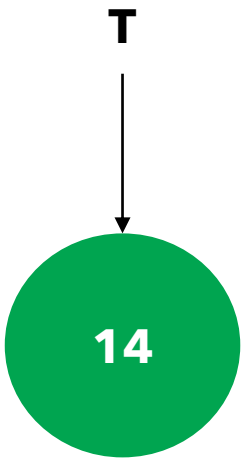
Alberi binari

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

BST – inserimento ricorsivo

```
T = nil
bst_insert(T, 14)
```

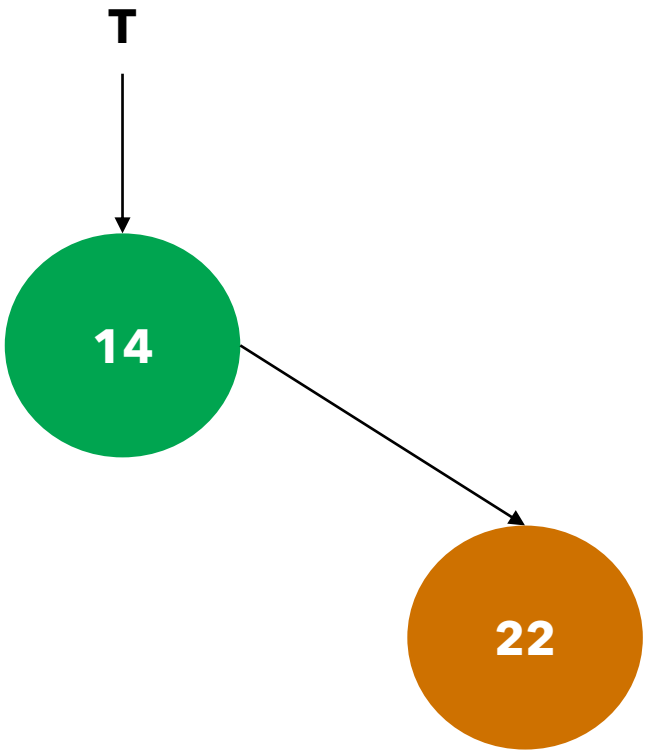
caso base: si crea un nuovo nodo e lo si fa puntare da T



BST – inserimento ricorsivo

`bst_insert(T, 22)`

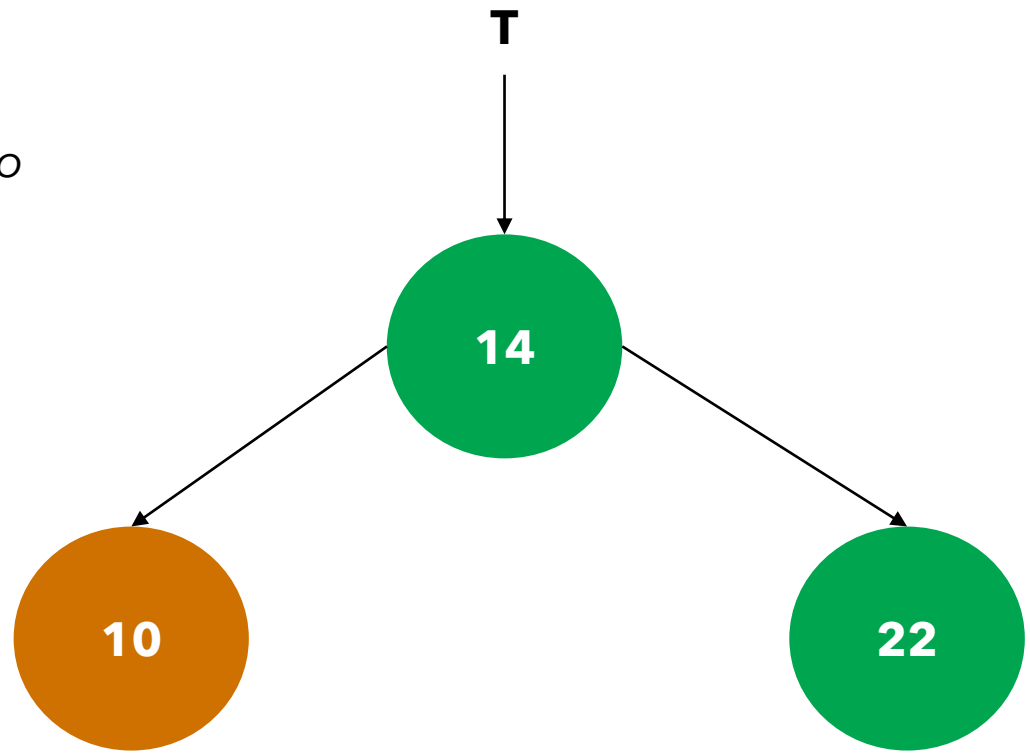
caso ricorsivo: si inserisce un nuovo nodo *al posto giusto* (come foglia) all'interno del sottoalbero `T.right`



BST – inserimento ricorsivo

`bst_insert(T, 10)`

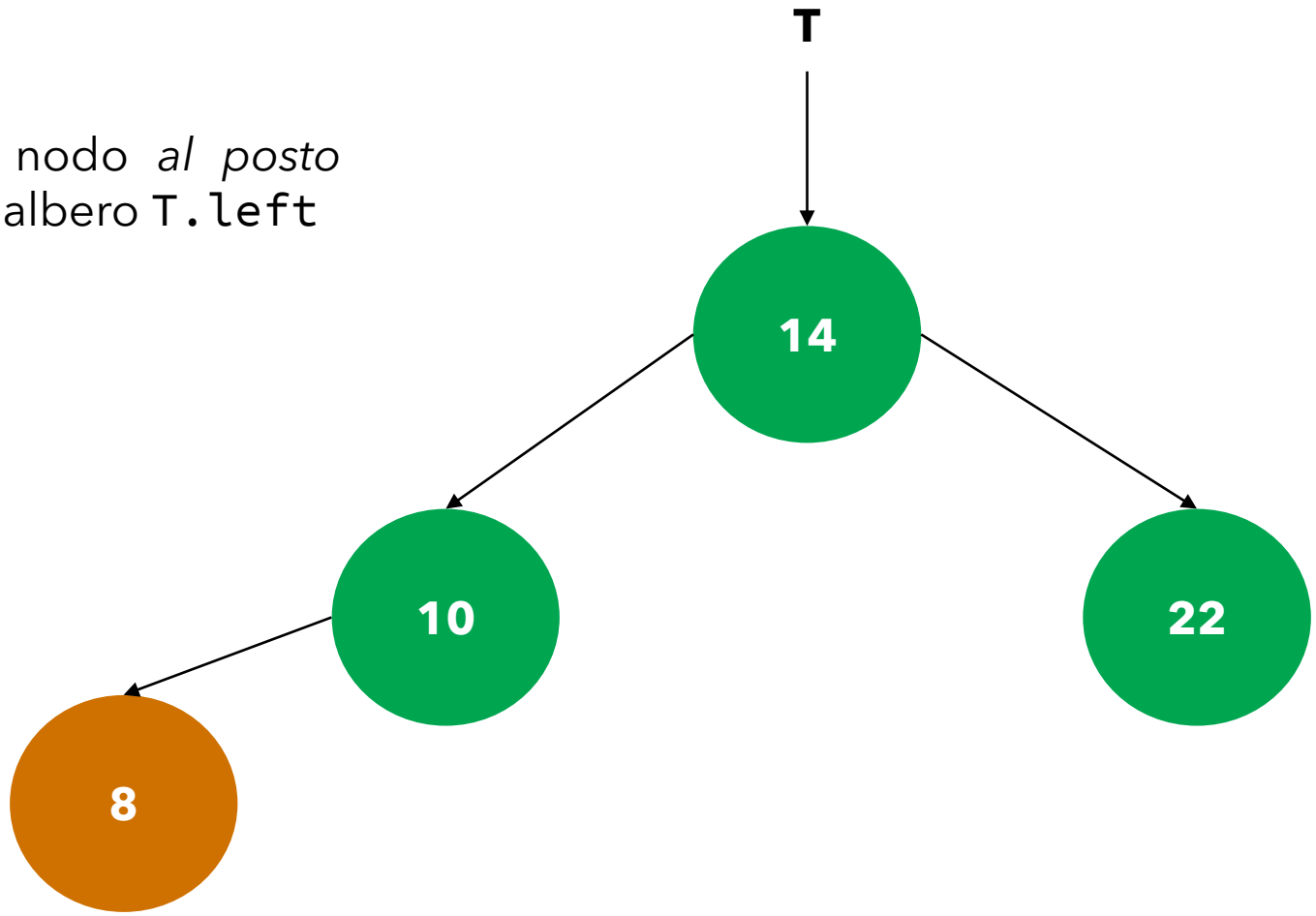
caso ricorsivo: si inserisce un nuovo nodo *al posto giusto* (come foglia) all'interno del sottoalbero `T.left`



BST – inserimento ricorsivo

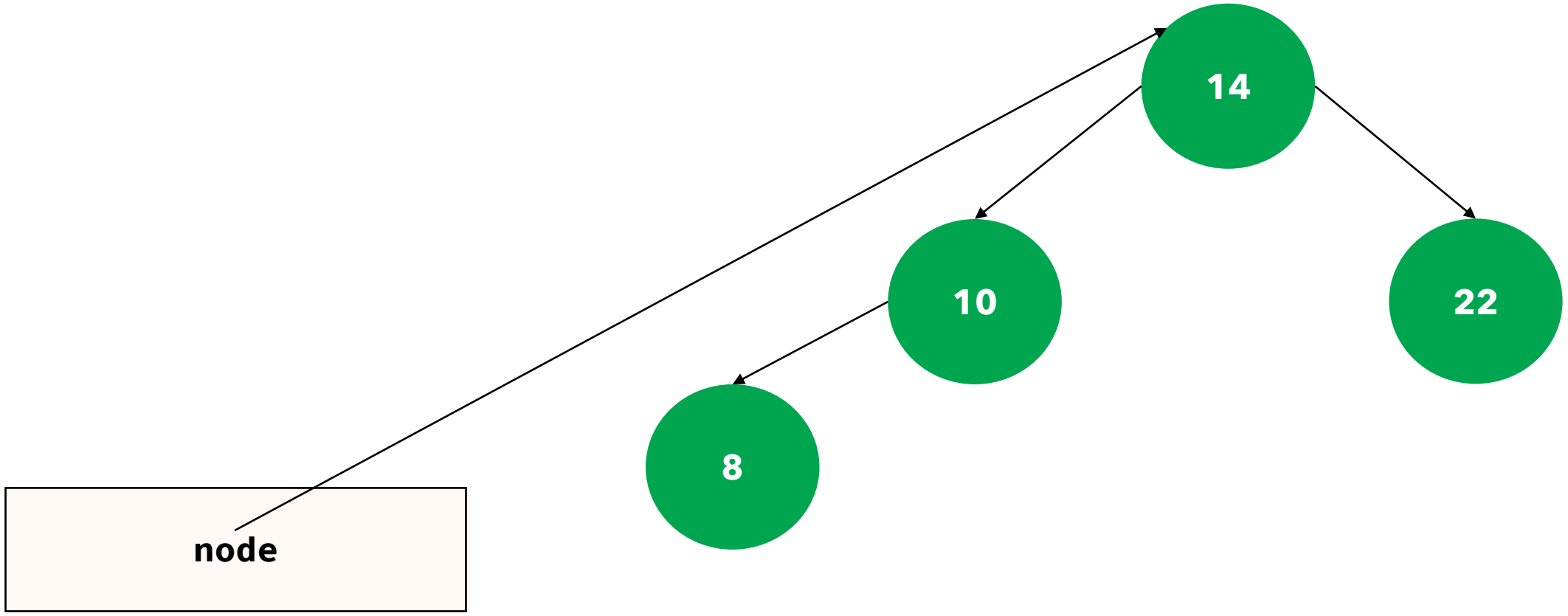
`bst_insert(T, 8)`

caso ricorsivo: si inserisce un nuovo nodo *al posto giusto* (come foglia) all'interno del sottoalbero `T.left`



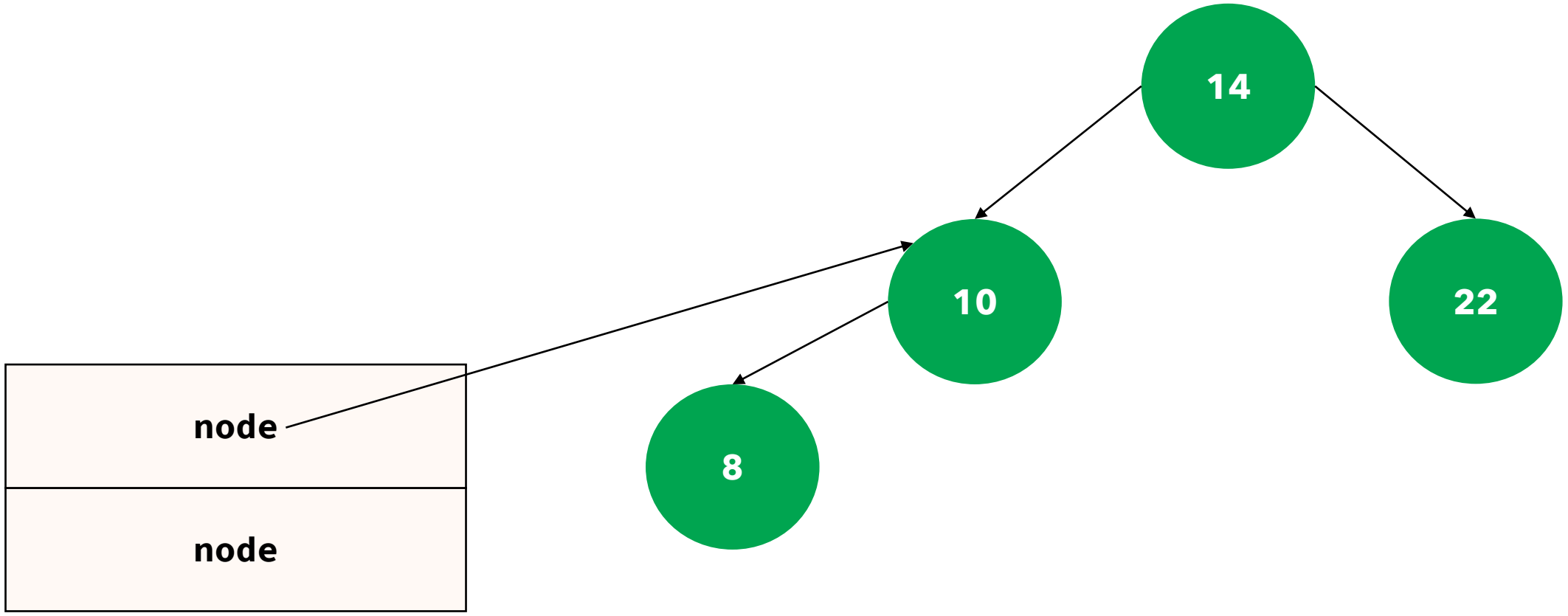
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



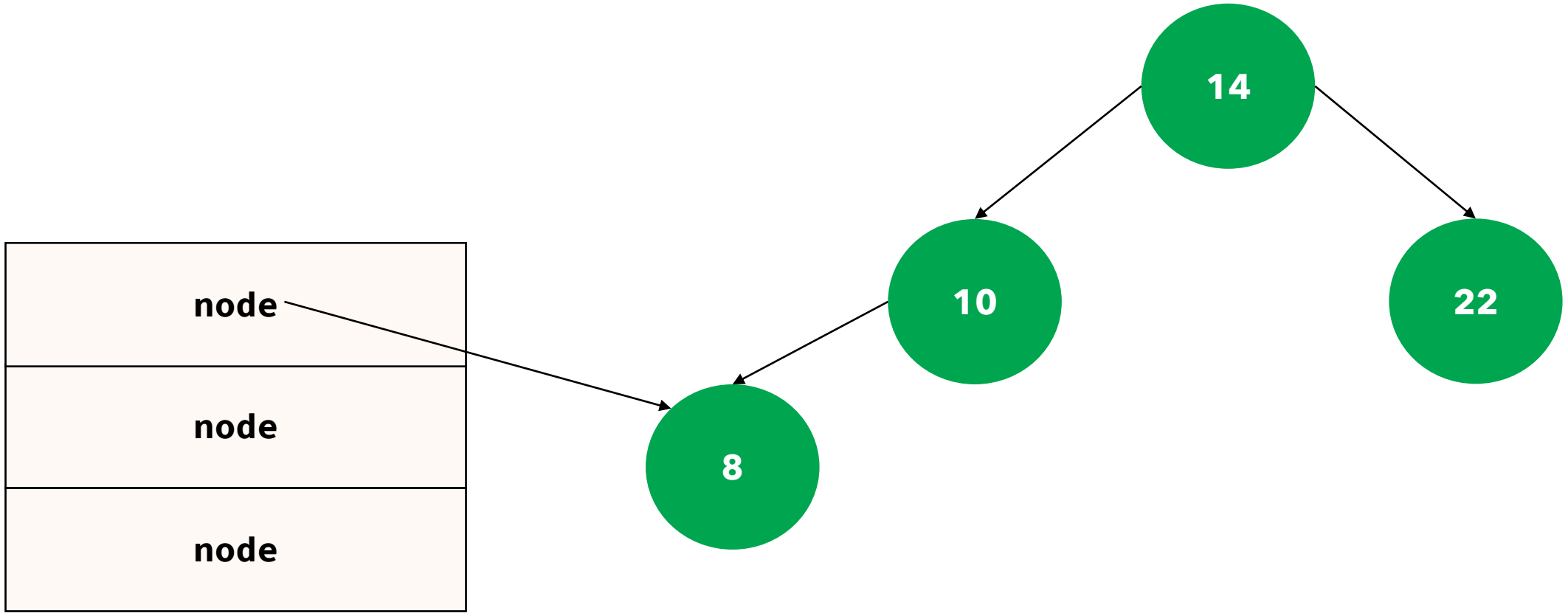
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



