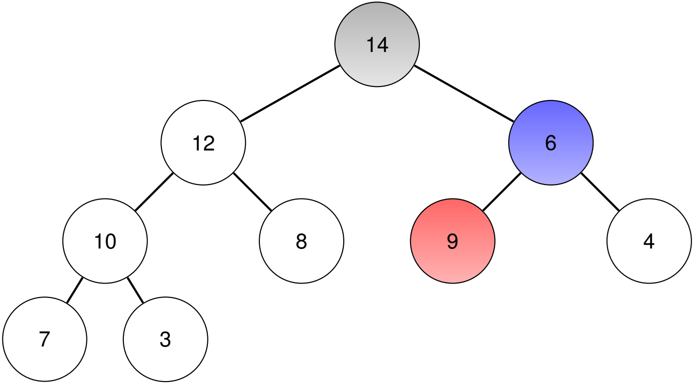
Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



14	12	6	10	8	9	4	7	3

Heapsort

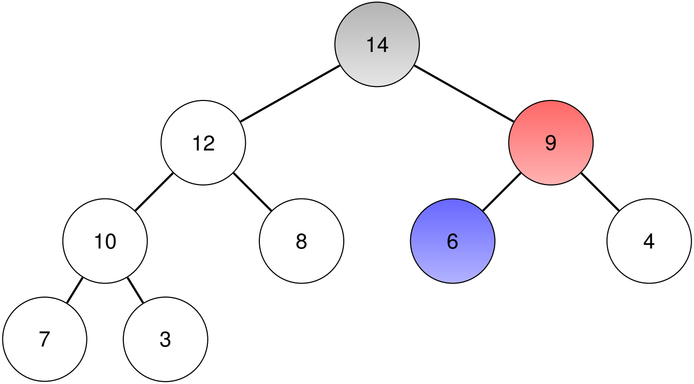
Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



14	12	6	10	8	9	4	7	3

Heapsort

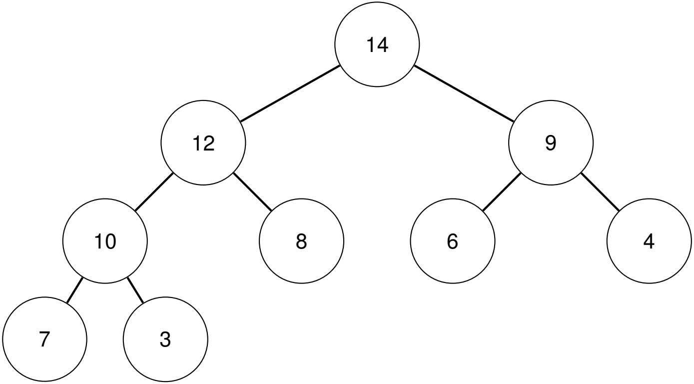
Applichiamo di HEAPify dal primo nodo non foglia fino alla radice



14	12	9	10	8	6	4	7	3

Heapsort

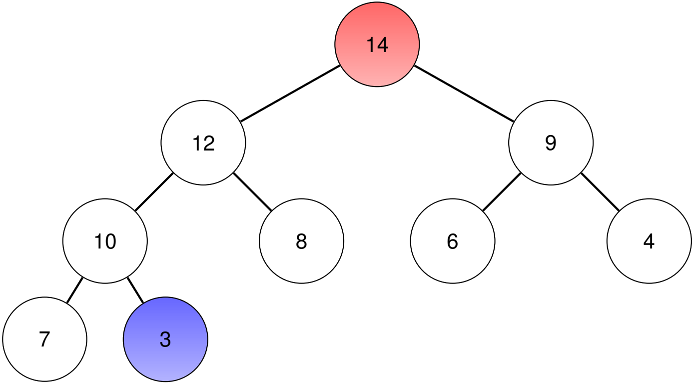
Heap:



14	12	9	10	8	6	4	7	3

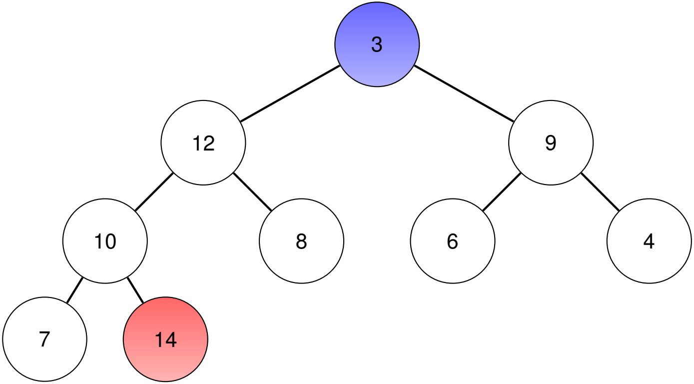
Heapsort

Scambio primo e ultimo elemento:



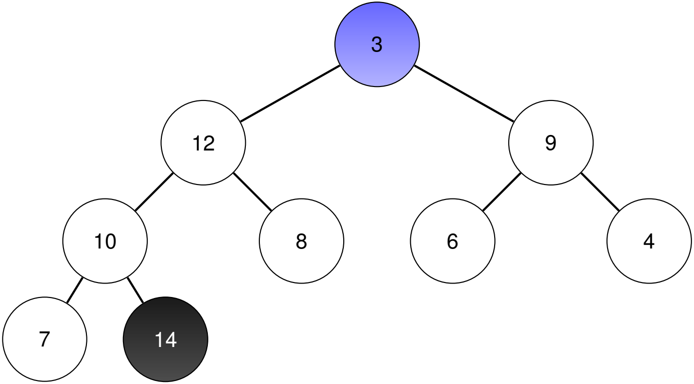
14	12	9	10	8	6	4	7	3

Scambio primo e ultimo elemento:



3	12	9	10	8	6	4	7	14

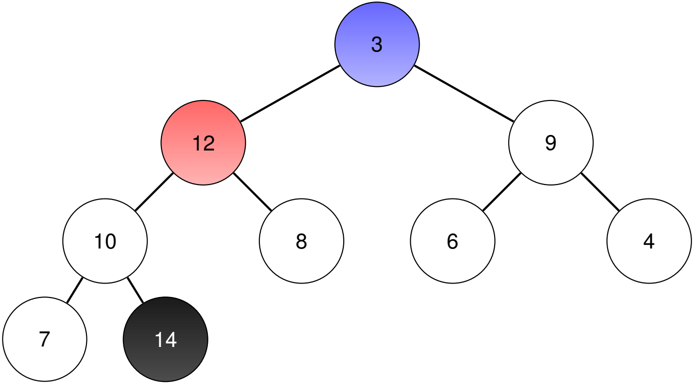
Ricostruzione Heap:



3	12	9	10	8	6	4	7	14

Heapsort

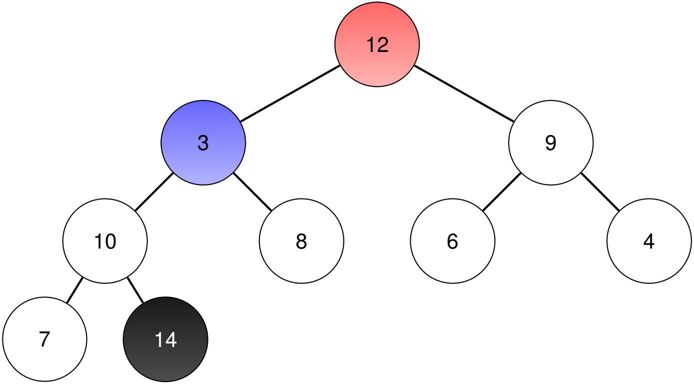
Ricostruzione Heap:



3	12	9	10	8	6	4	7	14

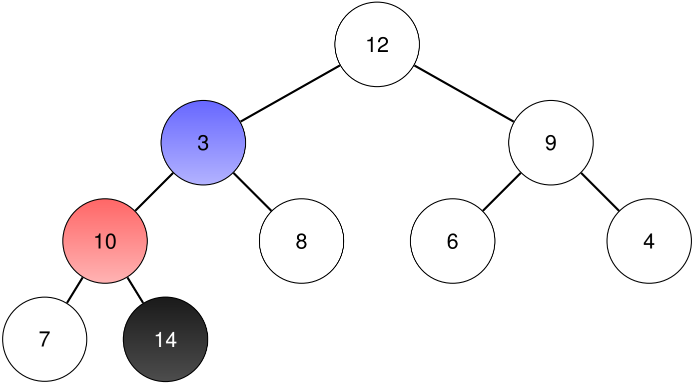
Heapsort

Ricostruzione Heap:



12	3	9	10	8	6	4	7	14

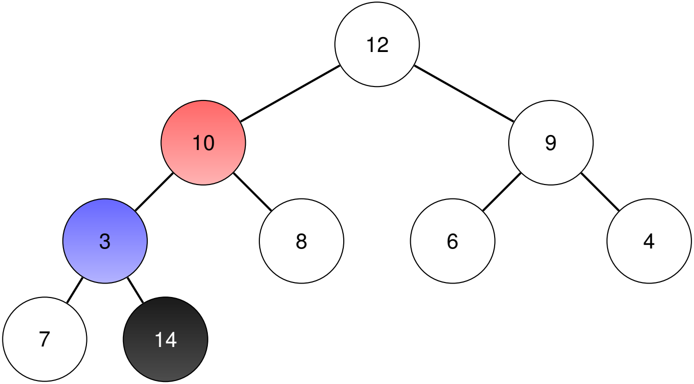
Ricostruzione Heap:



12	3	9	10	8	6	4	7	14

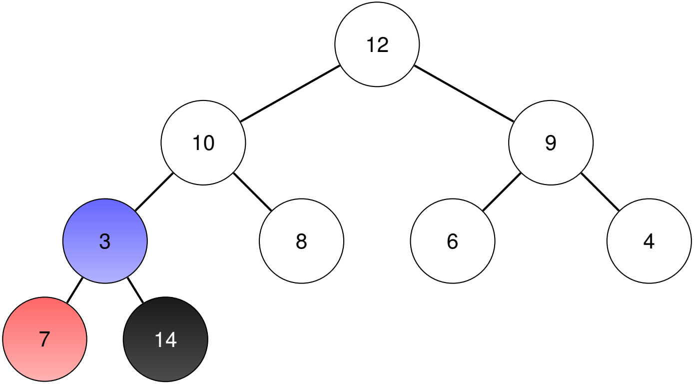
Heapsort

Ricostruzione Heap:



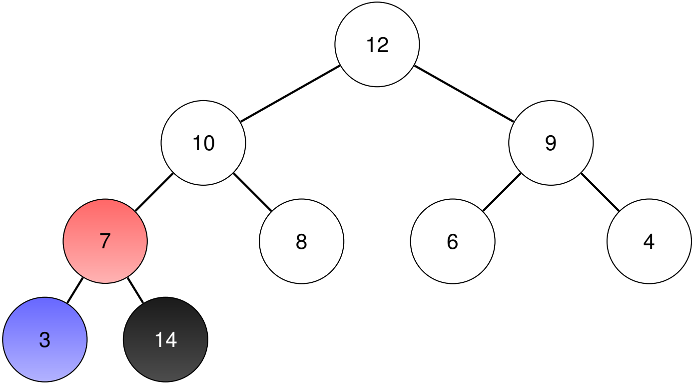
12	10	9	3	8	6	4	7	14

Ricostruzione Heap:



12	10	9	3	8	6	4	7	14

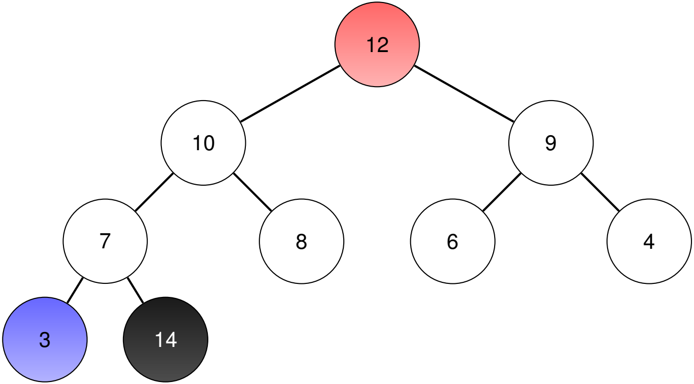
Ricostruzione Heap:



12	10	9	7	8	6	4	3	14

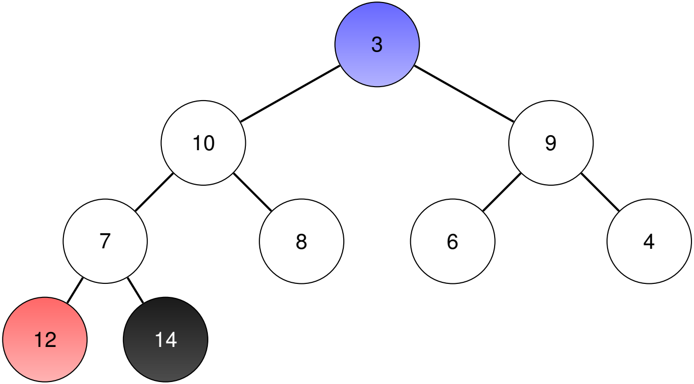
Heapsort

Scambio primo e ultimo elemento:



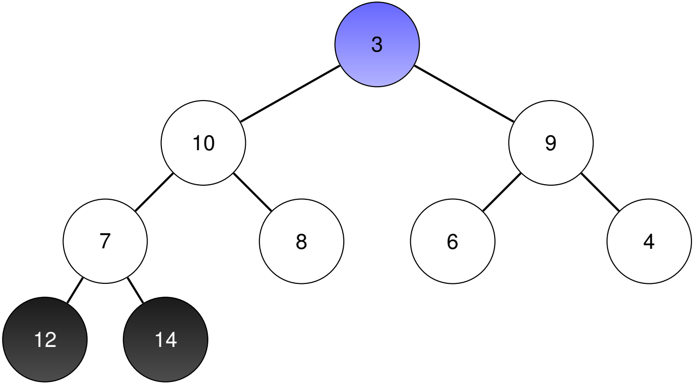
12	10	9	7	8	6	4	3	14

Scambio primo e ultimo elemento:



3	10	9	7	8	6	4	12	14

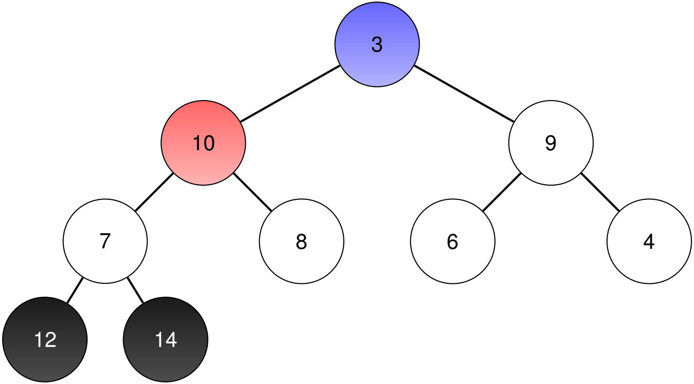
Ricostruzione Heap:



3	10	9	7	8	6	4	12	14

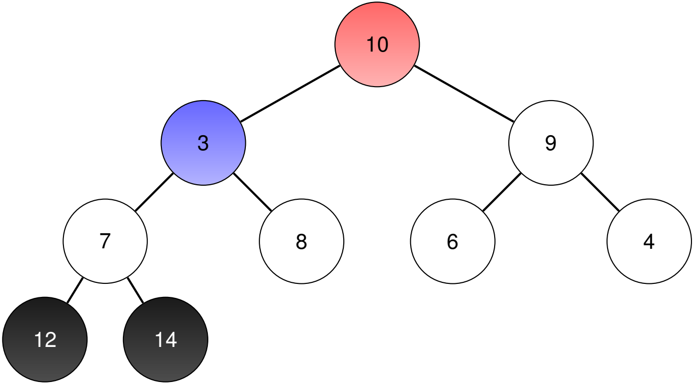
Heapsort

Ricostruzione Heap:



3	10	9	7	8	6	4	12	14

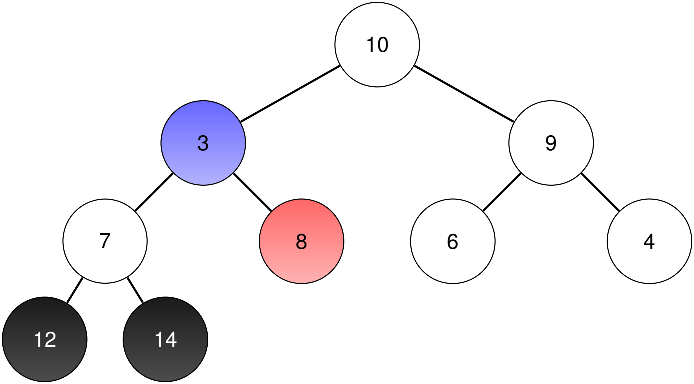
Ricostruzione Heap:



10	3	9	7	8	6	4	12	14

Heapsort

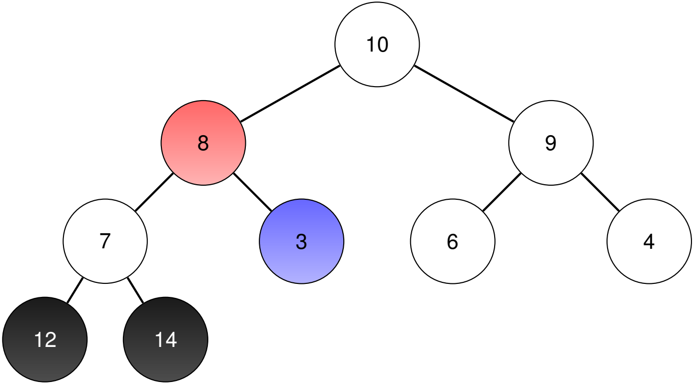
Ricostruzione Heap:



10	3	9	7	8	6	4	12	14

Heapsort

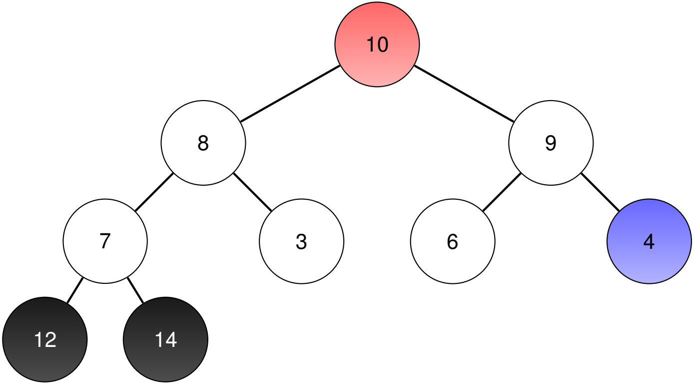
Ricostruzione Heap:



10	8	9	7	3	6	4	12	14

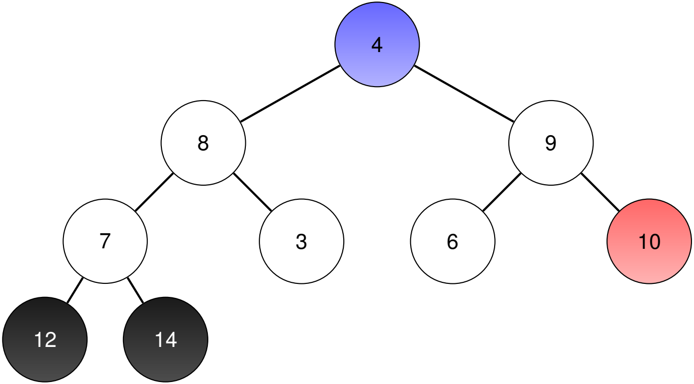
Heapsort

Scambio primo e ultimo elemento:



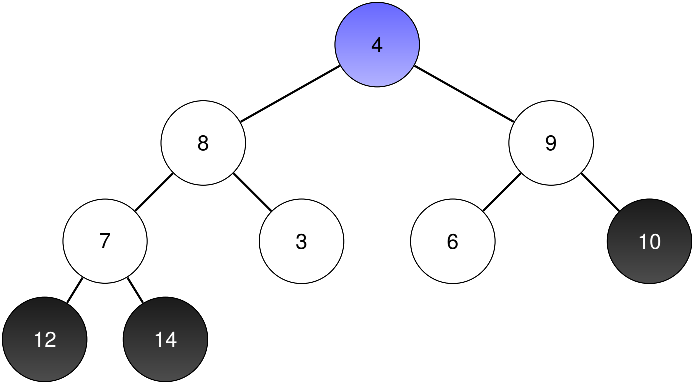
10	8	9	7	3	6	4	12	14

Scambio primo e ultimo elemento:



4	8	9	7	3	6	10	12	14

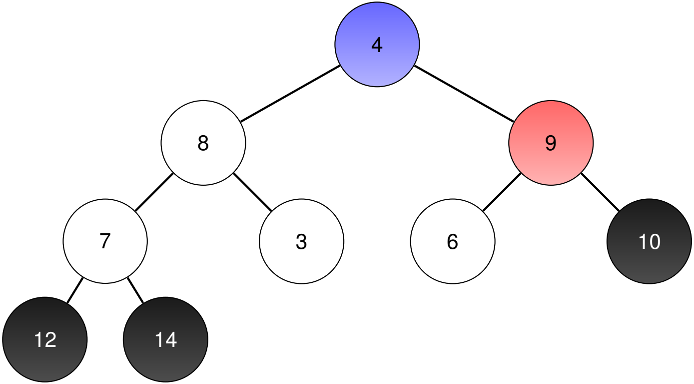
Ricostruzione Heap:



4	8	9	7	3	6	10	12	14

Heapsort

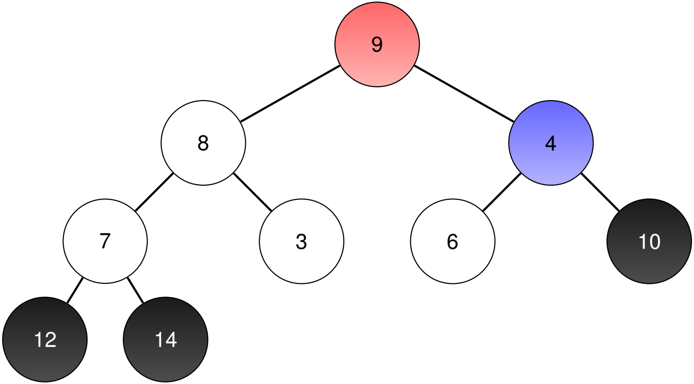
Ricostruzione Heap:



4	8	9	7	3	6	10	12	14

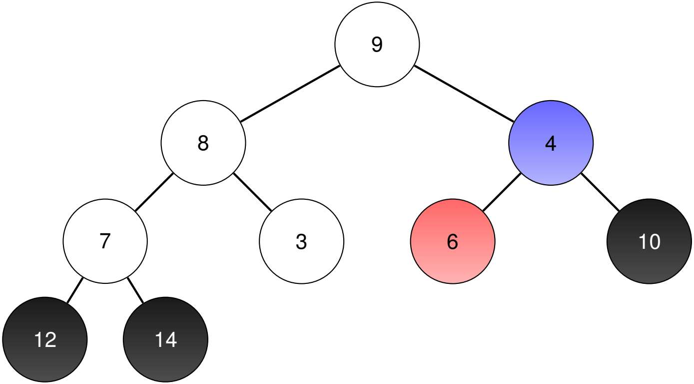
Heapsort

Ricostruzione Heap:



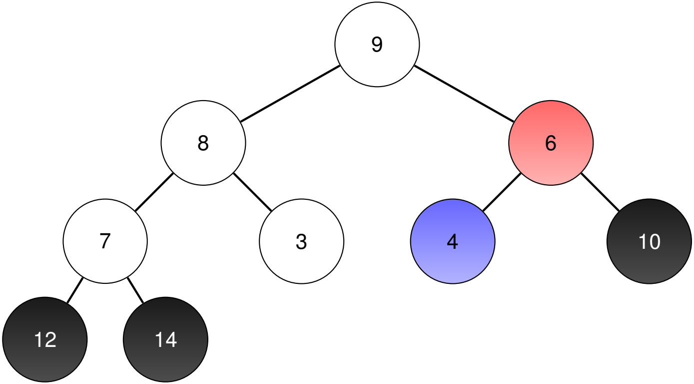
9	8	4	7	3	6	10	12	14

Ricostruzione Heap:



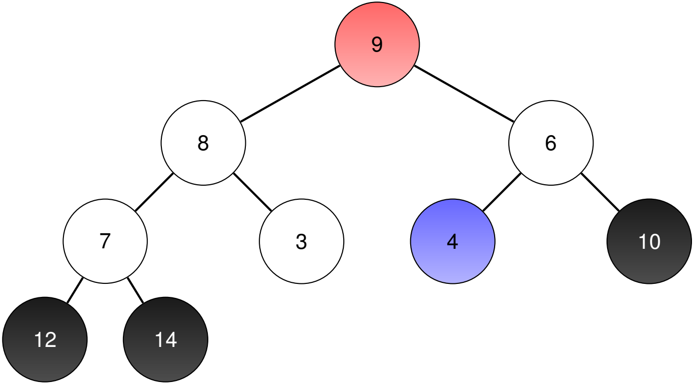
9	8	4	7	3	6	10	12	14

Ricostruzione Heap:



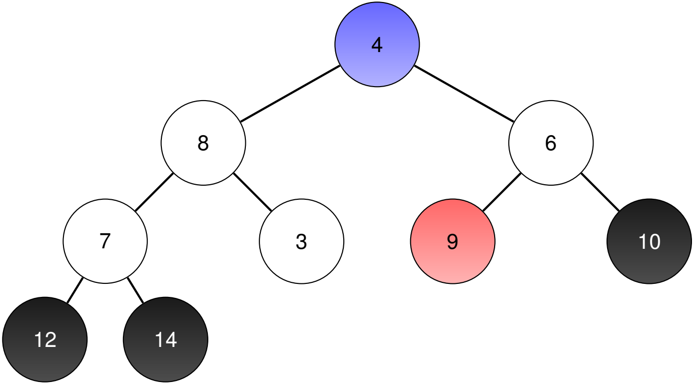
9	8	6	7	3	4	10	12	14

Scambio primo e ultimo elemento:



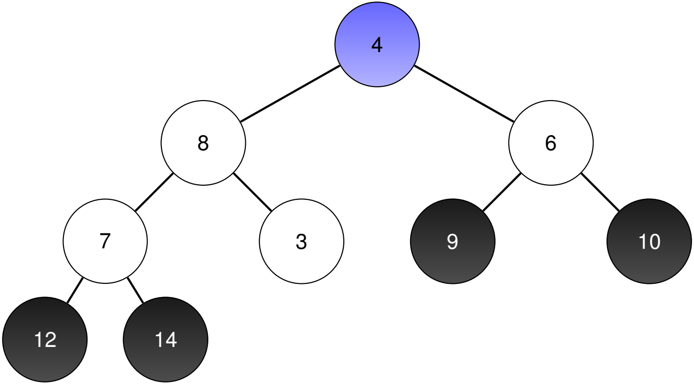
9	8	6	7	3	4	10	12	14

Scambio primo e ultimo elemento:



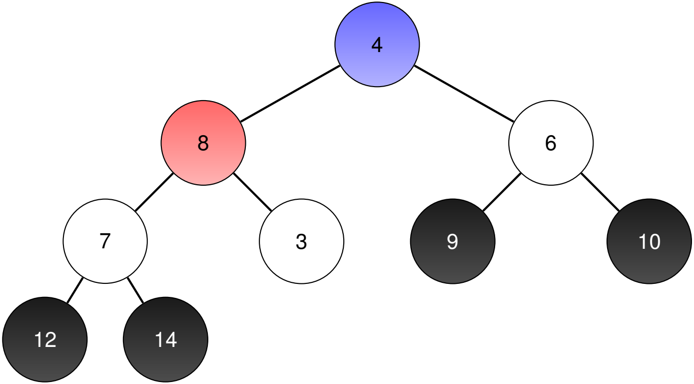
4	8	6	7	3	9	10	12	14

Ricostruzione Heap:



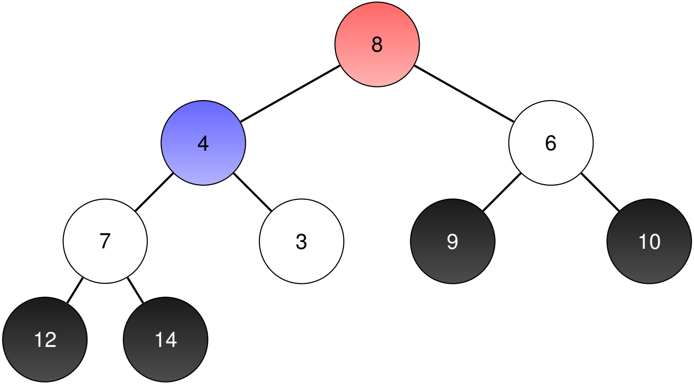
4	8	6	7	3	9	10	12	14

Ricostruzione Heap:



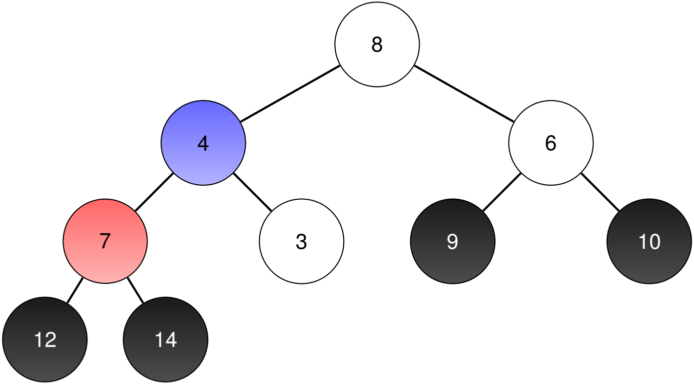
4	8	6	7	3	9	10	12	14

Ricostruzione Heap:



8	4	6	7	3	9	10	12	14

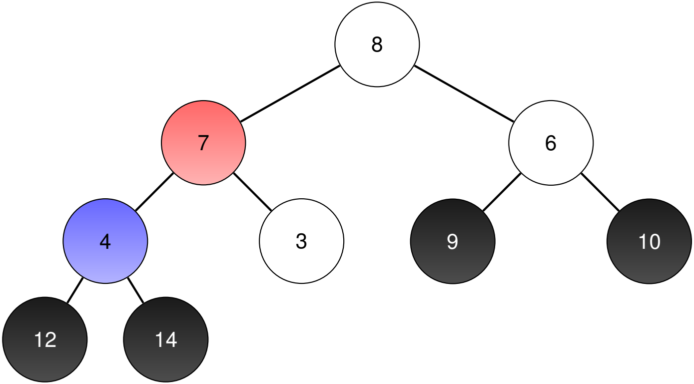
Ricostruzione Heap:



8	4	6	7	3	9	10	12	14

Heapsort

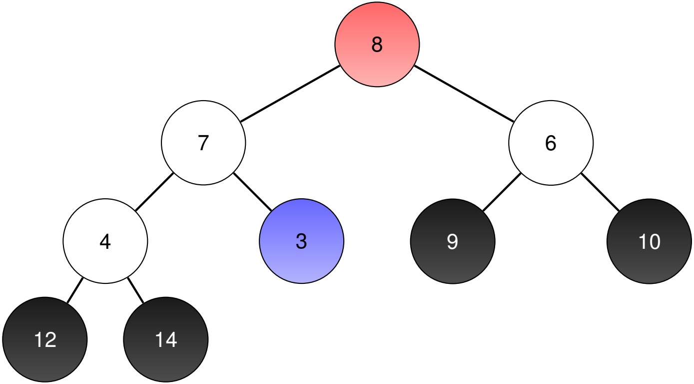
Ricostruzione Heap:



8	7	6	4	3	9	10	12	14

Heapsort

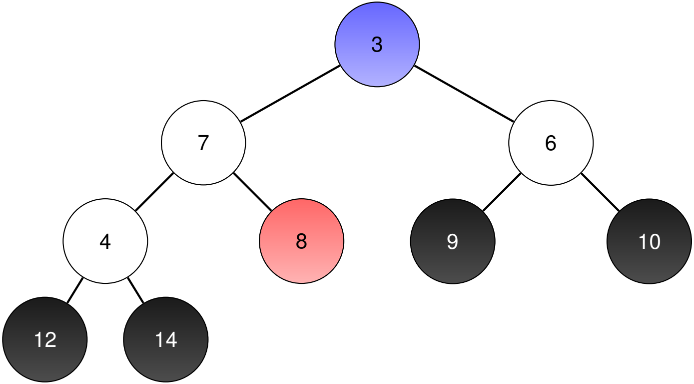
Scambio primo e ultimo elemento:



8	7	6	4	3	9	10	12	14

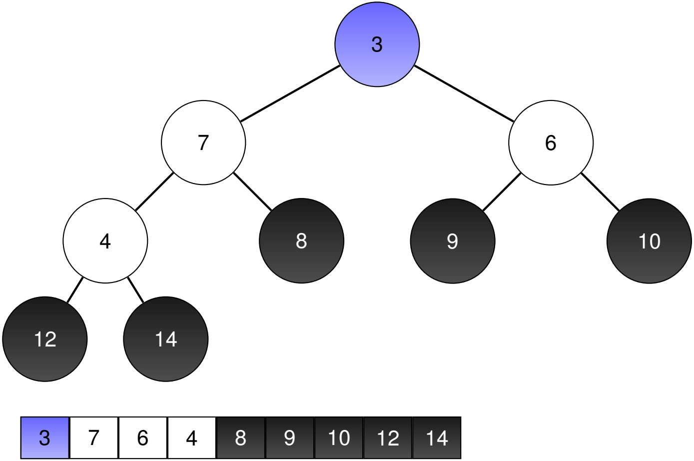
Heapsort

Scambio primo e ultimo elemento:

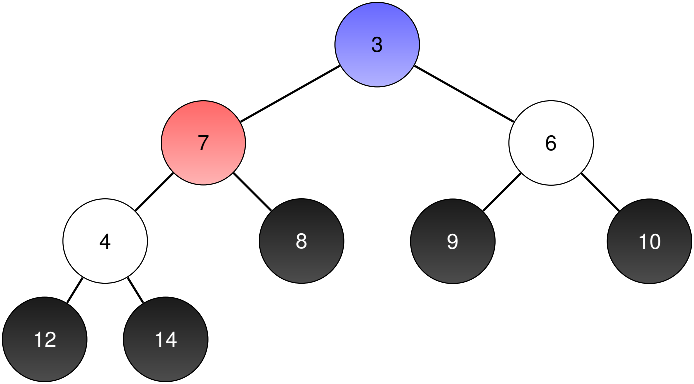


3	7	6	4	8	9	10	12	14

Ricostruzione Heap:

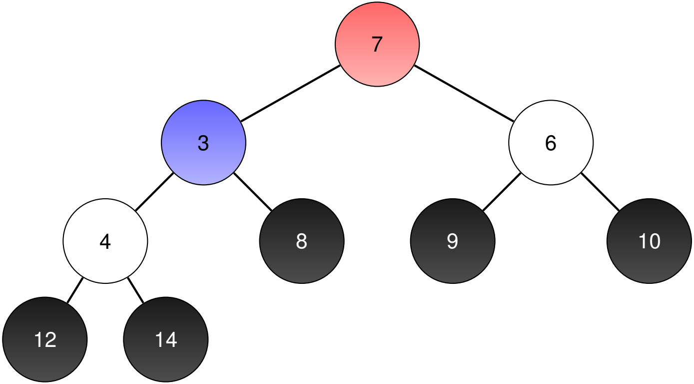


Ricostruzione Heap:



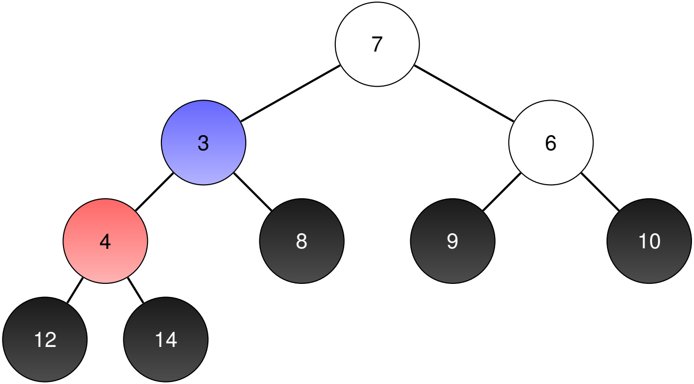
3	7	6	4	8	9	10	12	14

Ricostruzione Heap:



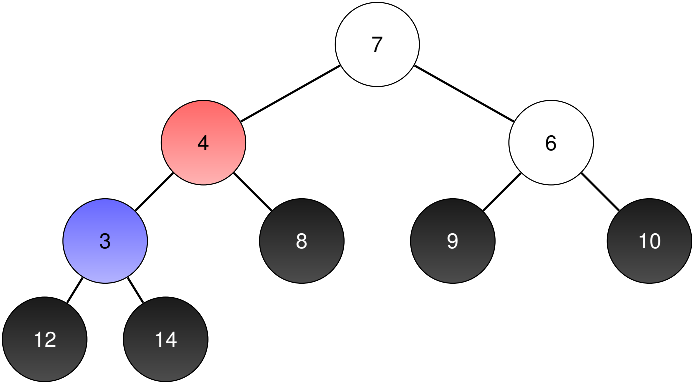
7	3	6	4	8	9	10	12	14

Ricostruzione Heap:



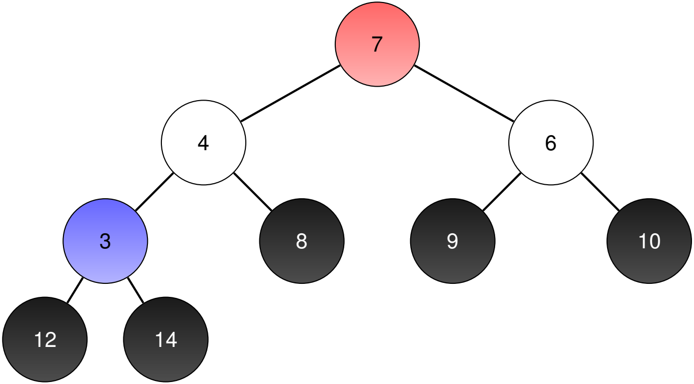
7	3	6	4	8	9	10	12	14

Ricostruzione Heap:



7	4	6	3	8	9	10	12	14

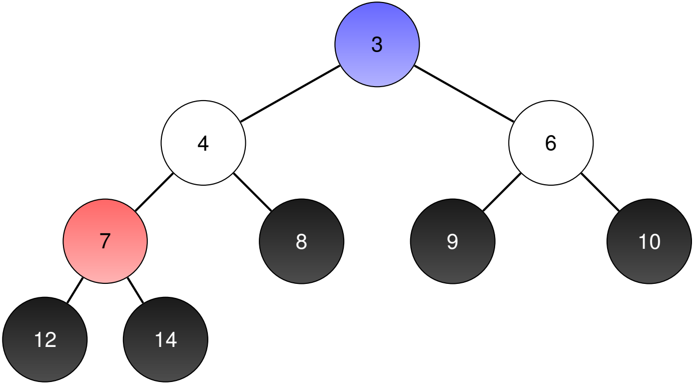
Scambio primo e ultimo elemento:



7	4	6	3	8	9	10	12	14

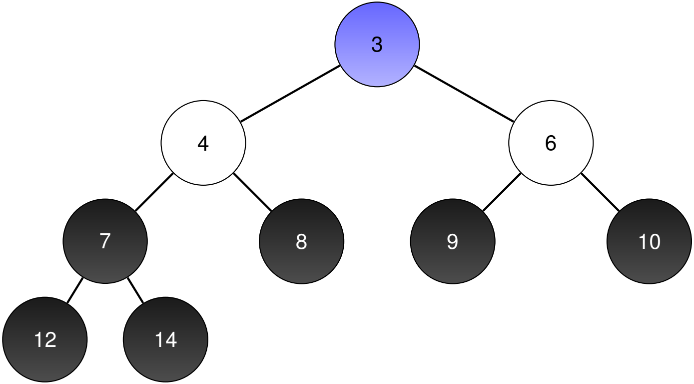
Heapsort

Scambio primo e ultimo elemento:



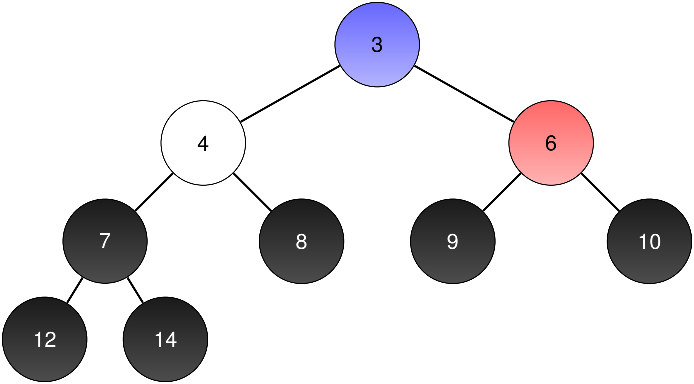
3	4	6	7	8	9	10	12	14

Ricostruzione Heap:



3	4	6	7	8	9	10	12	14

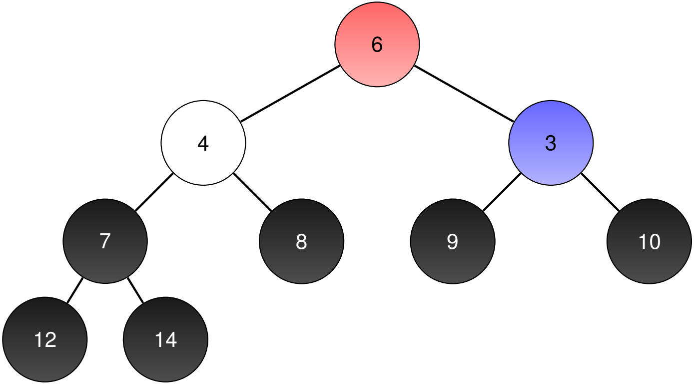
Ricostruzione Heap:



3	4	6	7	8	9	10	12	14

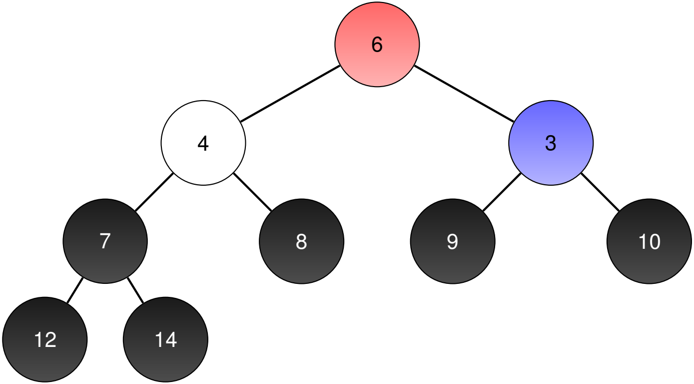
Heapsort

Ricostruzione Heap:



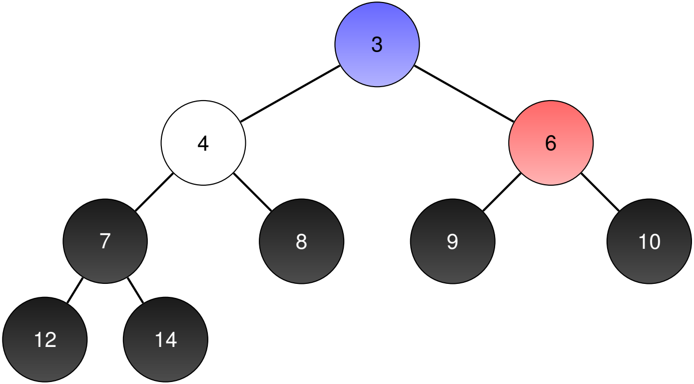
6	4	3	7	8	9	10	12	14

Scambio primo e ultimo elemento:



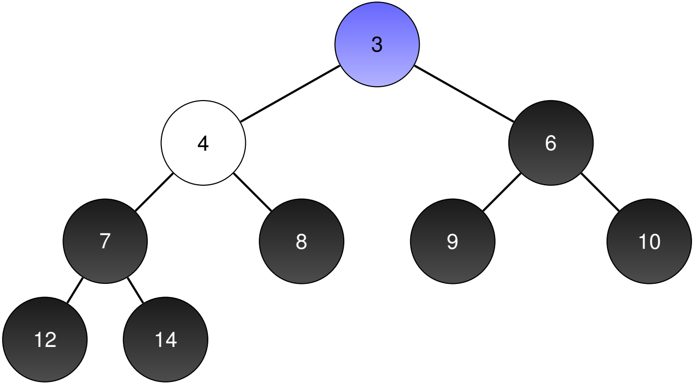
6	4	3	7	8	9	10	12	14

Scambio primo e ultimo elemento:



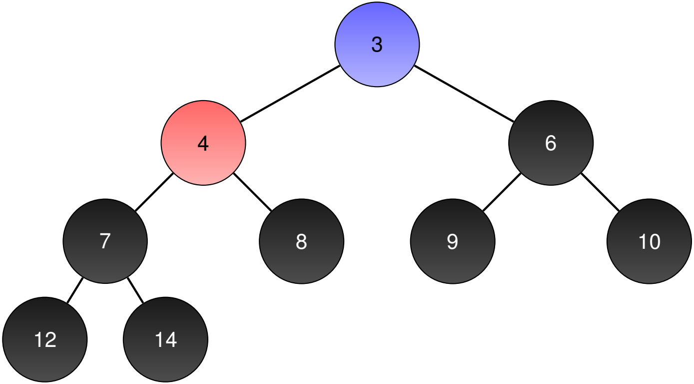
3	4	6	7	8	9	10	12	14

Ricostruzione Heap:



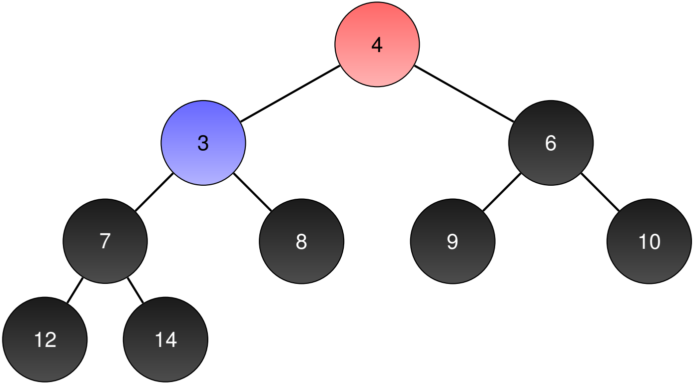
3	4	6	7	8	9	10	12	14

Ricostruzione Heap:



3	4	6	7	8	9	10	12	14

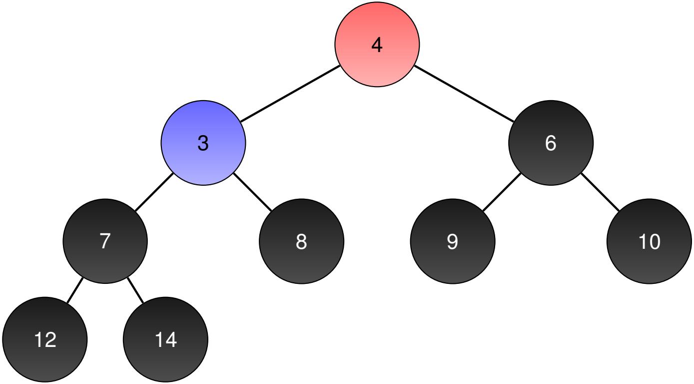
Ricostruzione Heap:



4	3	6	7	8	9	10	12	14

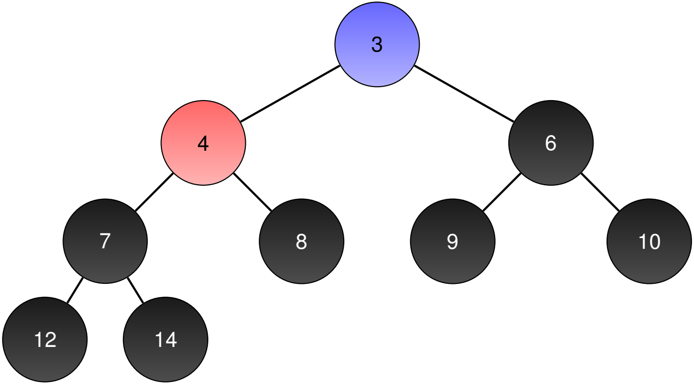
Heapsort

Scambio primo e ultimo elemento:



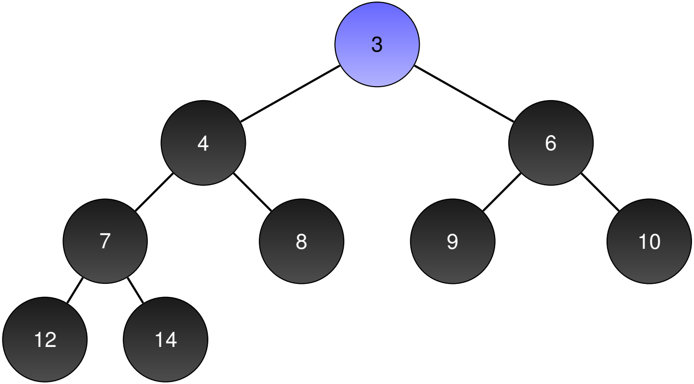
4	3	6	7	8	9	10	12	14

Scambio primo e ultimo elemento:



3	4	6	7	8	9	10	12	14

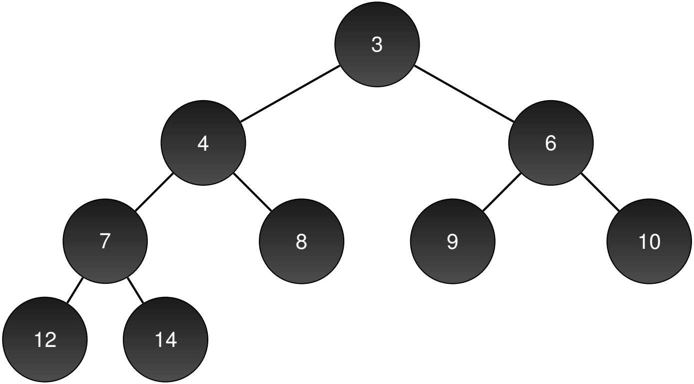
Ricostruzione Heap: 1 solo elemento $\Longrightarrow$ già heap



3	4	6	7	8	9	10	12	14

Heapsort

Elementi esauriti $\Longrightarrow$ array ordinato:



3	4	6	7	8	9	10	12	14