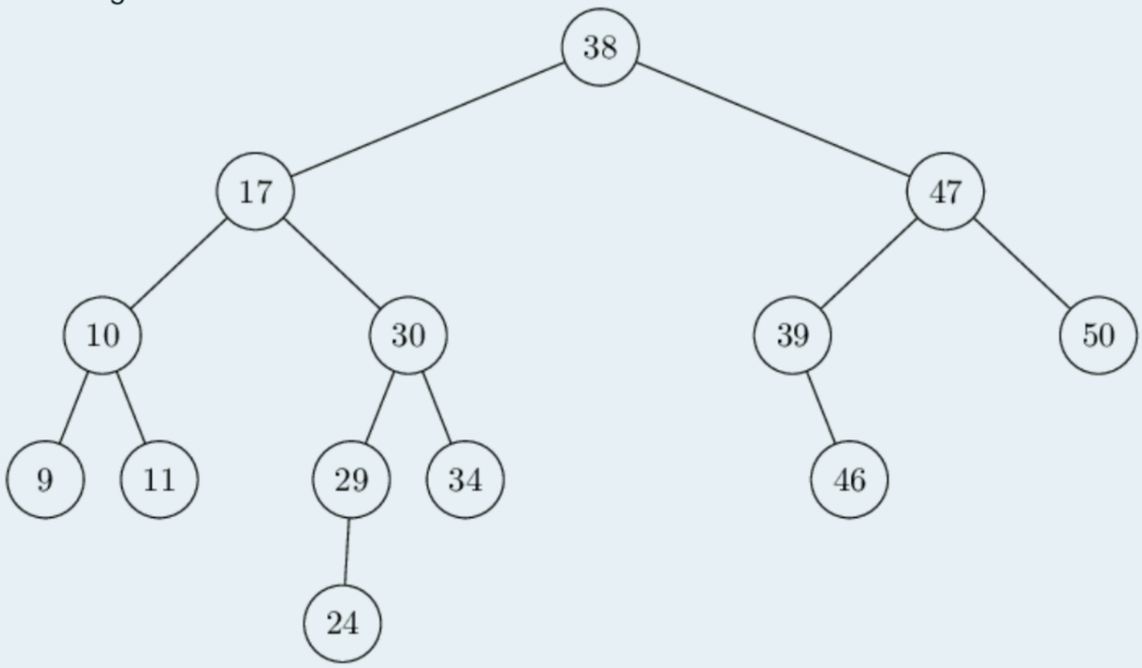
Dato il seguente albero AVL:

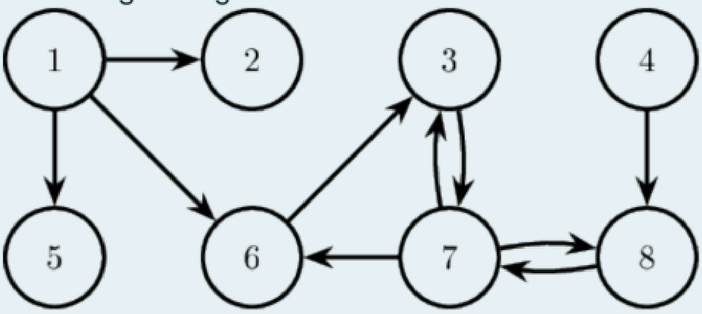


Indicare il fattore di bilanciamento del nodo 39: -1

Indicare il valore del nodo critico all'inserimento del valore 26. Se tale nodo non esiste, indicare 0: 29

A seguito del precedente inserimento, inserire in sequenza gli ulteriori valori 16, 8 e 14, tenendo conto delle possibili rotazioni. Indicare l'altezza del nodo 10 a seguito di tutti gli inserimenti: 2

Dato il seguente grafo orientato G:



E il seguente algoritmo di visita in profondita':

DFS(G, s)

```
for each u ∈ V(G) do
        color[u] ← WHITE
        π[u] ← nil
time ← 0
DFS-Visit(s)
```

DFS-Visit(u)

```
color[u] ← GRAY
time ← time + 1
d[u] ← time
for each v ∈ Adj[u] in ordine crescente di chiave do
        if color[v] = WHITE then
                π[v] ← u
                DFS-Visit(v)
color[u] ← BLACK
time ← time + 1
f[u] ← time
```