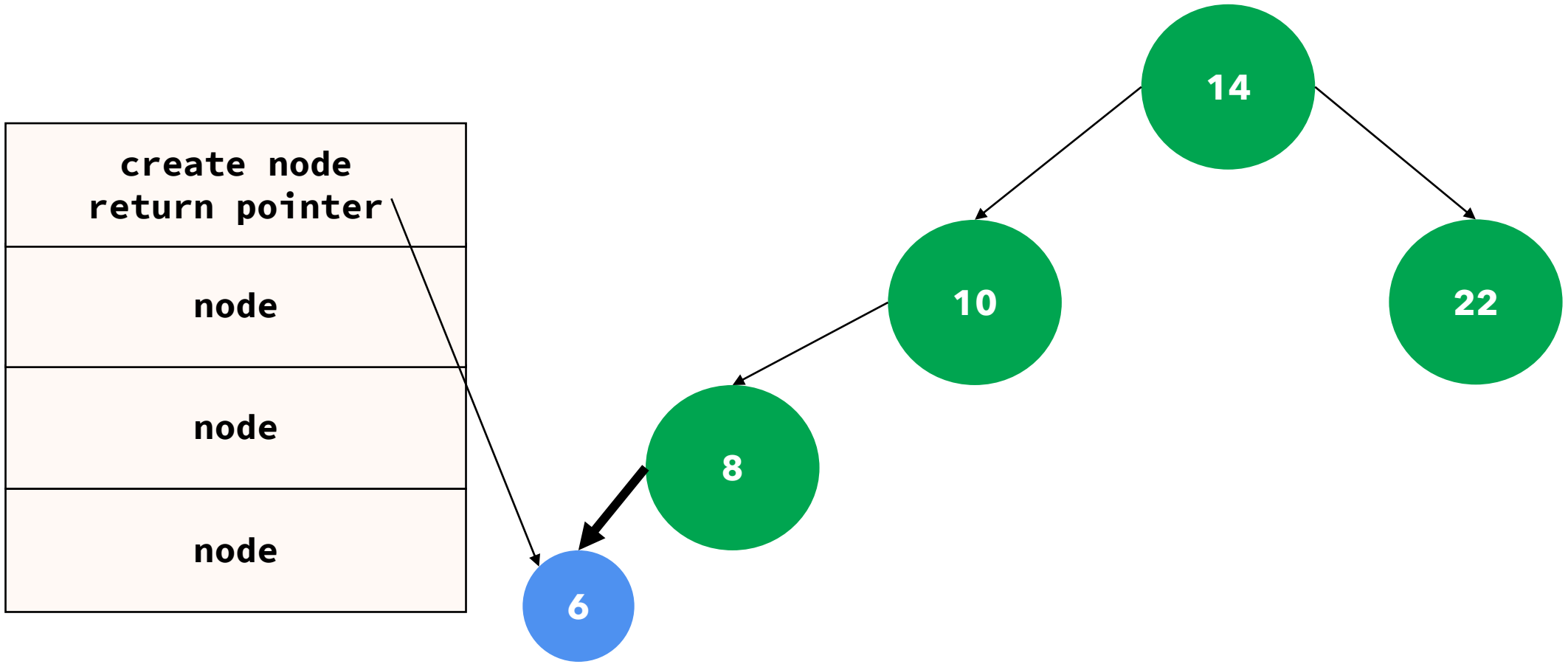
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



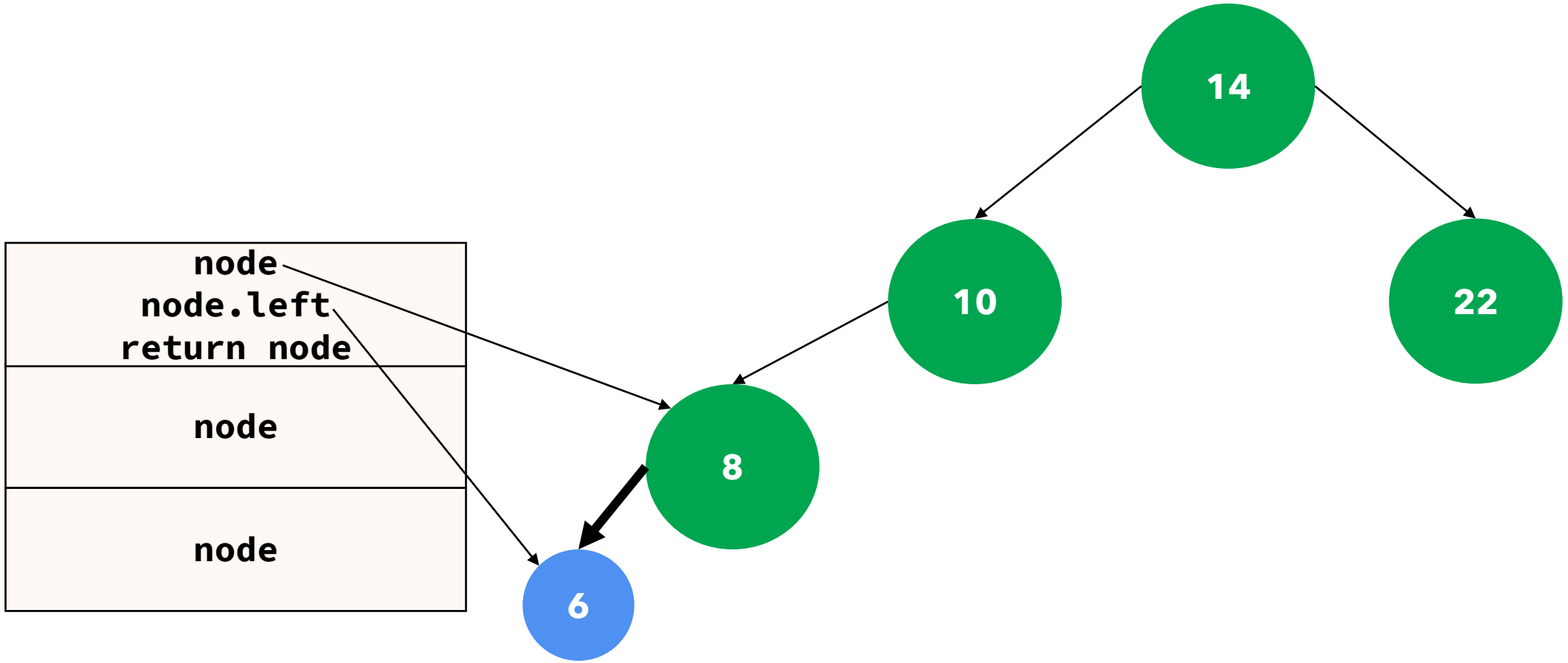
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



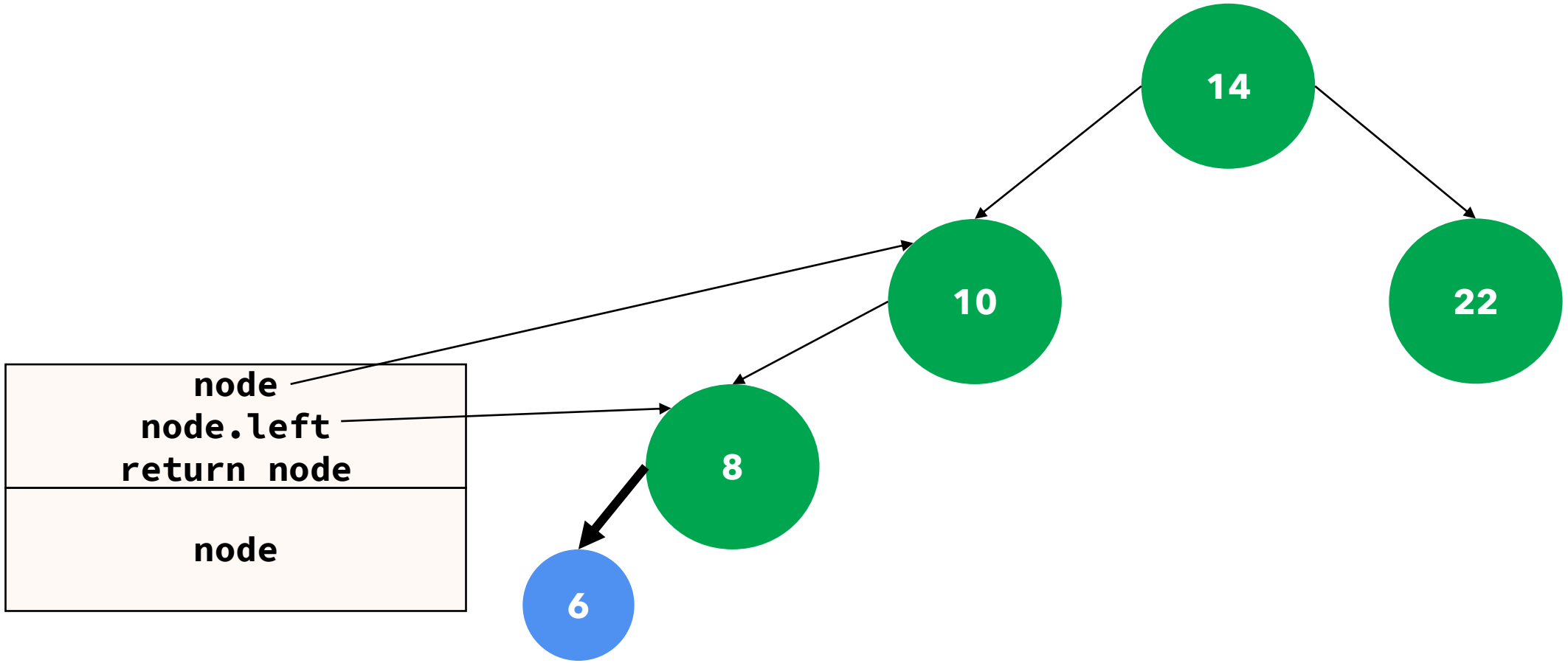
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



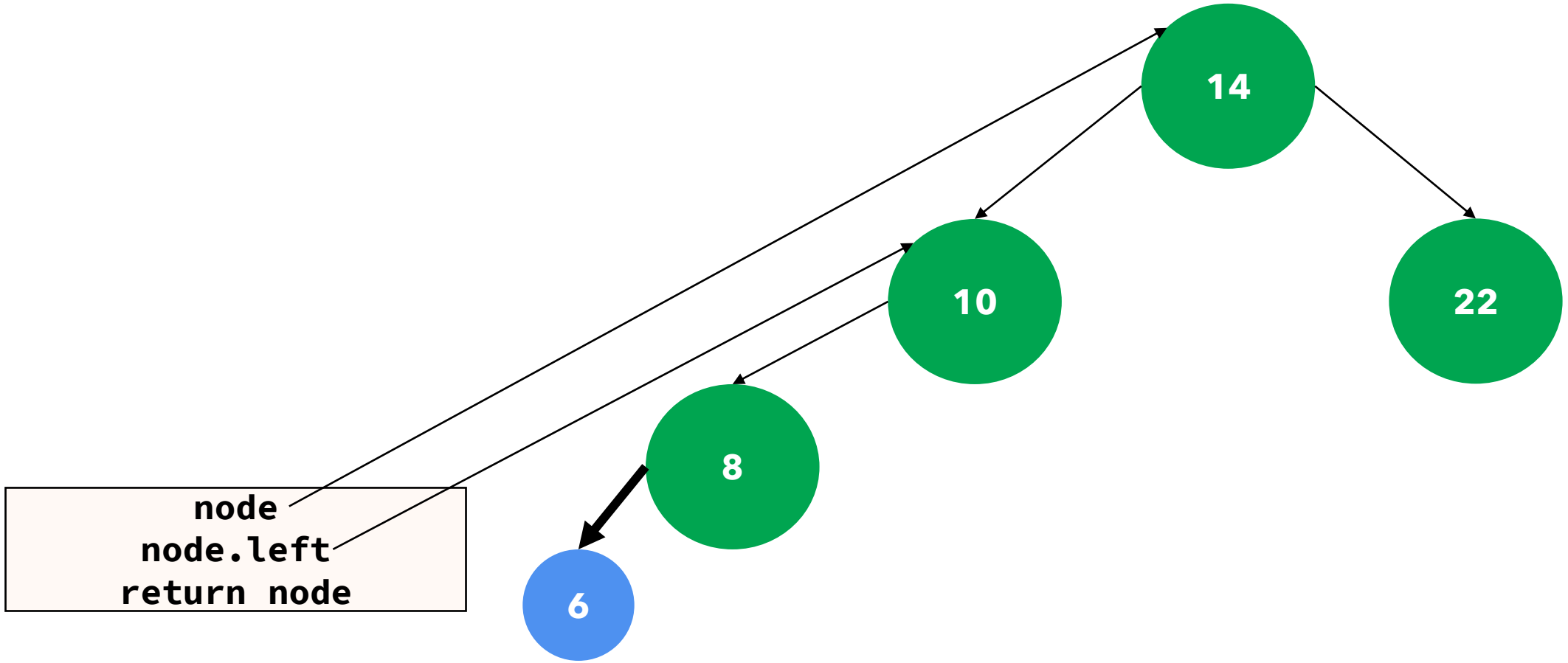
BST – inserimento ricorsivo

```
bst_insert(node, 6)
```



BST – inserimento ricorsivo

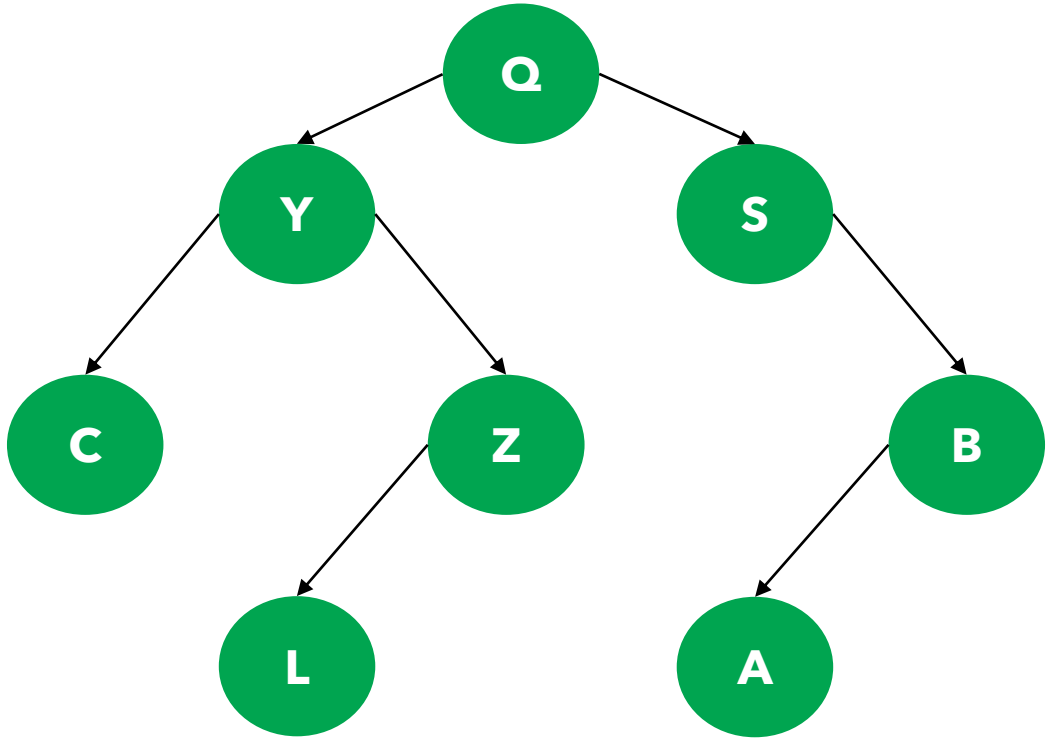
```
bst_insert(node, 6)
```



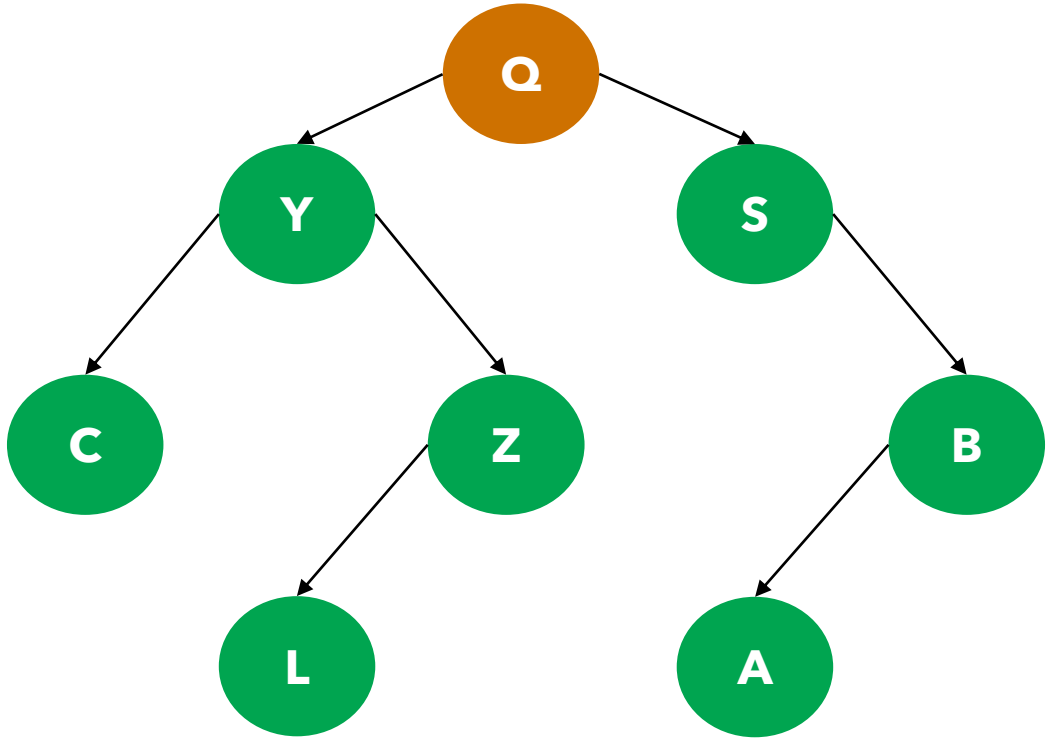
Depth-first search (DFS)

- Nella ricerca in profondità (**depth-first**), per ogni nodo vengono visitati ricorsivamente in profondità prima il sottoalbero sinistro e poi il sottoalbero destro
- Questo significa che prima di *tornare indietro* ad un nodo T, uno dei due sottoalberi di T deve essere visitato completamente
- Per tornare indietro fino al punto giusto e non ripetere gli stessi percorsi serve uno **stack**. Ovviamente, la ricorsione ci aiuta molto, perché si basa sul call stack del programma
- Esistono diverse versioni della ricerca in profondità:
  - visita **preorder**
  - visita **inorder**
  - visita **postorder**

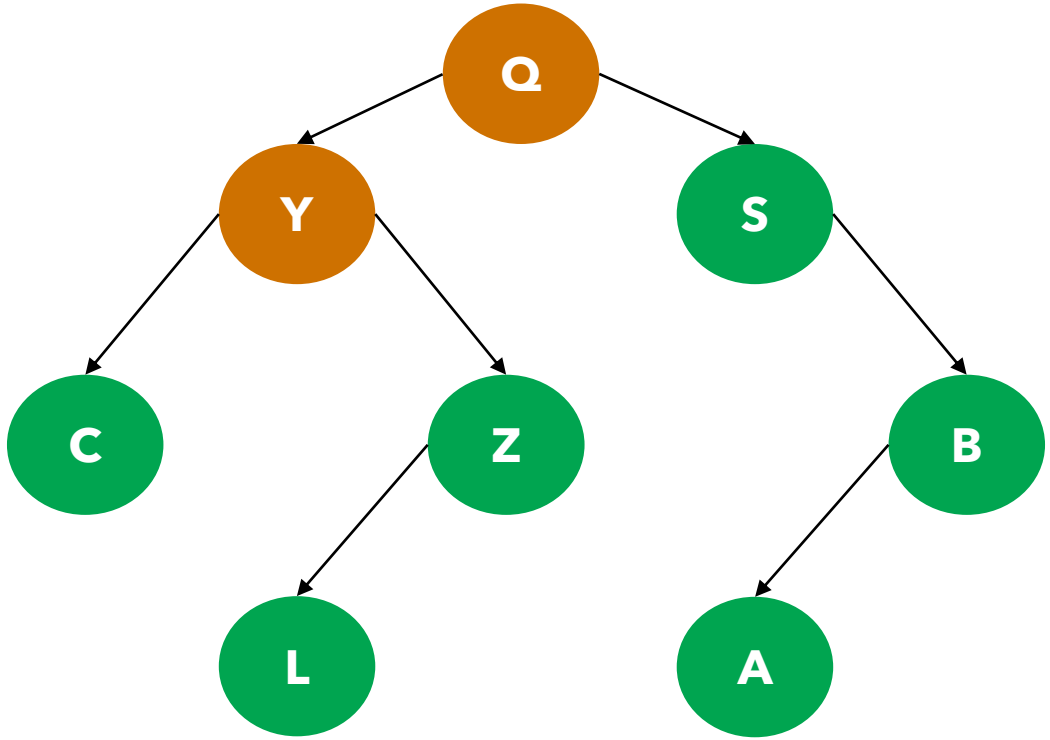
Depth-first search (DFS)



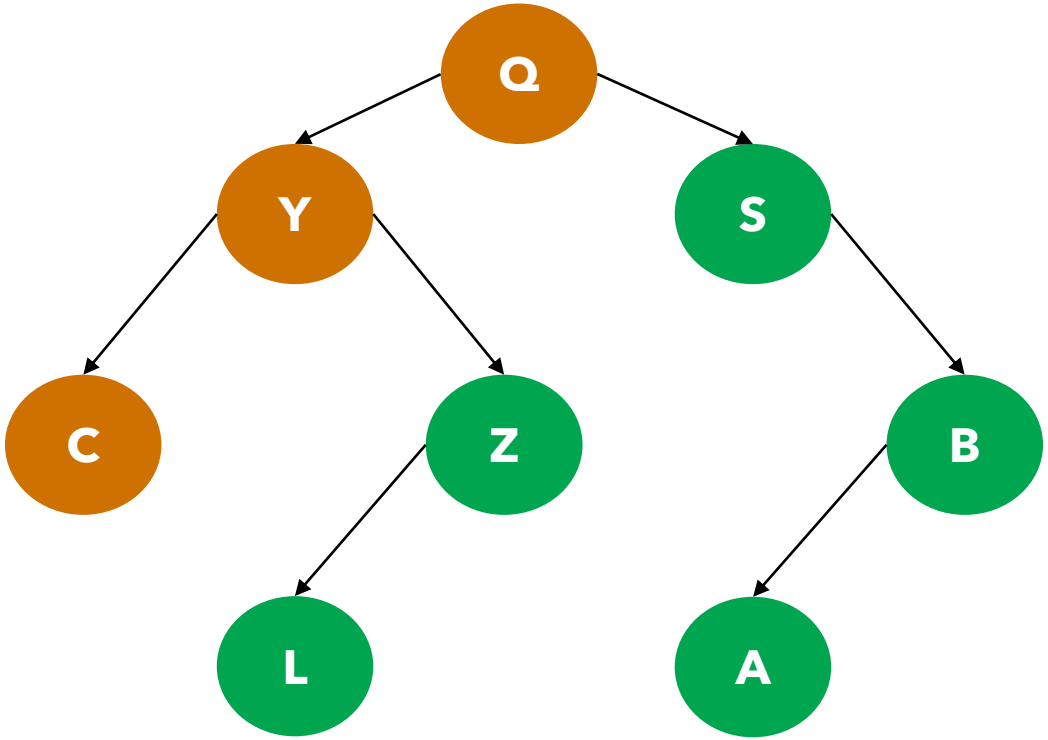
Depth-first search (DFS)



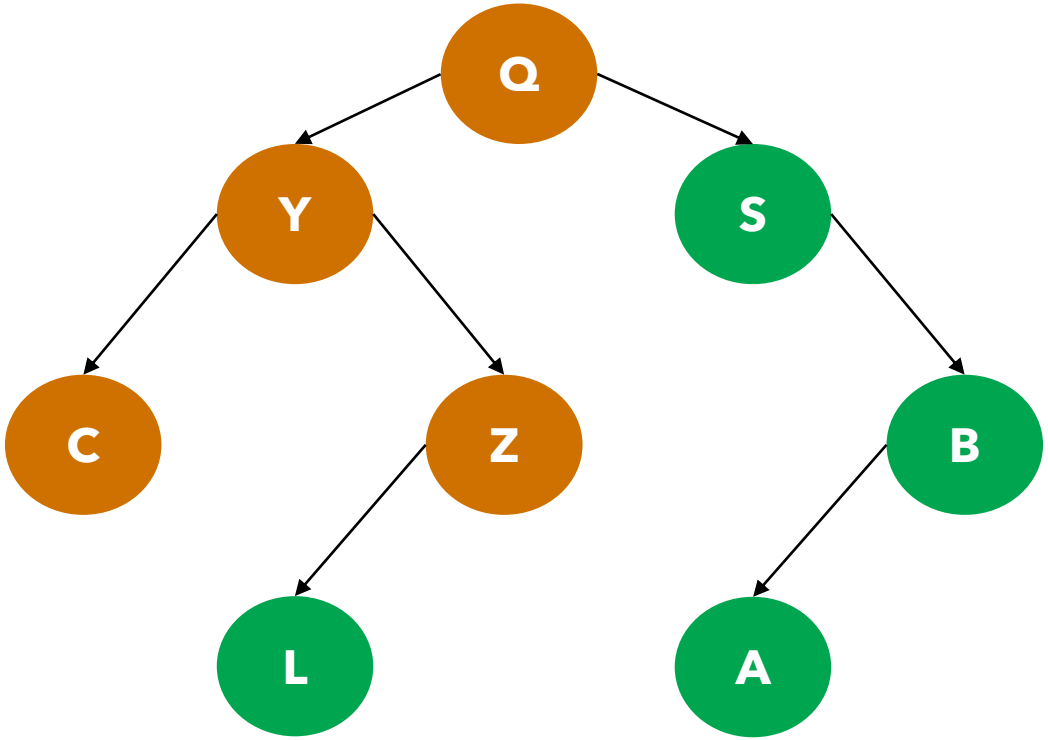
Depth-first search (DFS)



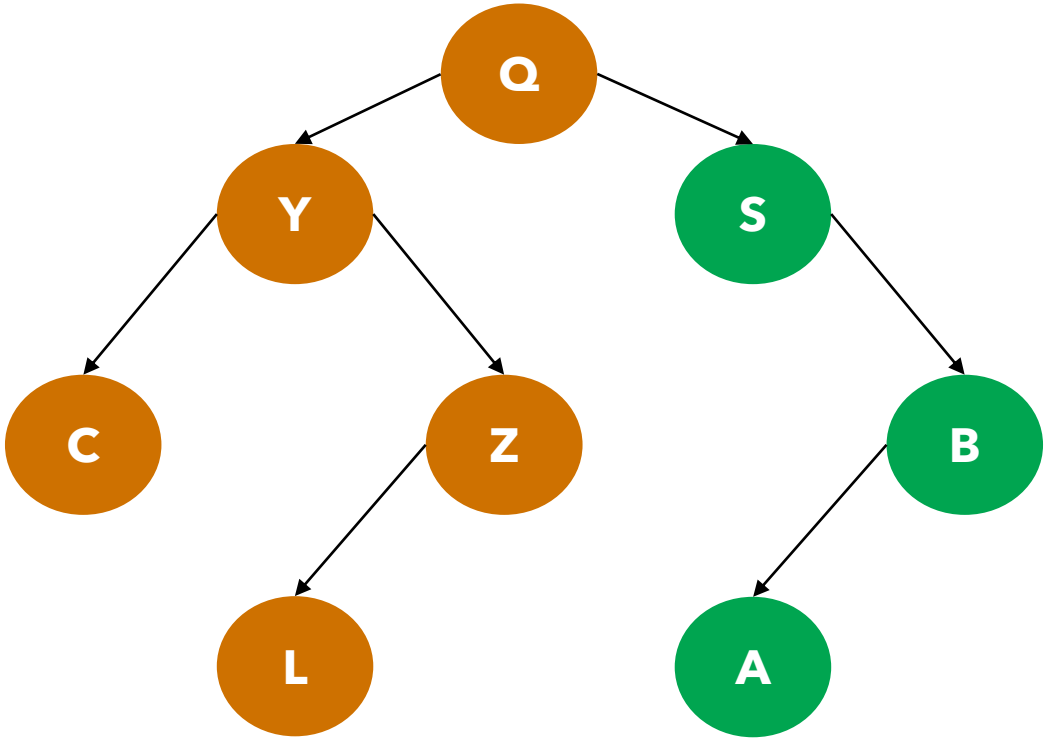
Depth-first search (DFS)



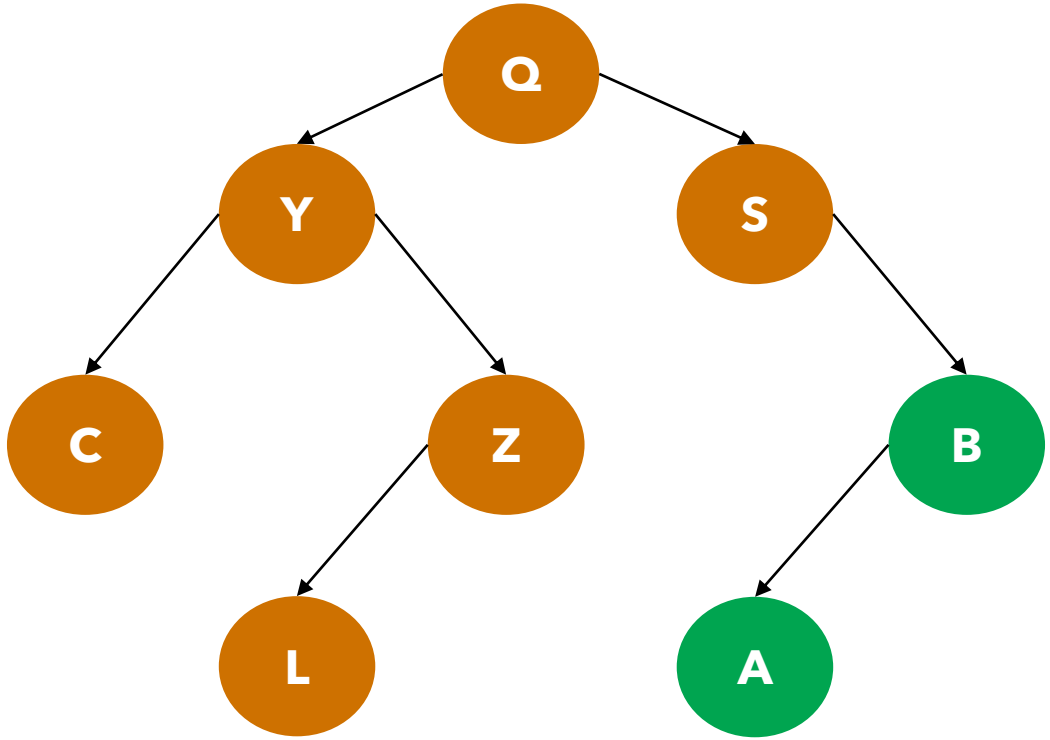
Depth-first search (DFS)



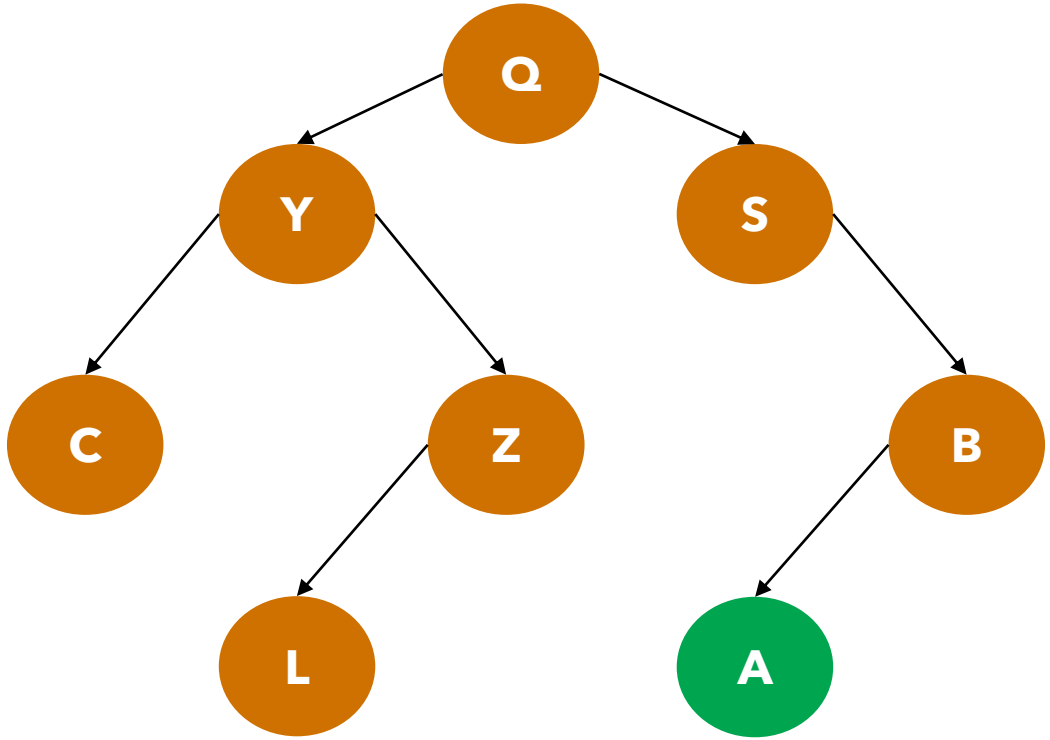
Depth-first search (DFS)



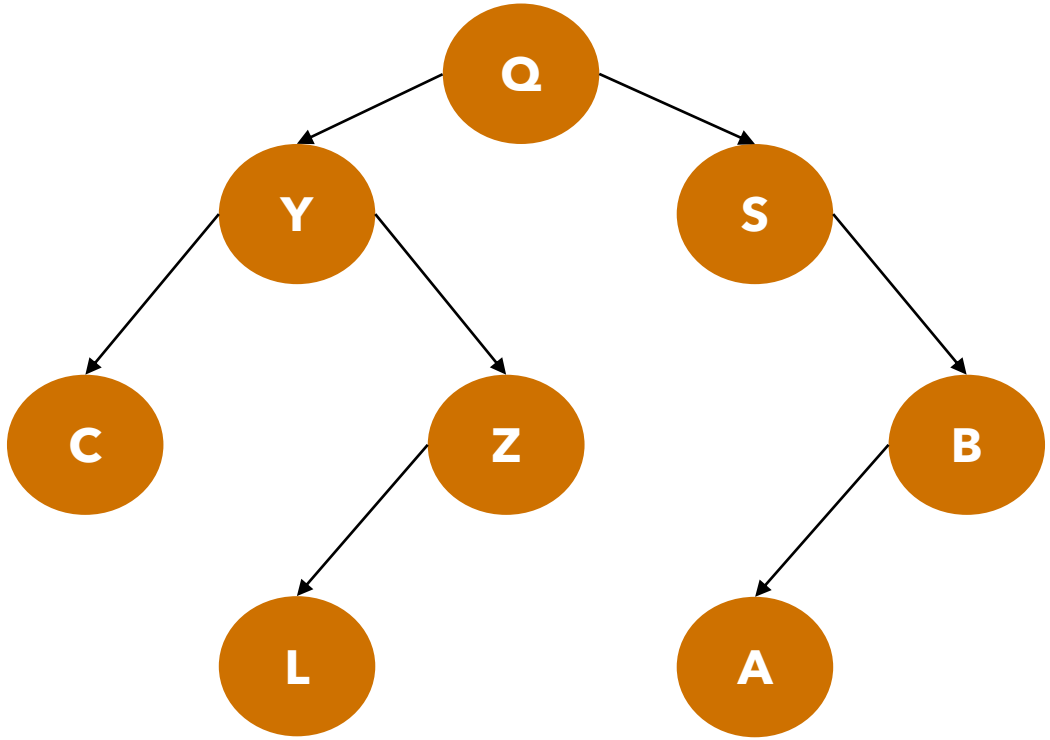
Depth-first search (DFS)



Depth-first search (DFS)



Depth-first search (DFS)



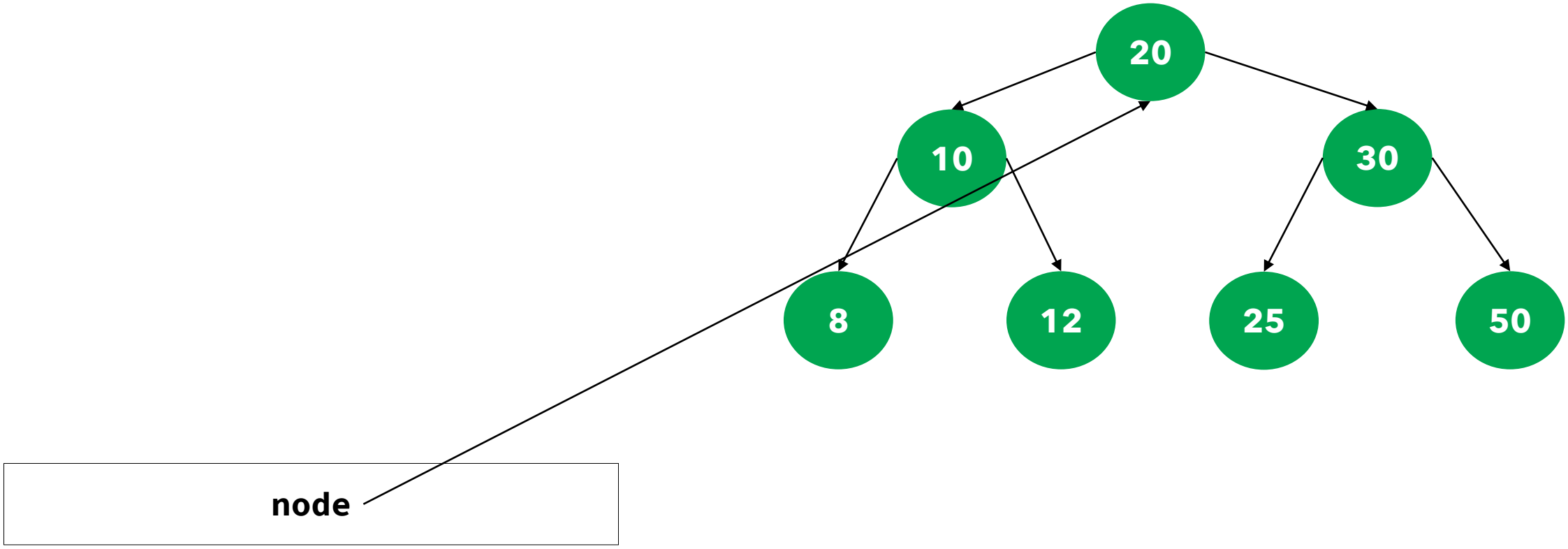
Depth-first search (DFS): inorder

`inorder_tree_walk(T):`

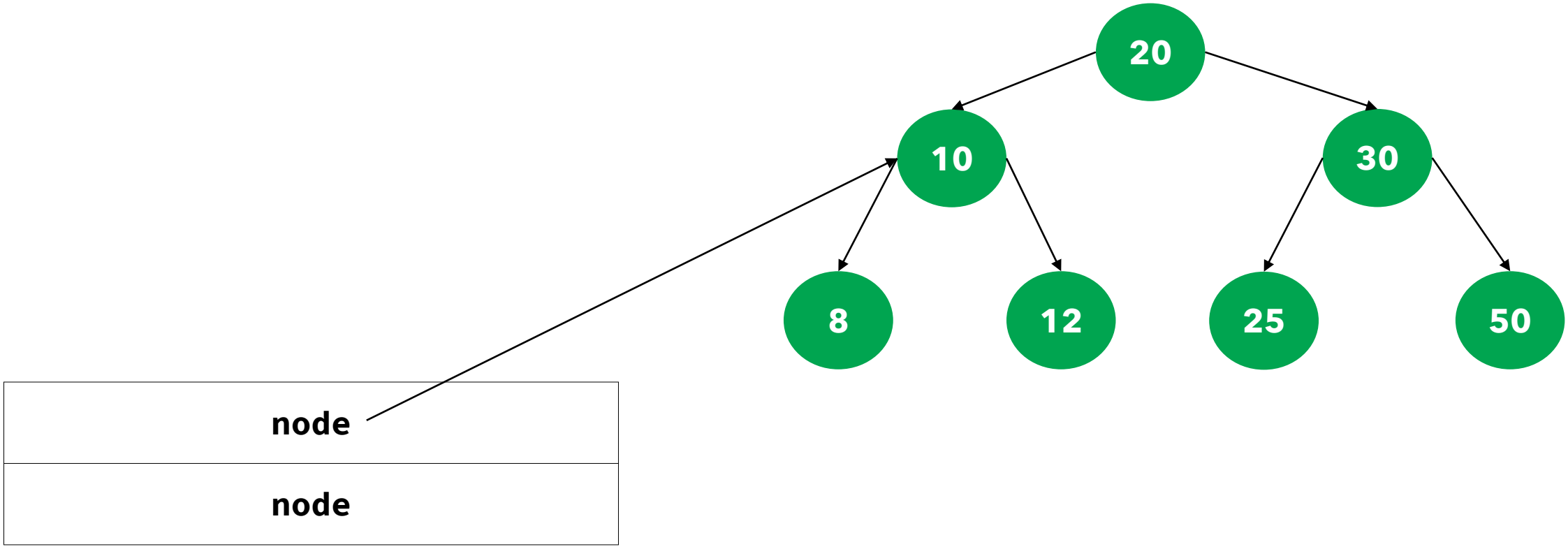
- `if T is empty, do nothing`
- `otherwise:`
  - `call inorder_tree_walk on T.left`
  - *`open`* `T (e.g.: print T.key)`
  - `call inorder_tree_walk on T.right`