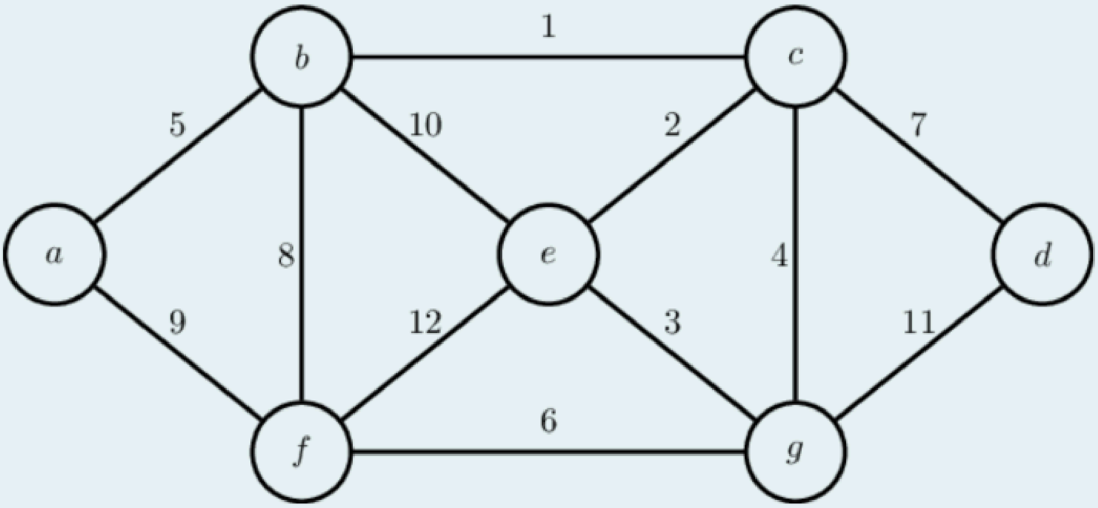
Si consideri la visita ottenuta dalla chiamata DFS$(G, 1)$ supponendo che le liste Adj di ciascun nodo siano in ordine crescente di chiave.

Indicare la chiave del quarto nodo ad assumere il colore GRAY durante la visita: 6

✔

Indicare il valore di $f[7]$ al termine della visita: 11 ✔

Dato il seguente grafo non orientato G:



Si consideri l'applicazione dell'algoritmo di Dijkstra a G con nodo sorgente $s = d$.

Dijkstra(G, w, s)

1. **foreach** $v \in V(G)$ **do**
2. $\quad d[v] \leftarrow \infty$
3. $\quad \pi[v] \leftarrow$ **nil**
4. $d[s] \leftarrow 0$
5. $Q \leftarrow V(G)$
6. **while** $Q \neq \emptyset$ **do**
7. $\quad u \leftarrow$ **Extract-Min**(Q)
8. $\quad S \leftarrow S \cup \{u\}$
9. $\quad\quad$ **foreach** $v \in Adj[u]$ **do**
10. $\quad\quad\quad$ **if** $d[v] > d[u] + w(u, v)$ **then**
11. $\quad\quad\quad d[v] \leftarrow d[u] + w(u, v)$
12. $\quad\quad\quad \pi[v] \leftarrow u$

Riempire la tabella sottostante con i valori di d e π per ogni nodo del grafo nel momento in cui il nodo g è estratto dalla coda Q (Riga 7). Usare 999 per indicare ∞.

a		b		c		d		e	
π	d	π	d	π	d	π	d	π	
b	13	c	8	d	7	nil	0	c	9
✓	✓	✓	✓	✓	✓	✓	✓	✓	

Dato l'algoritmo HeapSort descritto di seguito

HeapSort(A)
1. **Build-Max-Heap**(A)
2. **for** $i \leftarrow length[A]$ **downto** 2 **do**
3. scambia $A[1] \leftrightarrow A[i]$
4. $heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$
5. **Max-Heapify**$(A, 1)$

Considerare l'applicazione di HeapSort al seguente array A:

$A =$

28	4	31	24	37	36	13	33	27	32	23

Indicare il valore del figlio sinistro del nodo con valore 24 nella rappresentazione ad albero di A prima dell'esecuzione di Build-Max-Heap(A) a riga 1: 33 ✔

Indicare l'indice nell'array del valore 37 dopo l'esecuzione di Build-Max-Heap(A) a riga 1 (Gli indici partono da 1): 1 ✔

Indicare il valore alla radice dell'heap nel momento in cui $heap\text{-}size[A]$ viene assegnato a 9 (Prima della successiva chiamata a Max-Heapify): 4 ✔

Dato il seguente insieme non ordinato di funzioni in n

$$\{\ \log_2 \log_2 \log_2 n,\quad n \log_2 n + 2,\quad (\log_2 n)^3,\quad 2^n,\quad n! + 1,\quad n^2,\quad \sqrt[3]{\log_2 n}\ \}$$

Assegnare ciascuna funzione ad un indice diverso $i \in \{1, \ldots, 7\}$ affinché sia valido il seguente ordinamento:

$$f_1(n) \leq f_2(n) \leq f_3(n) \leq f_4(n) \leq f_5(n) \leq f_6(n) \leq f_7(n)$$

dove $f_i(n)$ è la funzione assegnata all'indice i, e $\leq$ è la relazione d'ordine basata sul tasso di crescita delle funzioni.

2^n	6 ✔
$n! + 1$	7 ✔
$(\log_2 n)^3$	3 ✔
$\sqrt[3]{\log_2 n}$	2 ✔
n^2	5 ✔
$\log_2 \log_2 \log_2 n$	1 ✔
$n \log_2 n + 2$	4 ✔

La risposta corretta è: 2^n → 6, $n! + 1$ → 7, $(\log_2 n)^3$ → 3, $\sqrt[3]{\log_2 n}$ → 2, n^2 → 5, $\log_2 \log_2 \log_2 n$ → 1, $n \log_2 n + 2$ → 4