- **NB:** su un BST stampa le chiavi in ordine ascendente

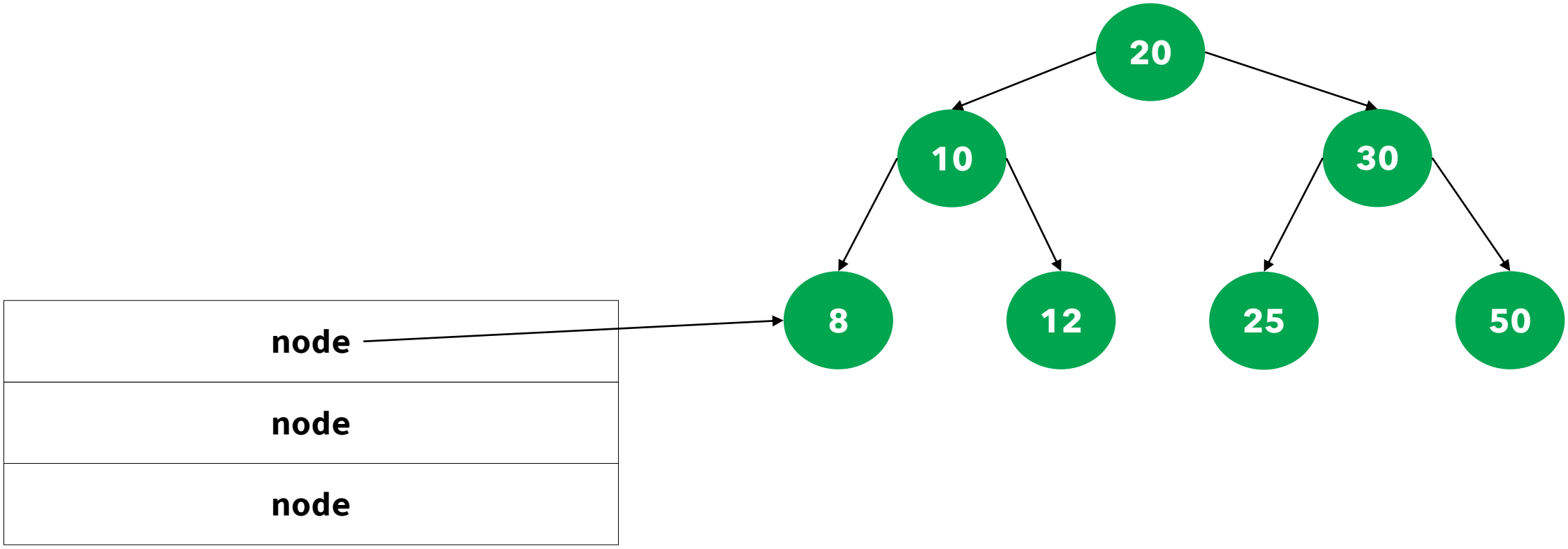
Depth-first search (DFS): inorder



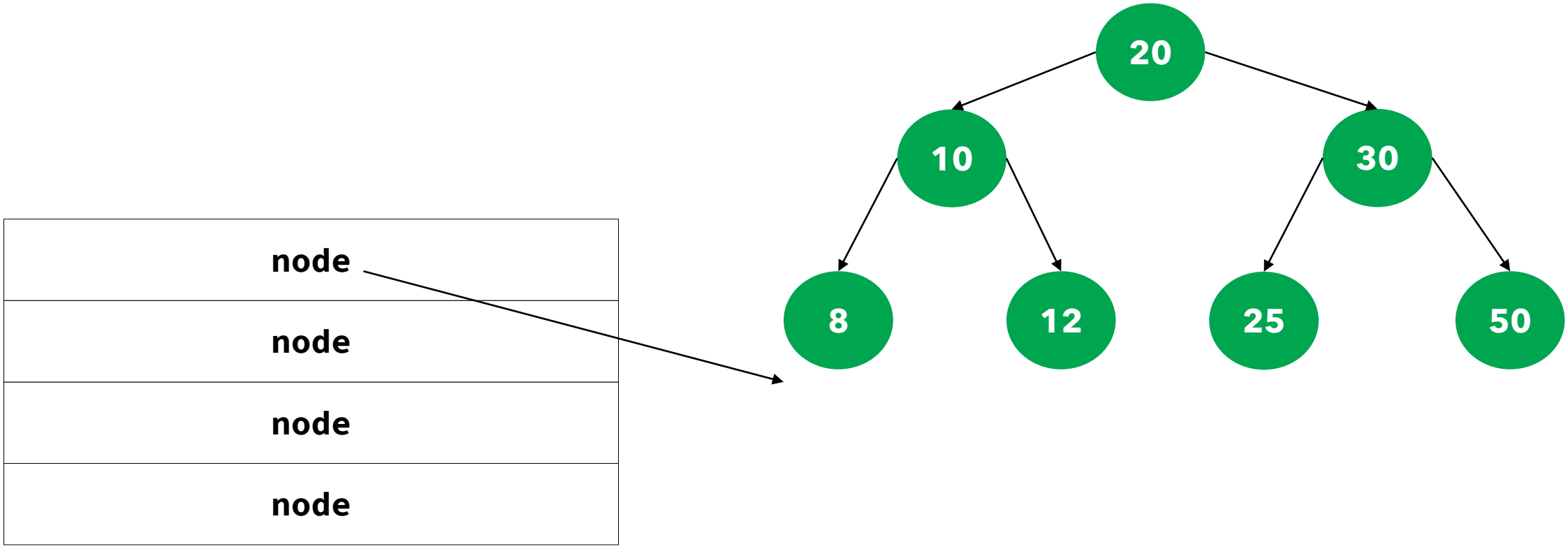
Depth-first search (DFS): inorder



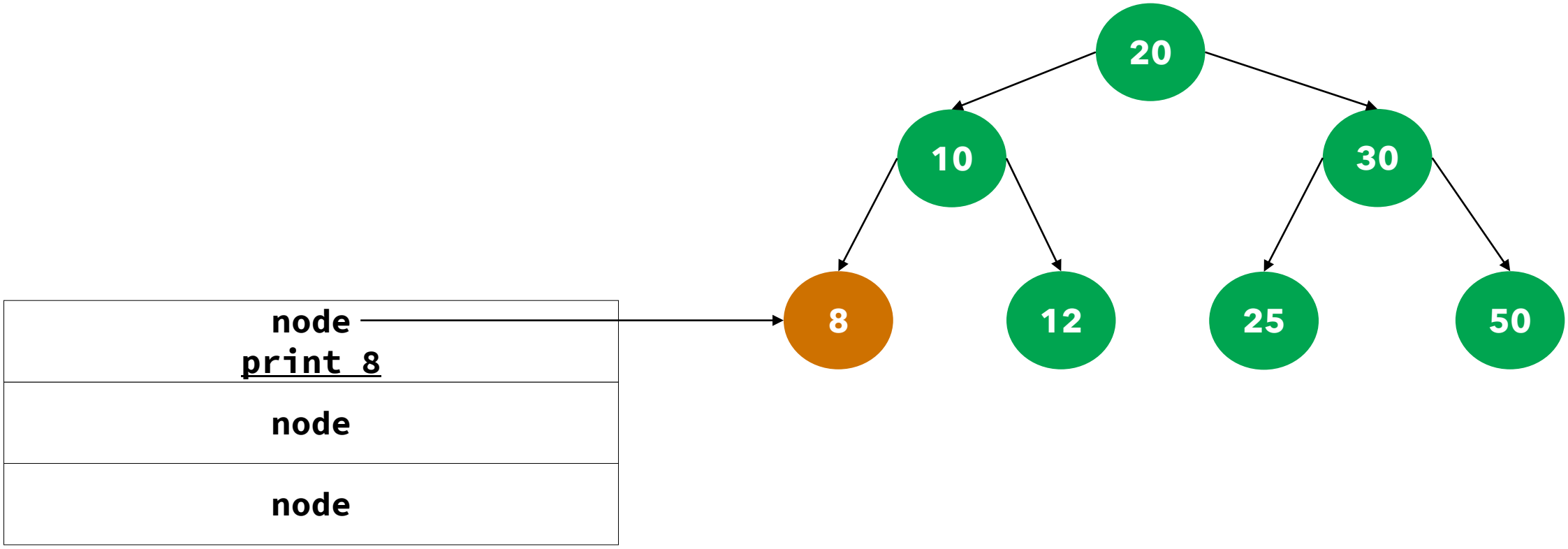
Depth-first search (DFS): inorder



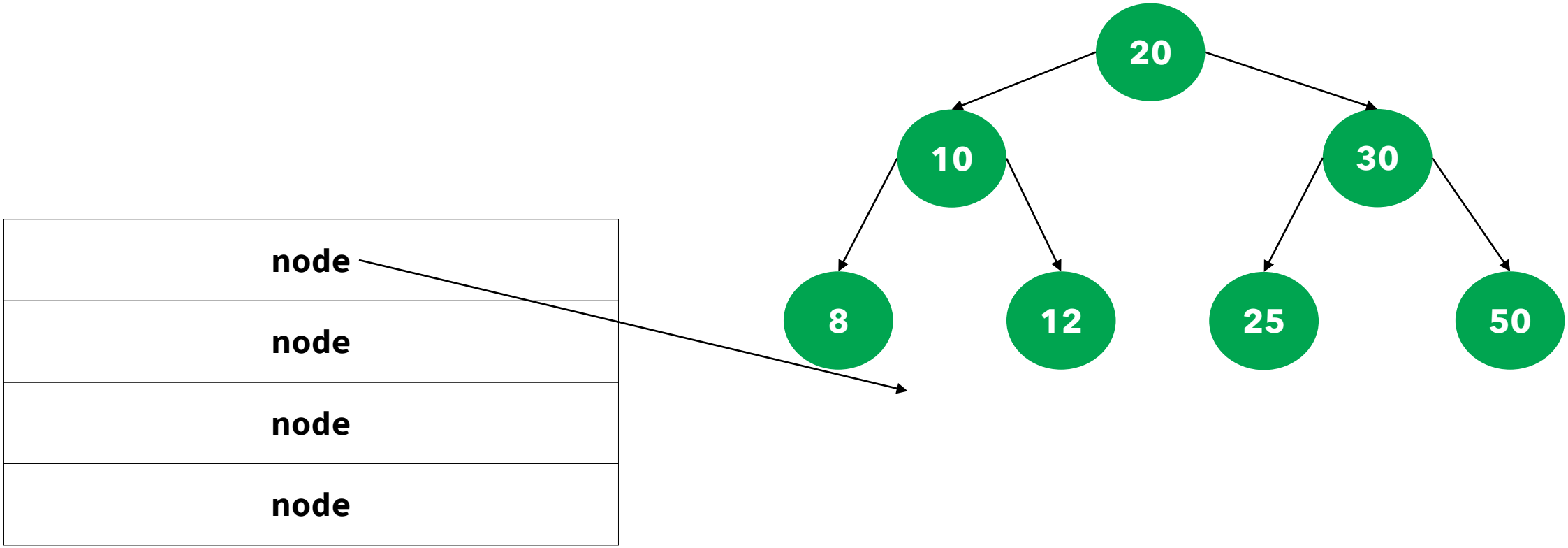
Depth-first search (DFS): inorder



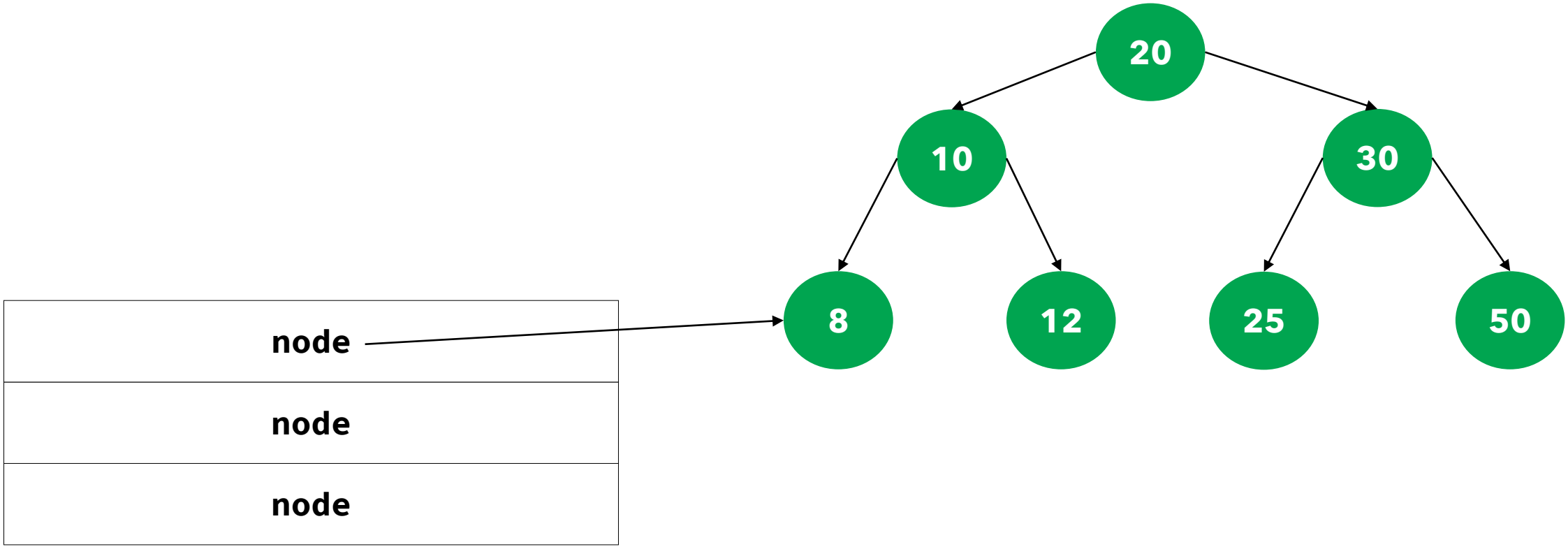
Depth-first search (DFS): inorder



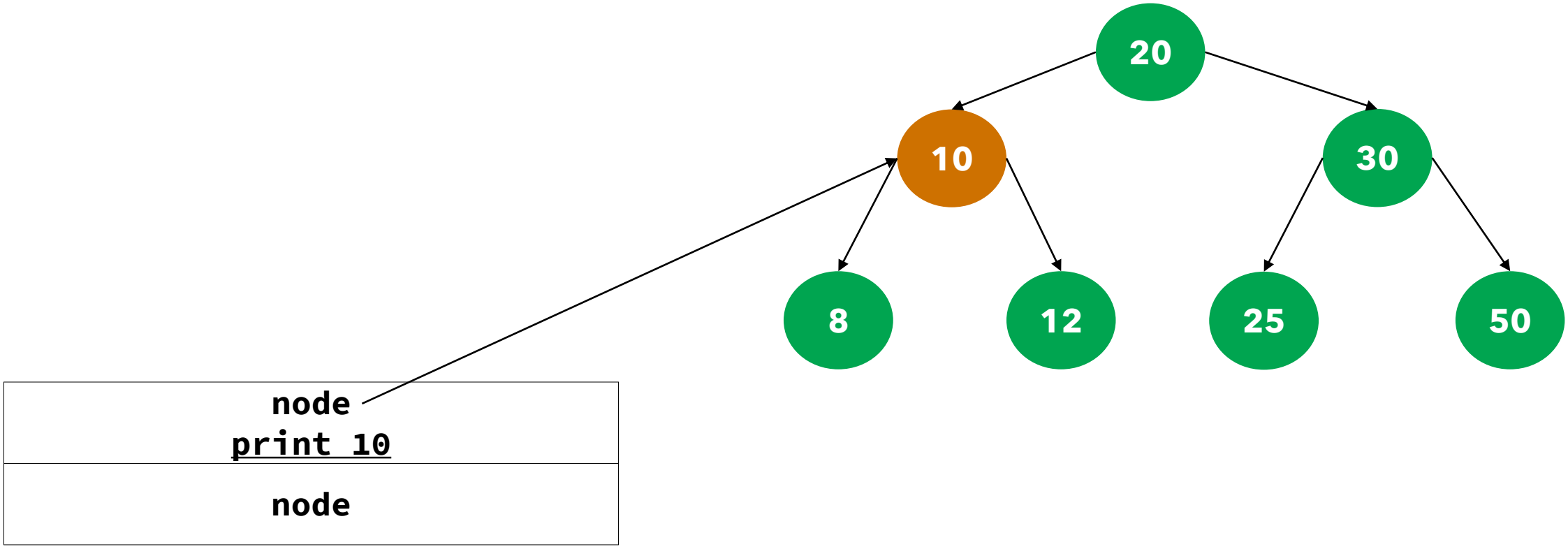
Depth-first search (DFS): inorder



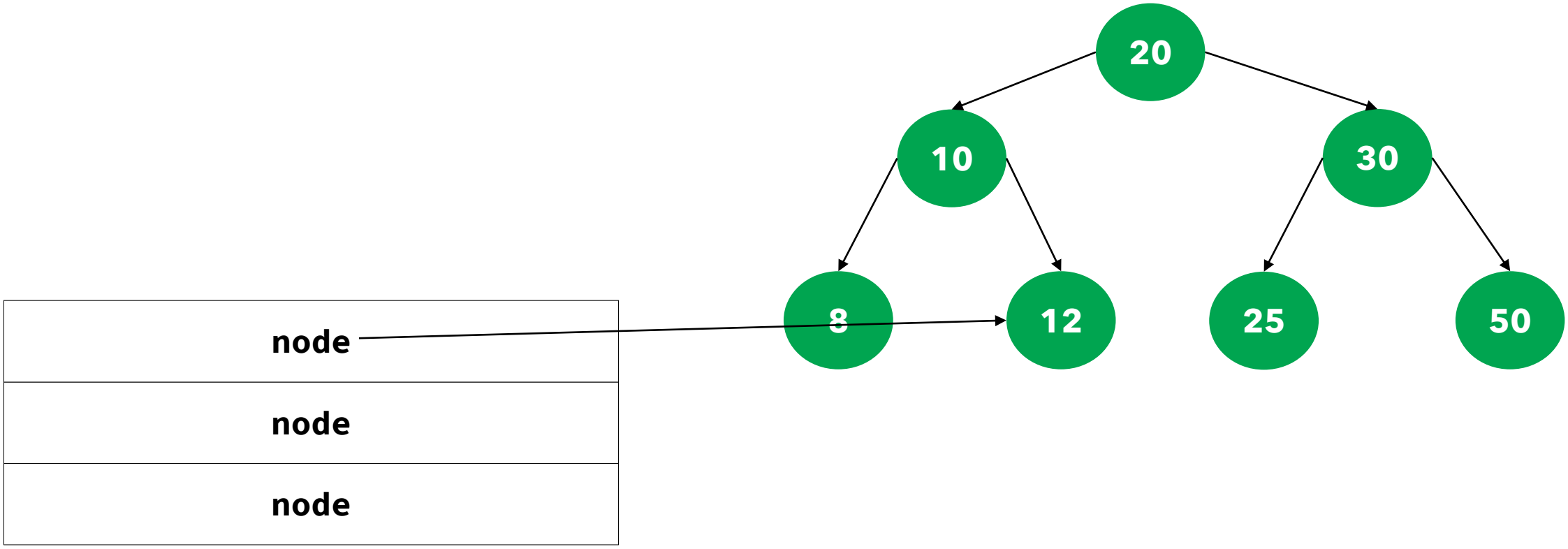
Depth-first search (DFS): inorder



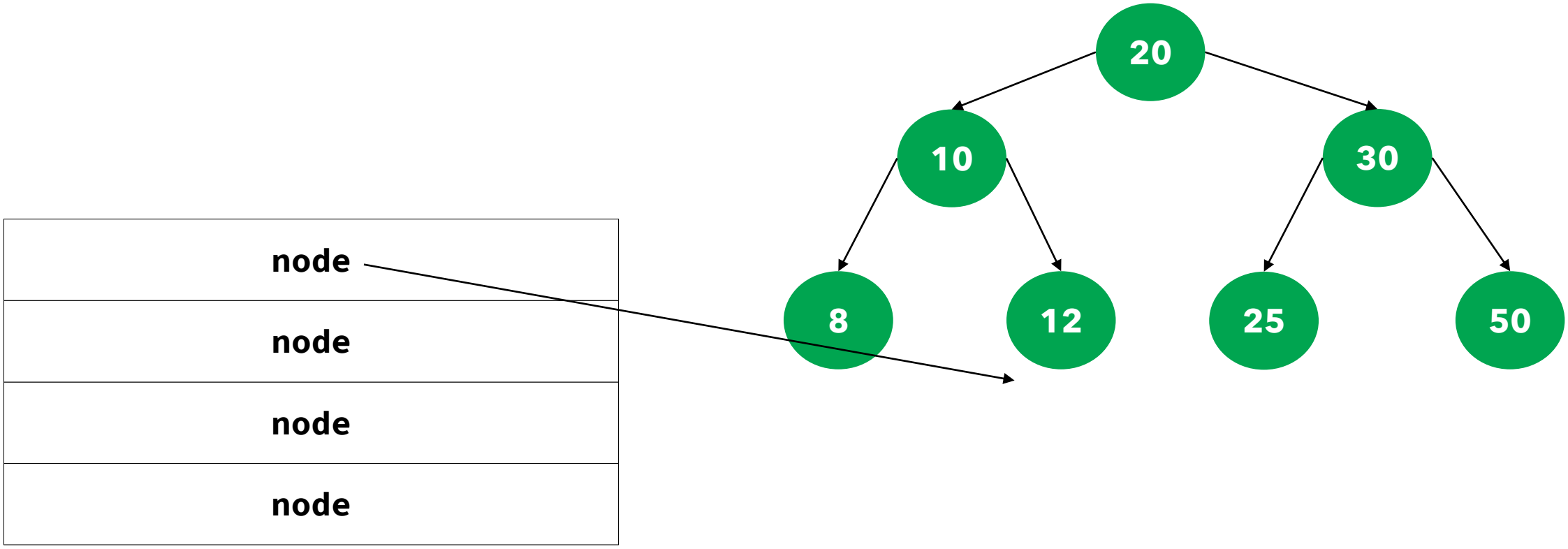
Depth-first search (DFS): inorder



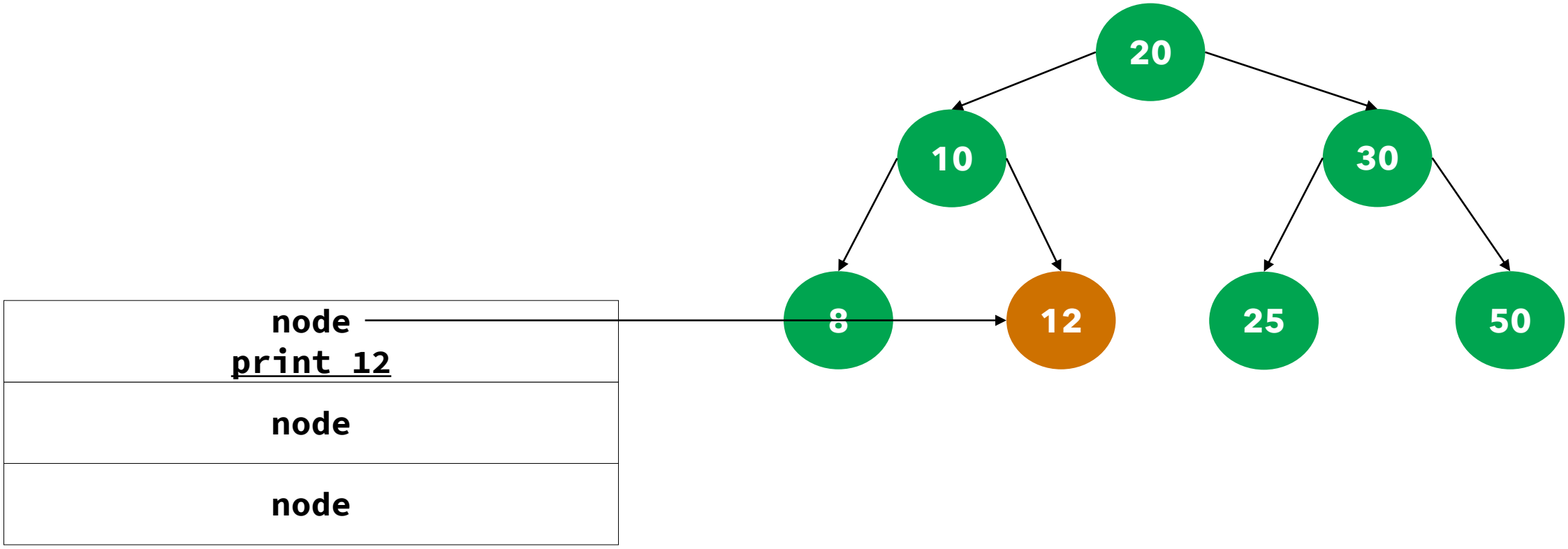
Depth-first search (DFS): inorder



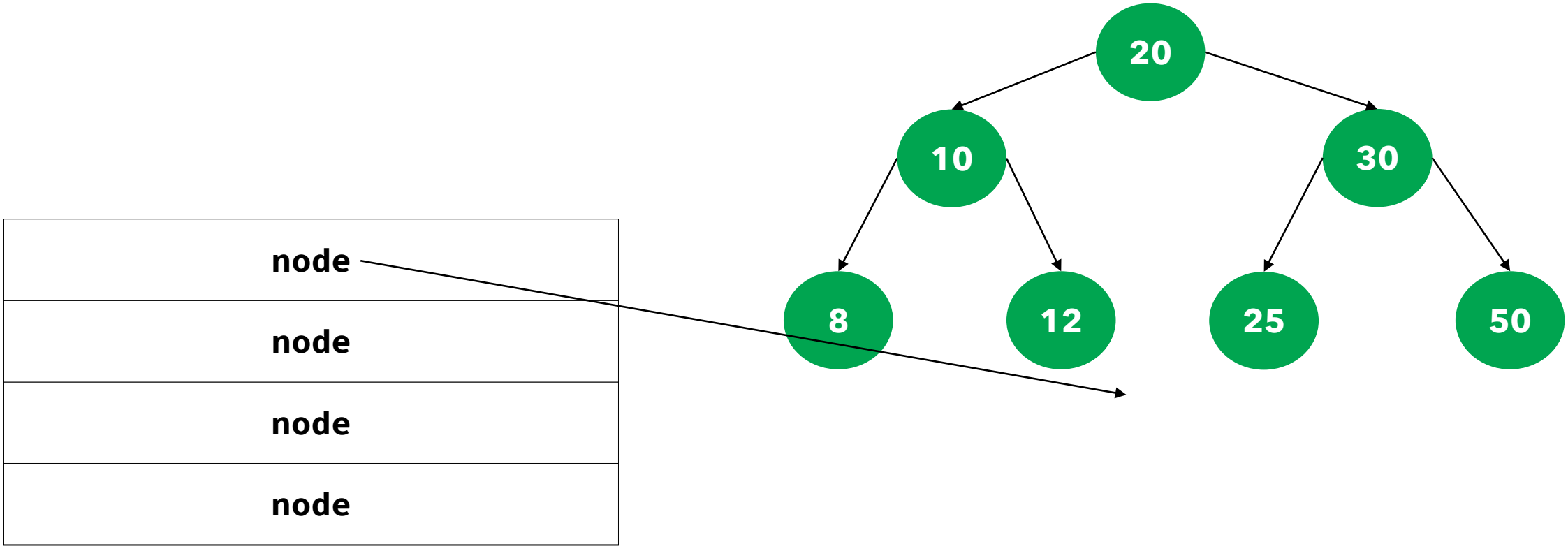
Depth-first search (DFS): inorder



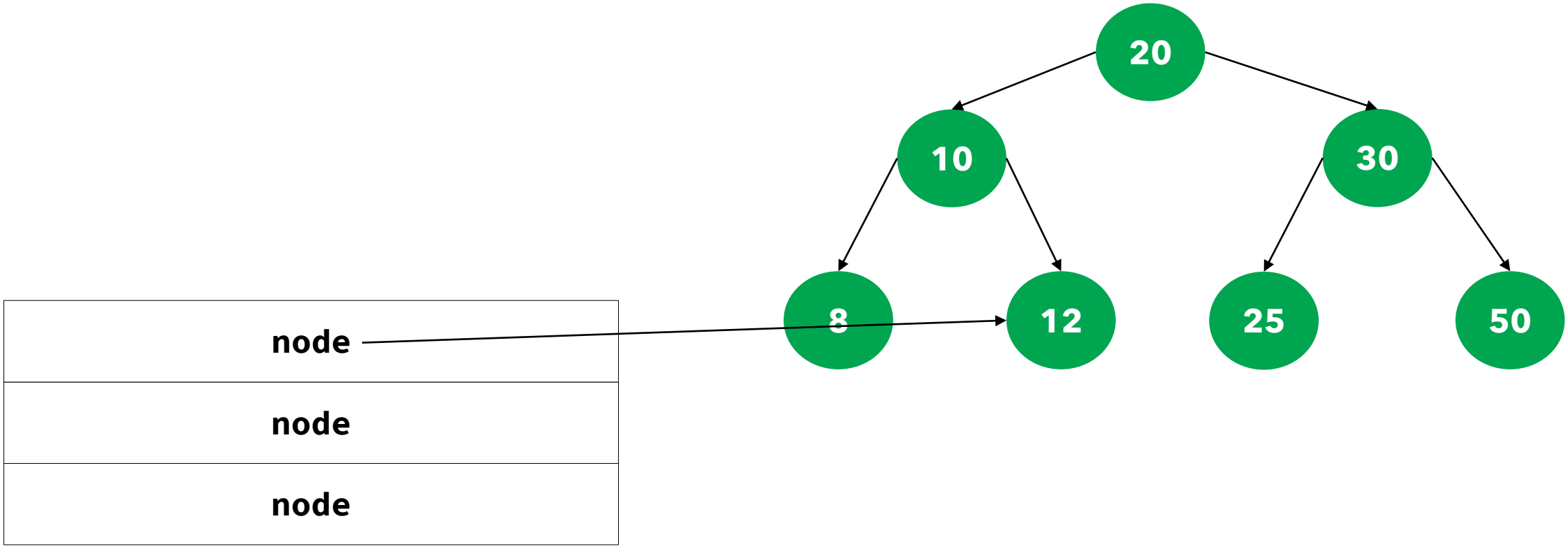
Depth-first search (DFS): inorder



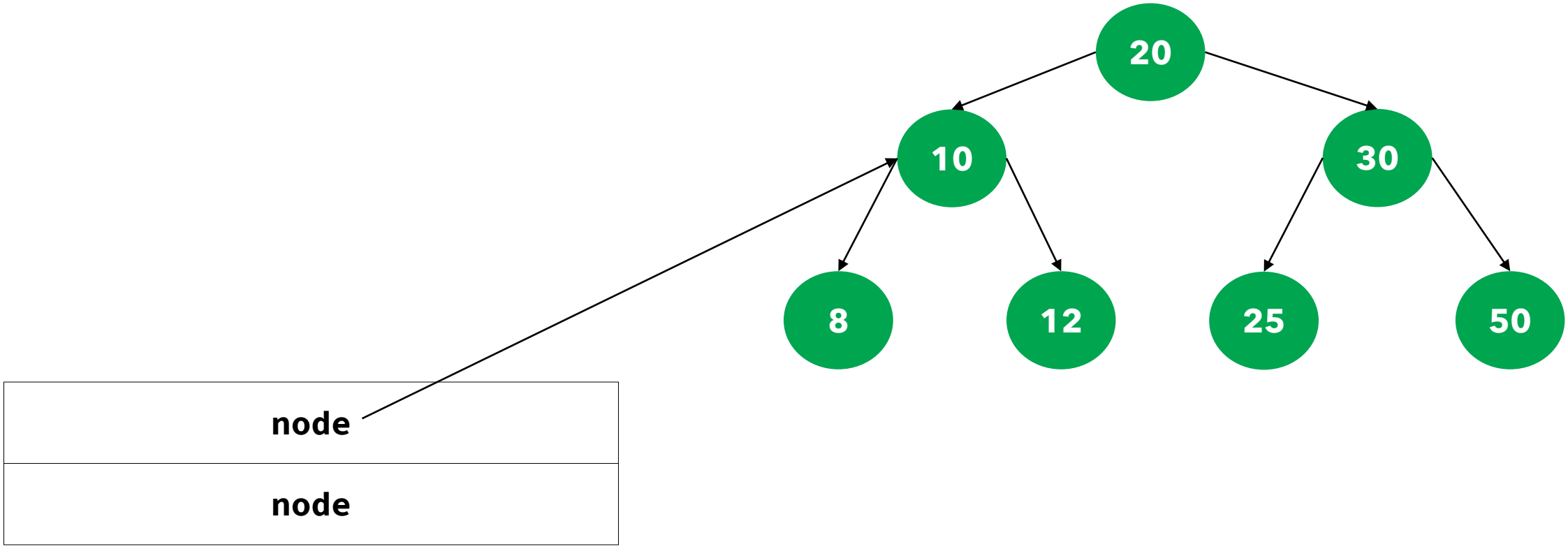
Depth-first search (DFS): inorder



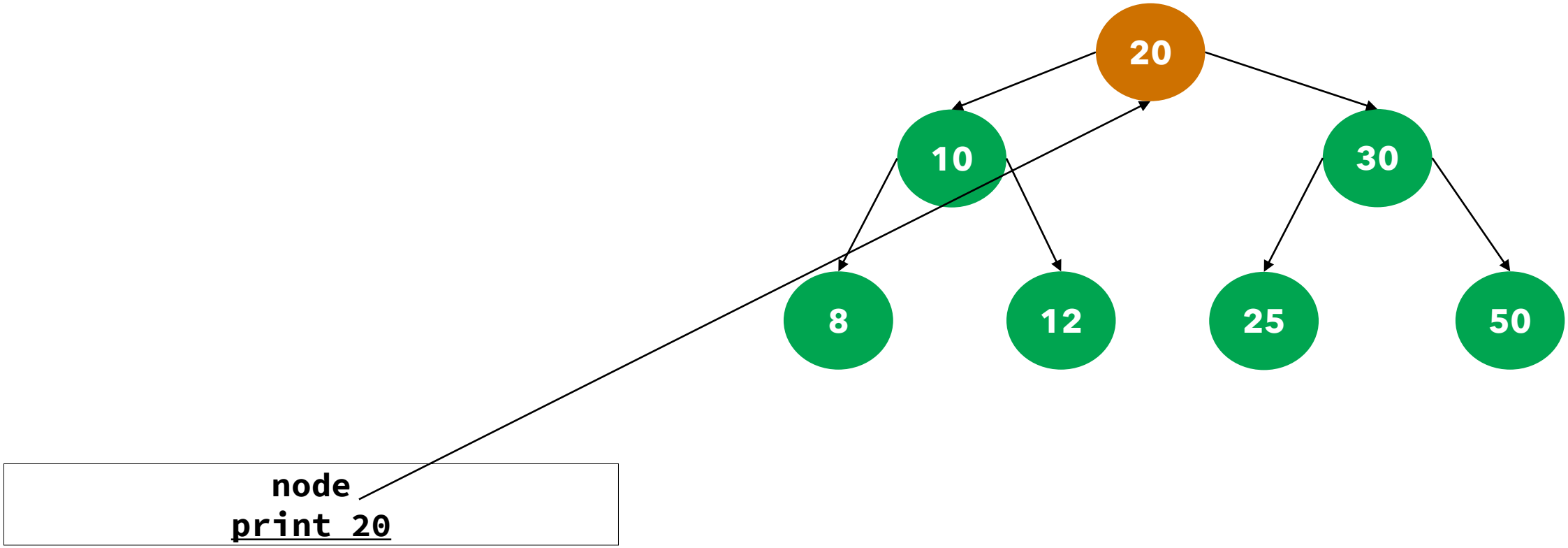
Depth-first search (DFS): inorder



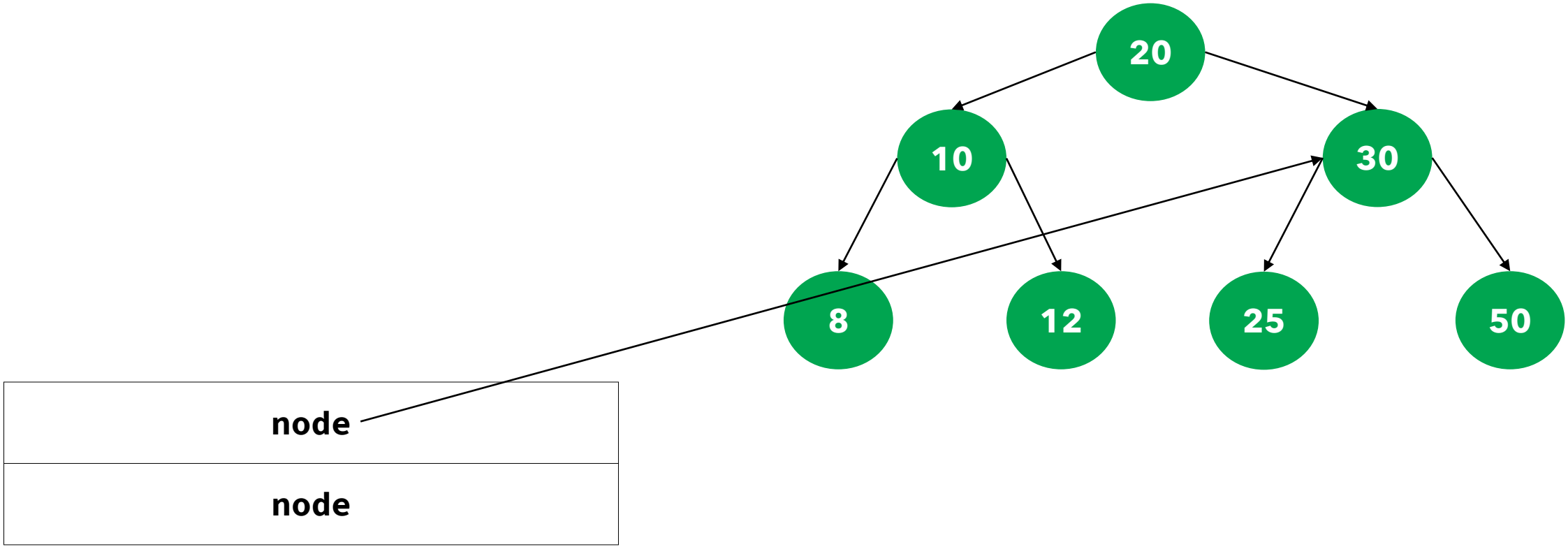
Depth-first search (DFS): inorder



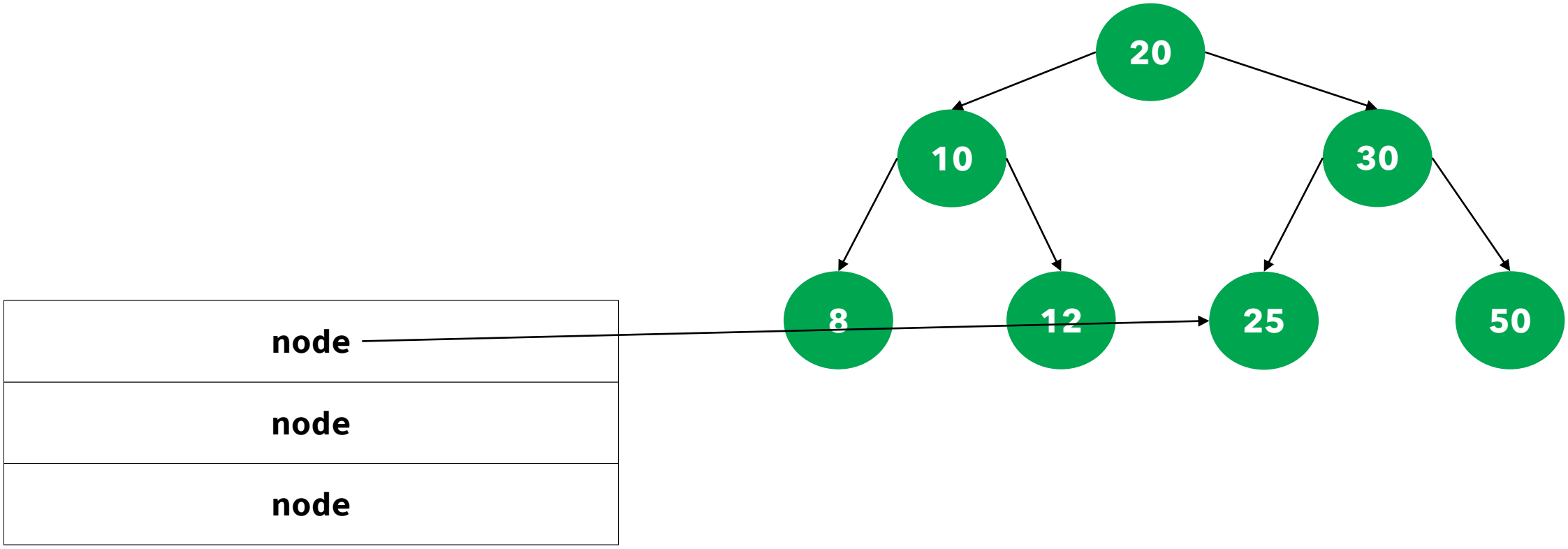
Depth-first search (DFS): inorder



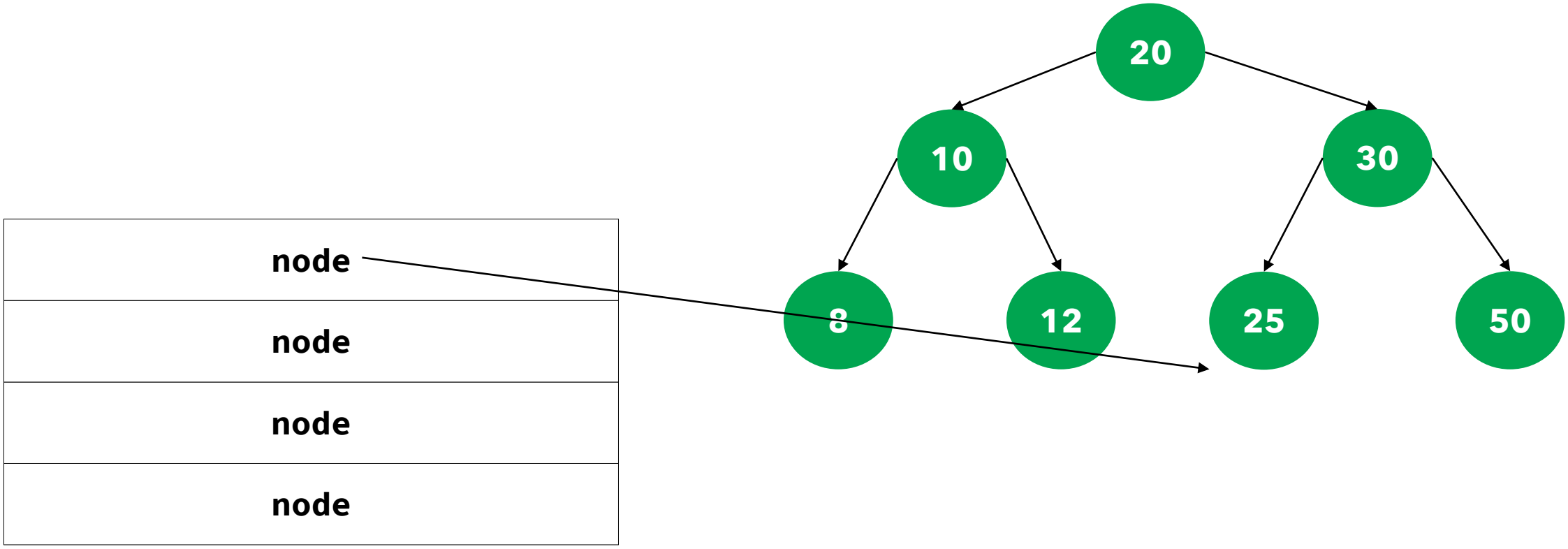
Depth-first search (DFS): inorder



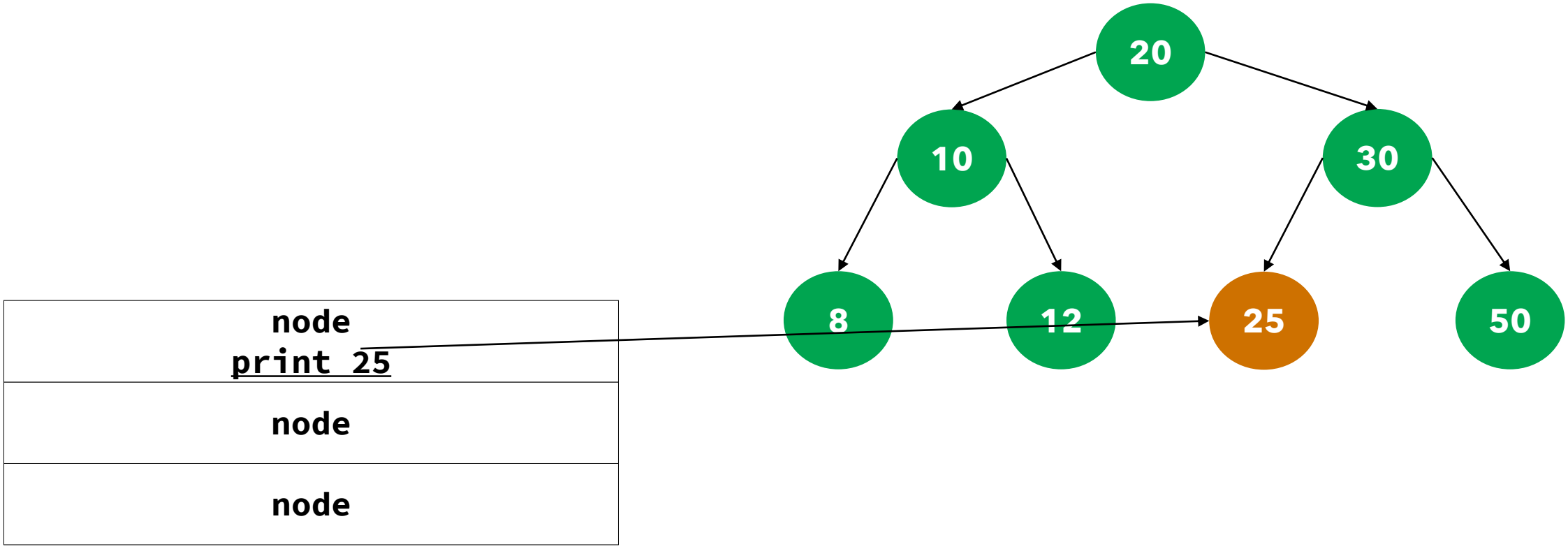
Depth-first search (DFS): inorder



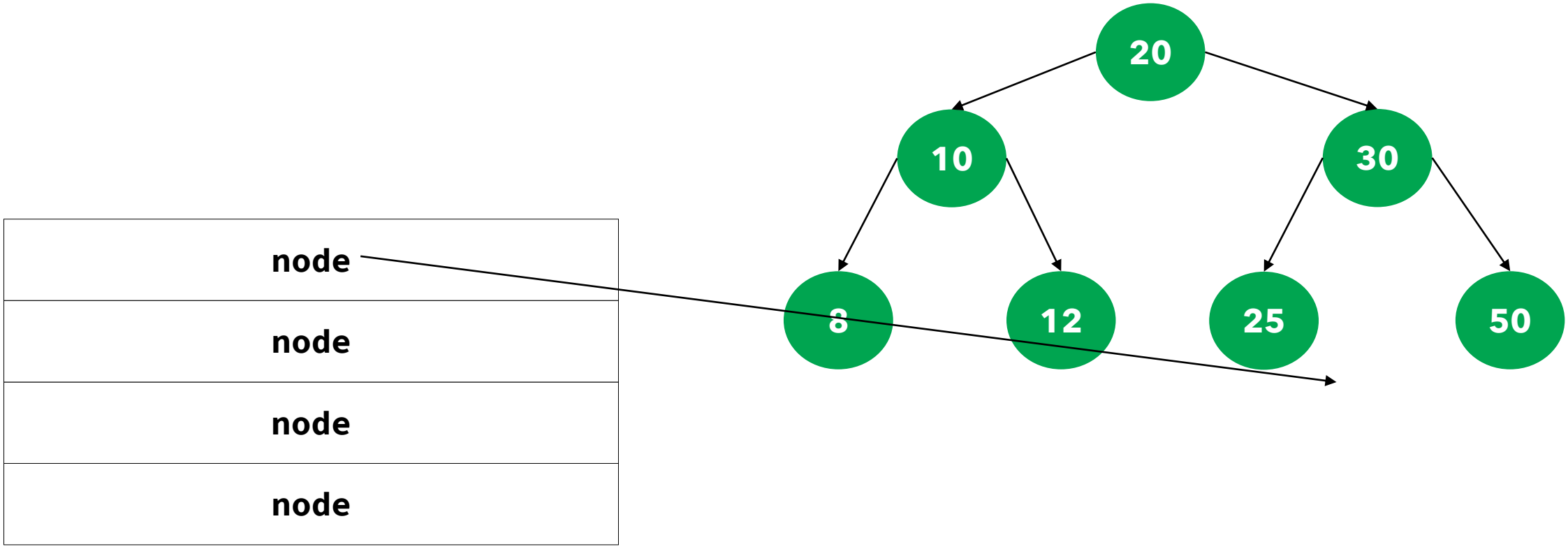
Depth-first search (DFS): inorder



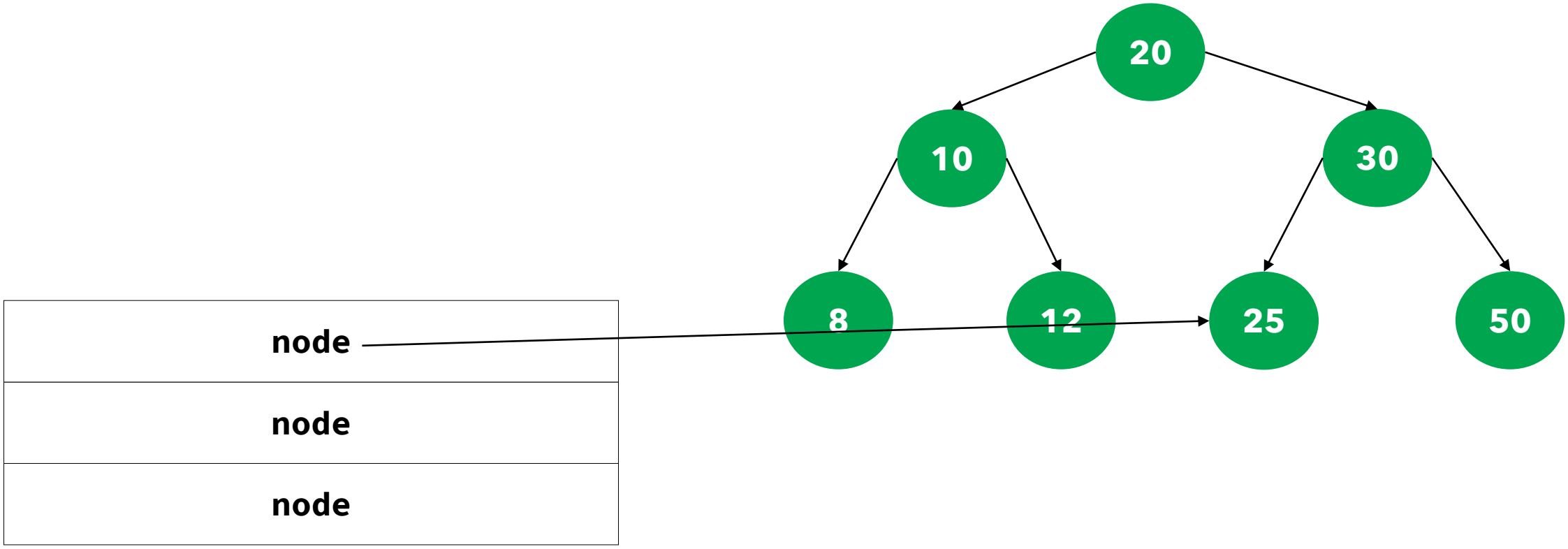
Depth-first search (DFS): inorder



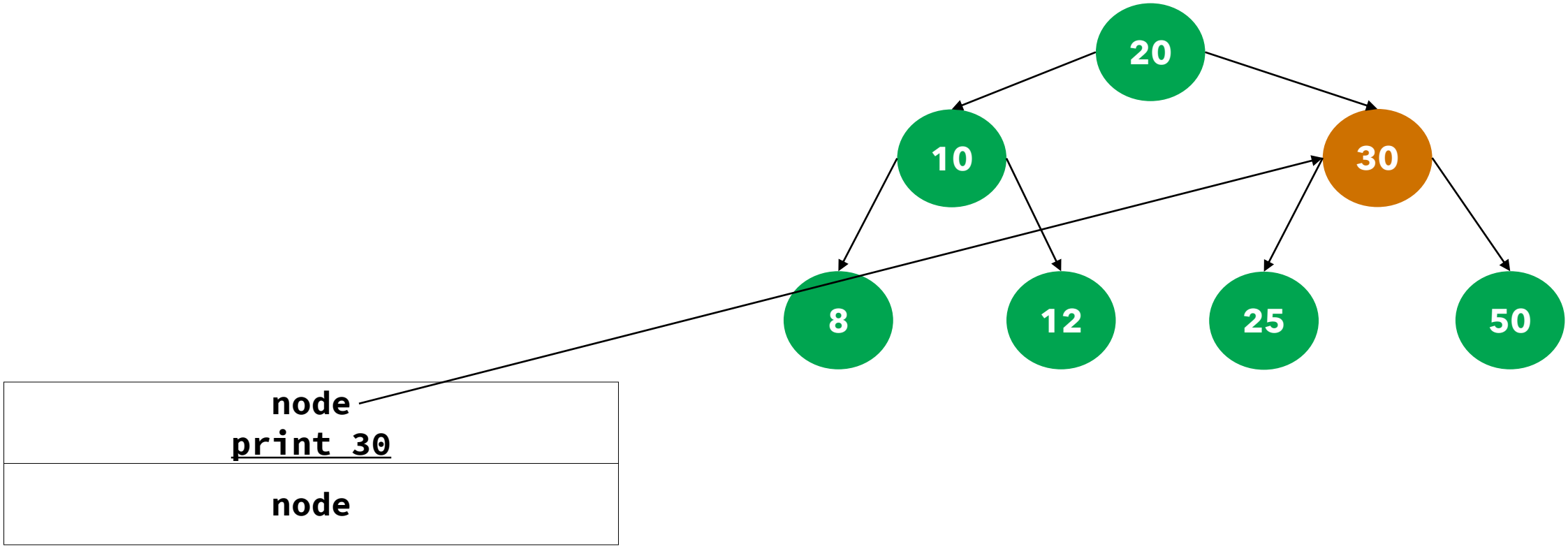
Depth-first search (DFS): inorder



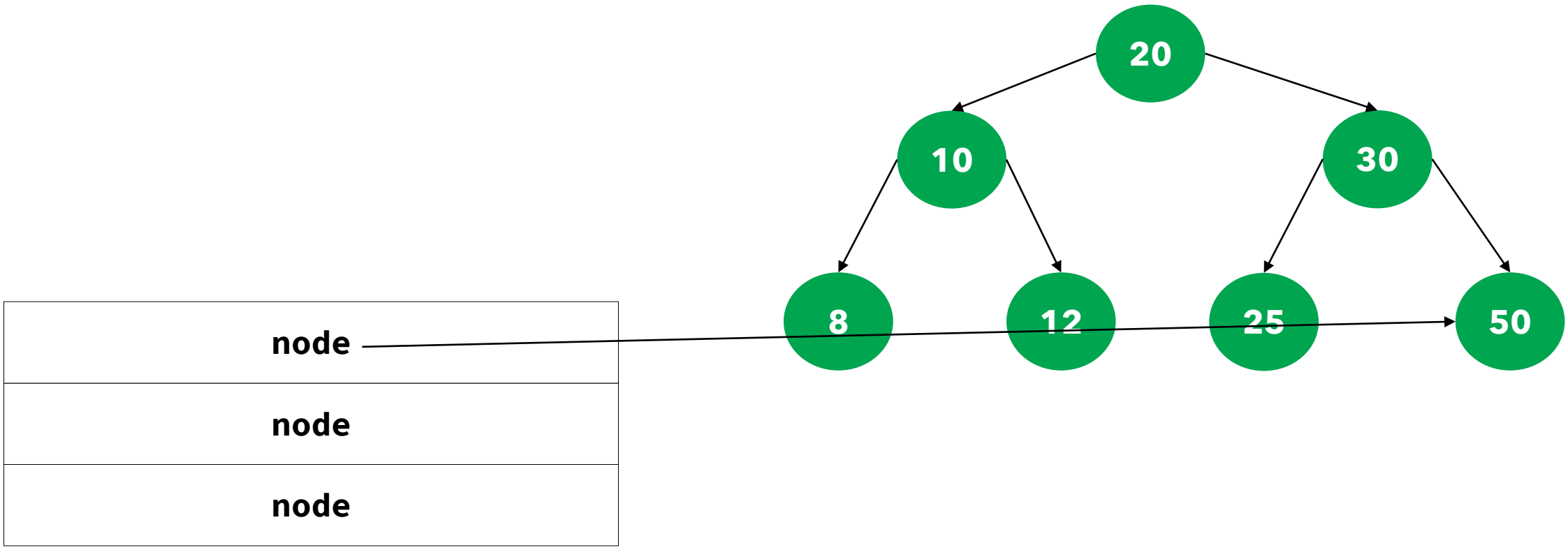
Depth-first search (DFS): inorder



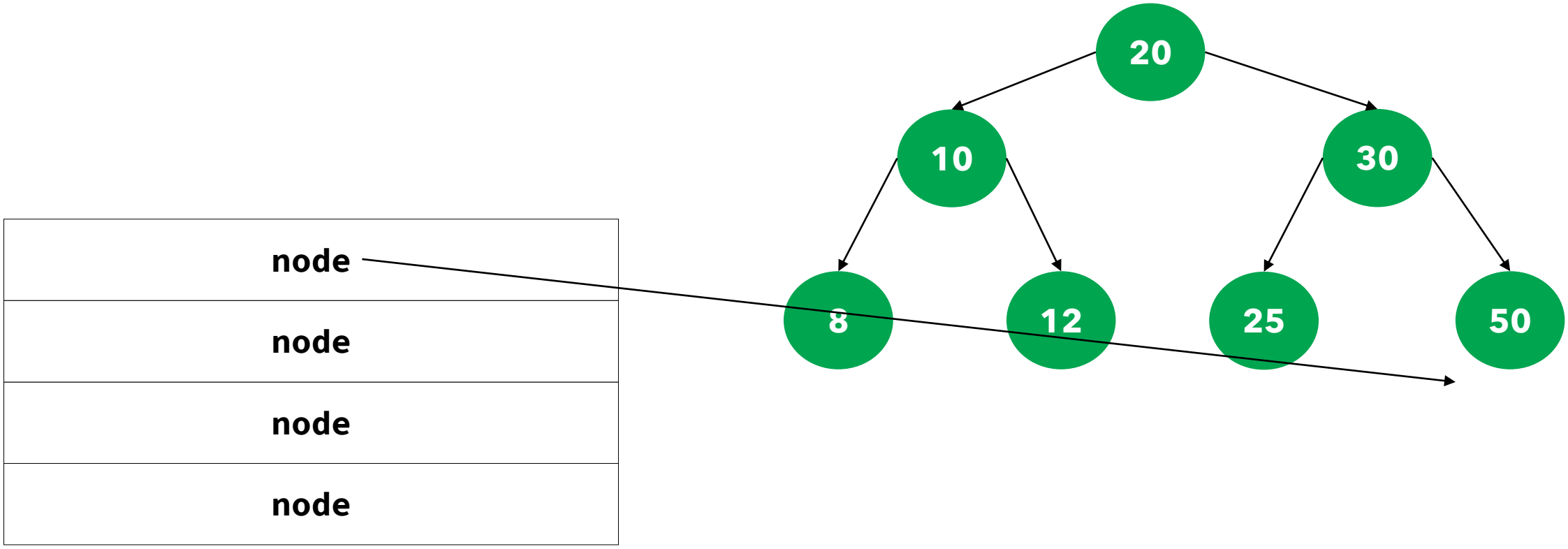
Depth-first search (DFS): inorder



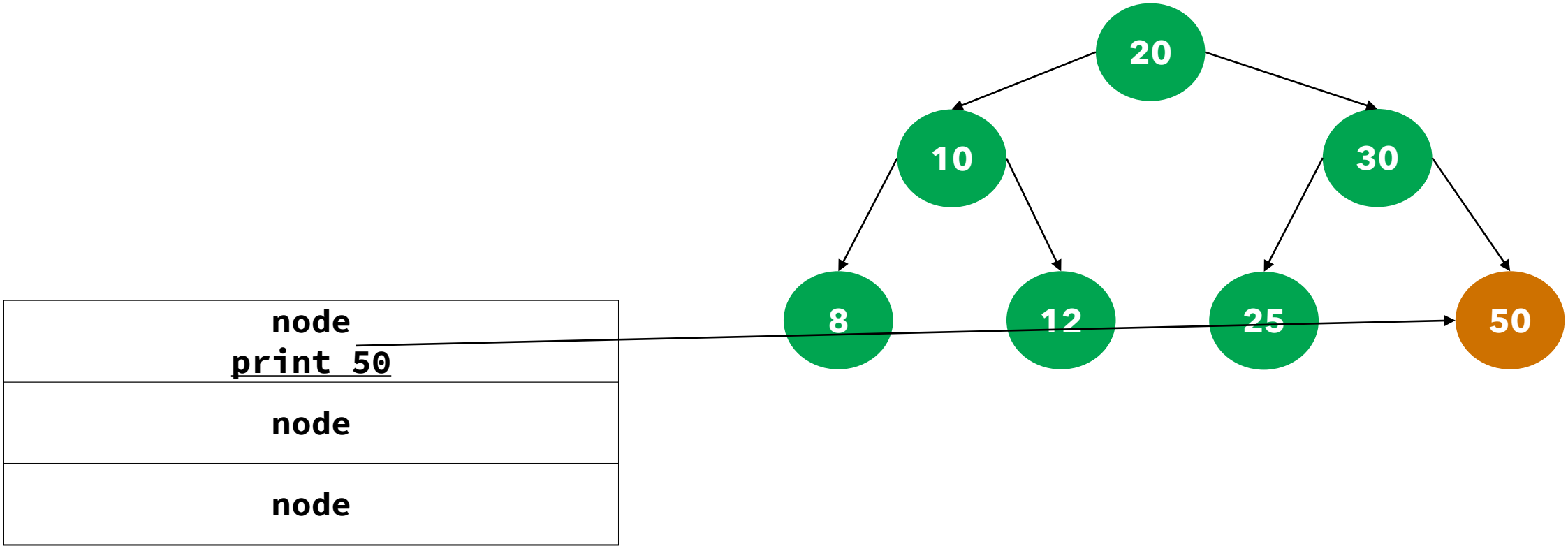
Depth-first search (DFS): inorder



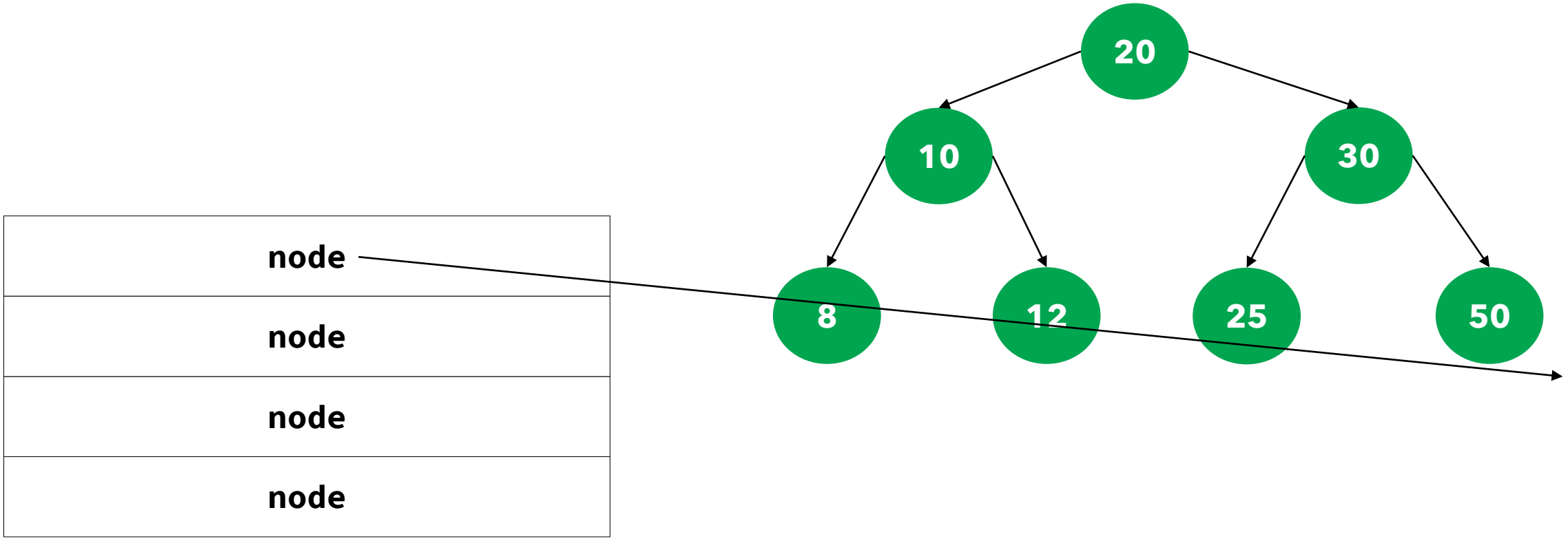
Depth-first search (DFS): inorder



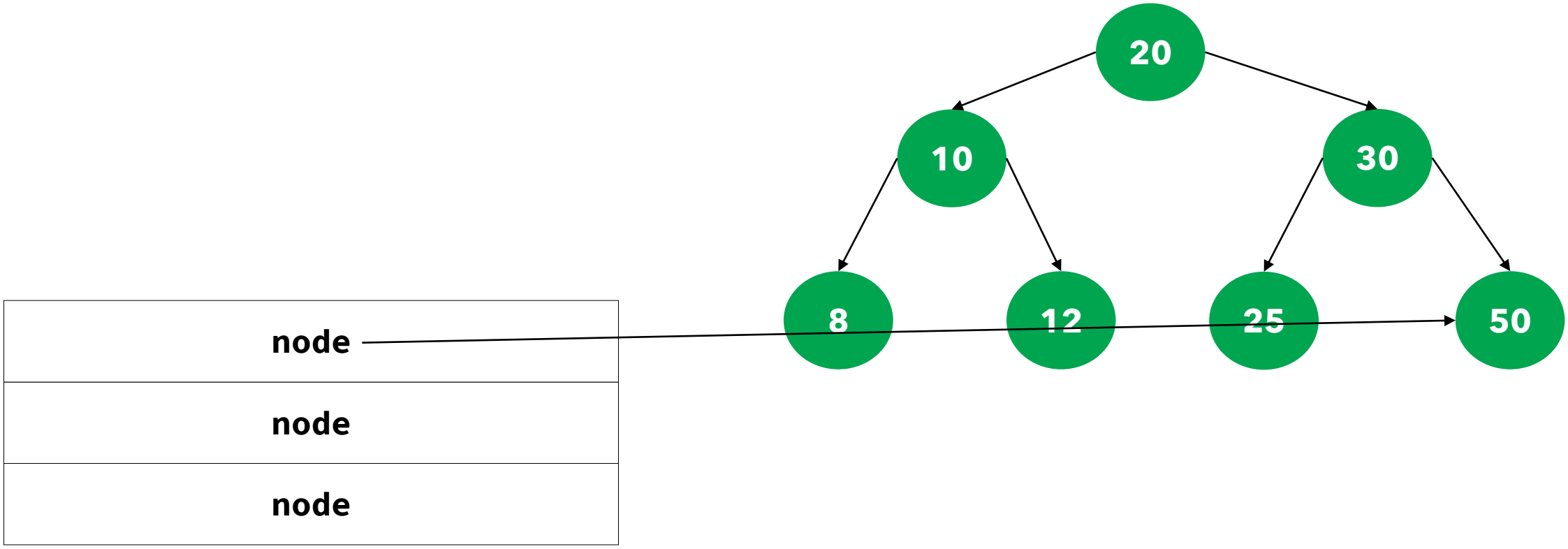
Depth-first search (DFS): inorder



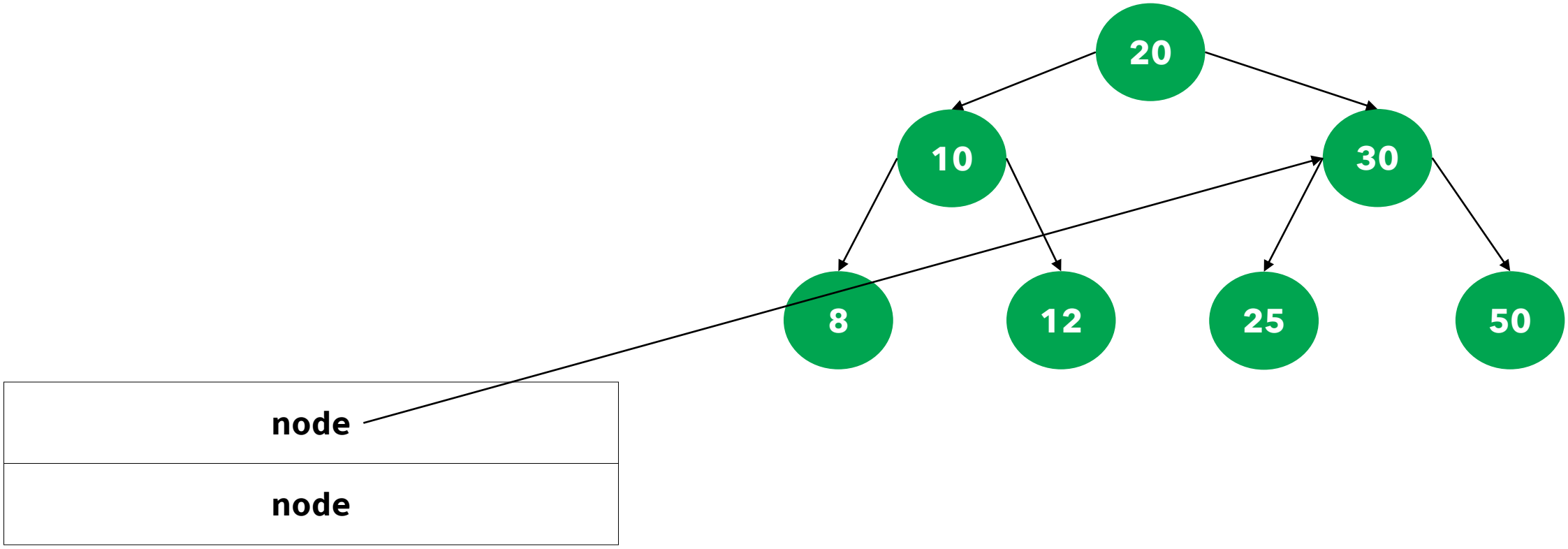
Depth-first search (DFS): inorder



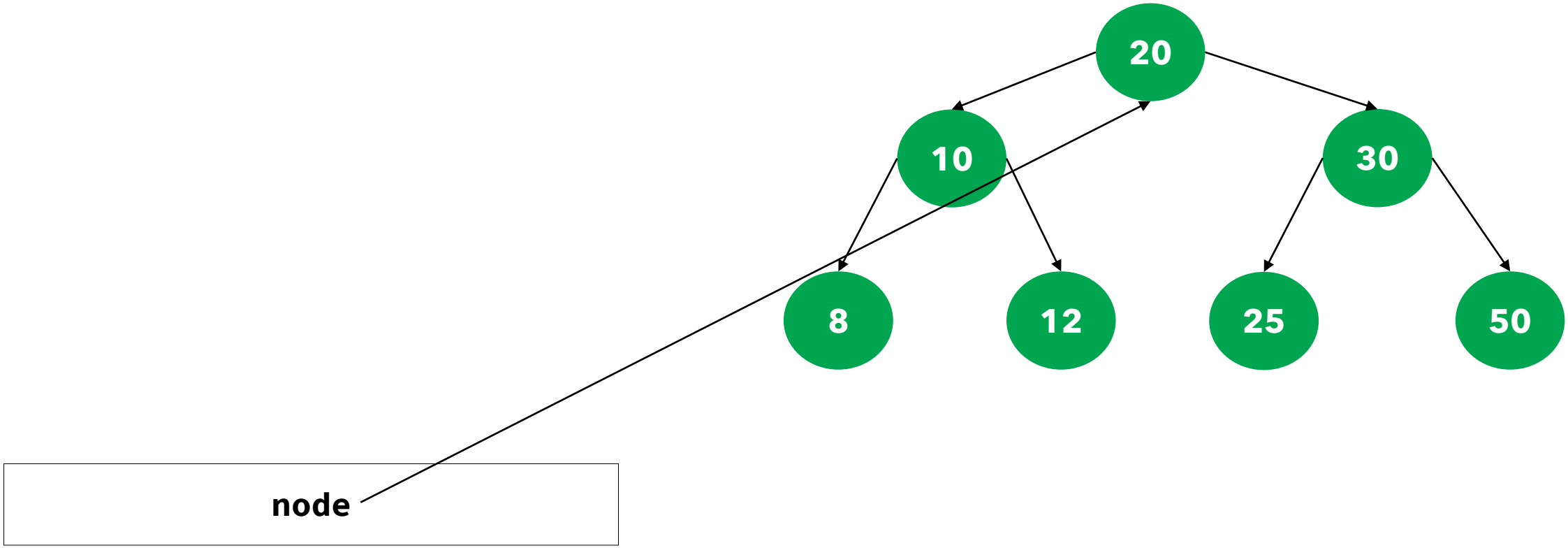
Depth-first search (DFS): inorder



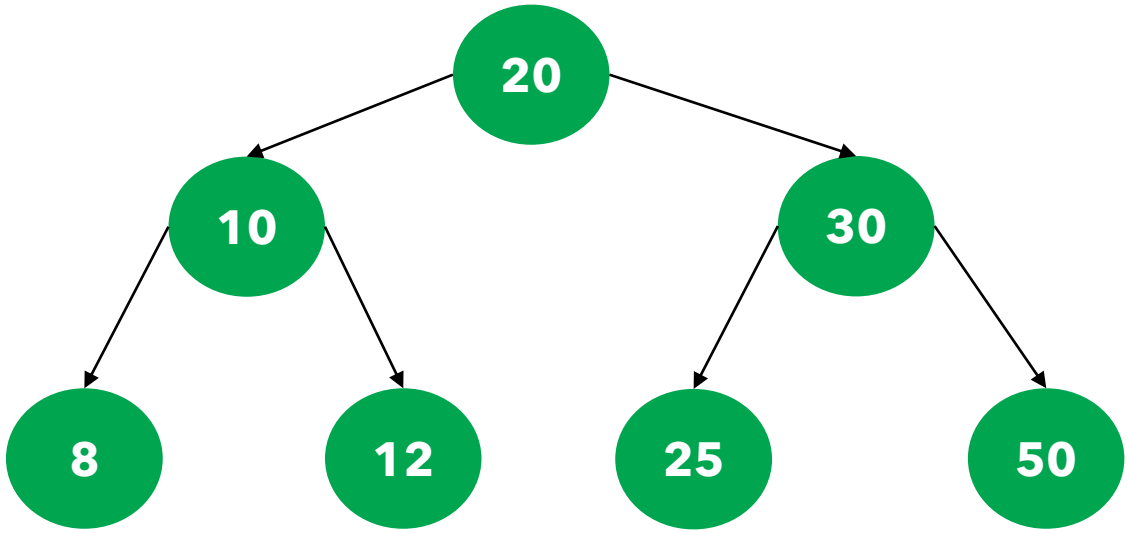
Depth-first search (DFS): inorder



Depth-first search (DFS): inorder



Depth-first search (DFS): inorder



Depth-first search (DFS): preorder

```
preorder_tree_walk(T):
```

- if T is empty, do nothing
- otherwise:
  - *open* T (e.g.: print T.key)
  - call preorder_tree_walk on T.left
  - call preorder_tree_walk on T.right

Depth-first search (DFS): postorder

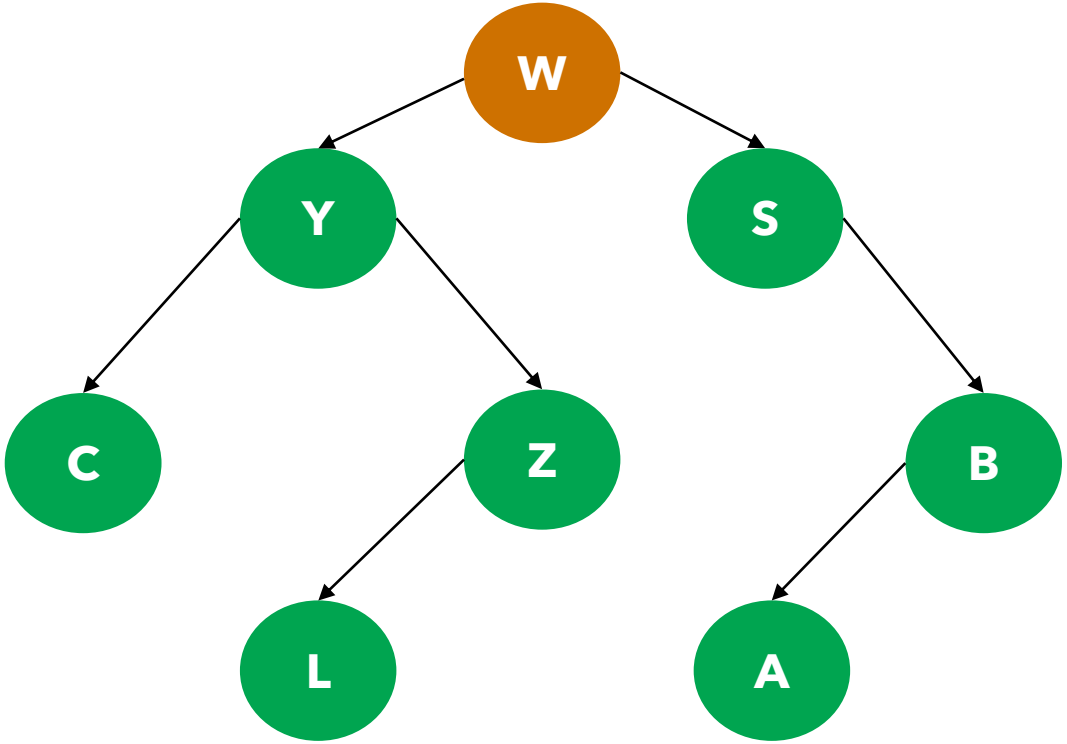
```
postorder_tree_walk(T):
```

- if T is empty, do nothing
- otherwise:
  - call postorder_tree_walk on T.left
  - call postorder_tree_walk on T.right
  - *open* T (e.g.: print T.key)

Breadth-first search (BFS)

- La ricerca in ampiezza è una ricerca per livelli
- Prima si visitano tutti i nodi al livello 0, poi tutti quelli al livello 1, poi tutti quelli al livello 2 etc…
- Visiteremo ciascun livello da sinistra a destra
- Per tenere traccia dei livelli nell'ordine corretto serve una **queue**

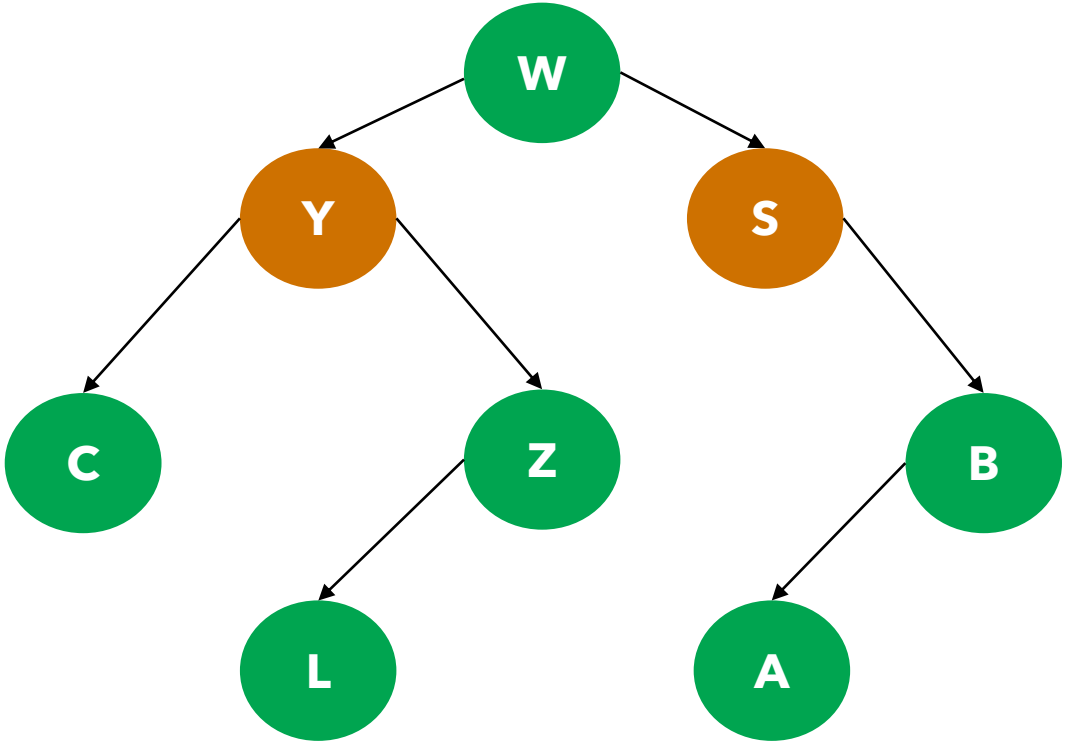
Breadth-first search (BFS)




```
Q: []
enqueue(W);

Q: [W]
```

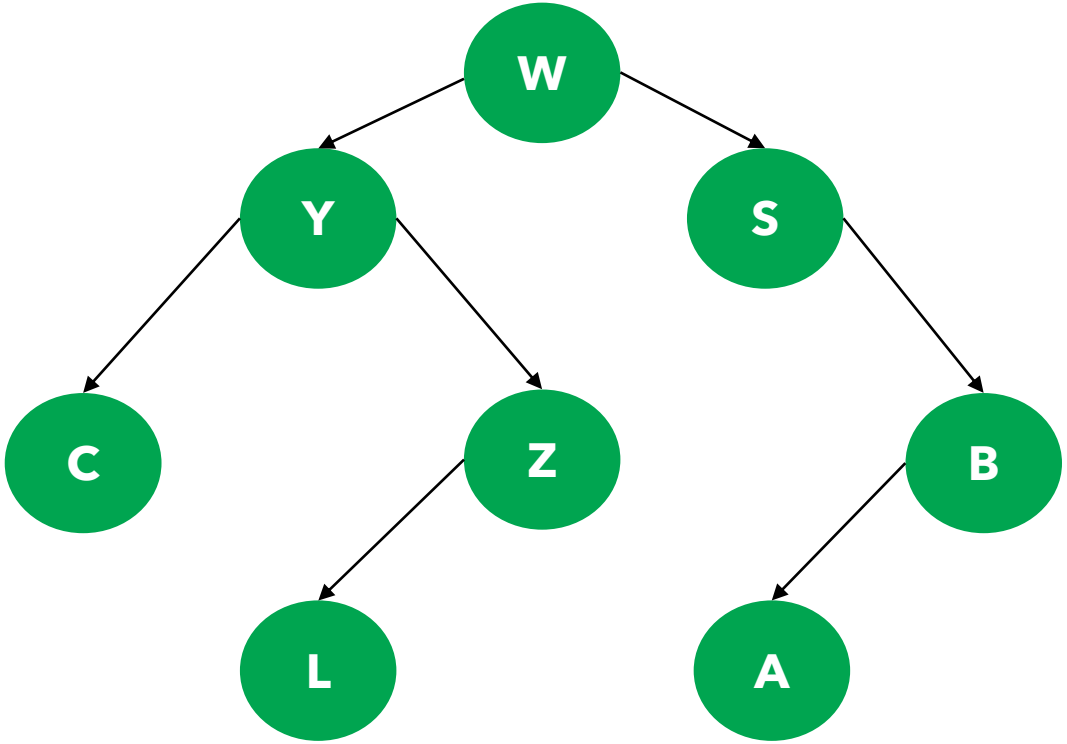
Breadth-first search (BFS)




```
enqueue(W.left);
enqueue(W.right);

Q: [W, Y, S]
```

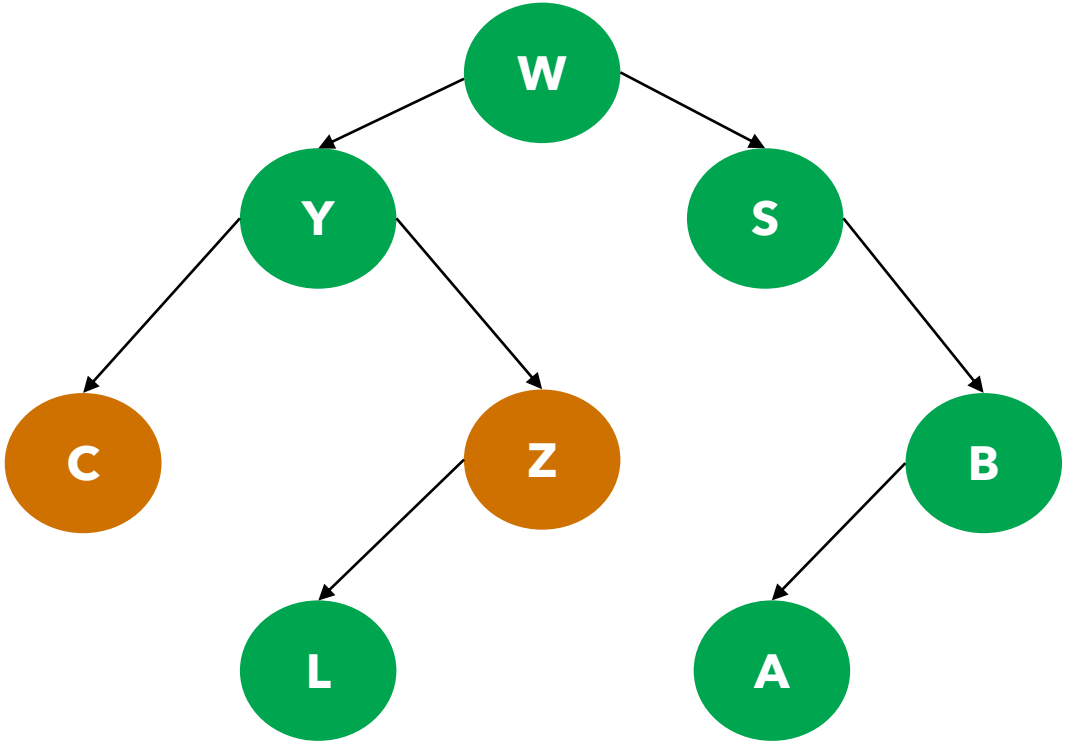
Breadth-first search (BFS)




```
dequeue();

Q: [Y, S]
```

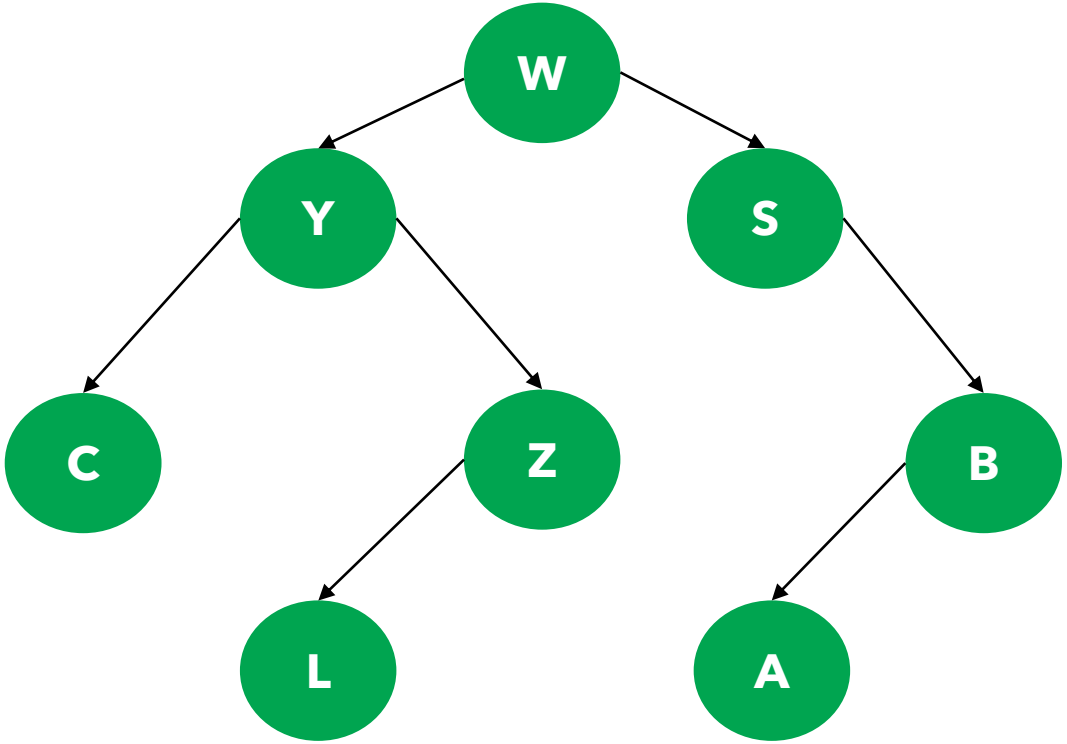
Breadth-first search (BFS)




```
enqueue(Y.left);
enqueue(Y.right);

Q: [Y, S, C, Z]
```

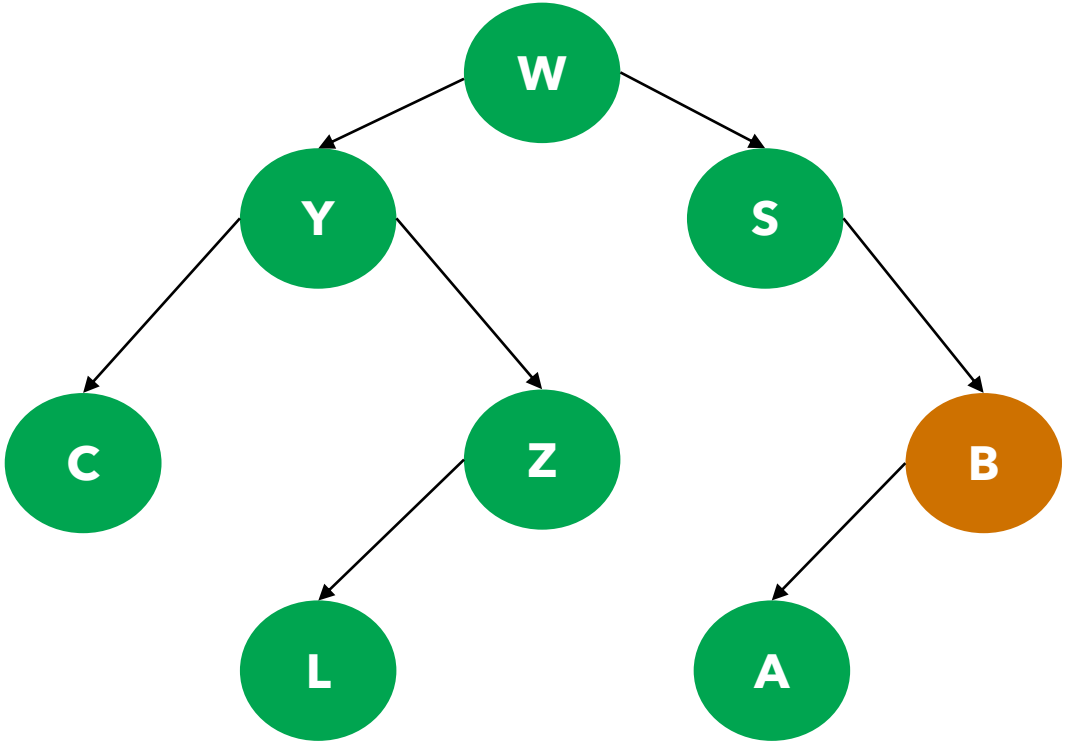
Breadth-first search (BFS)




```
dequeue();

Q: [S, C, Z]
```

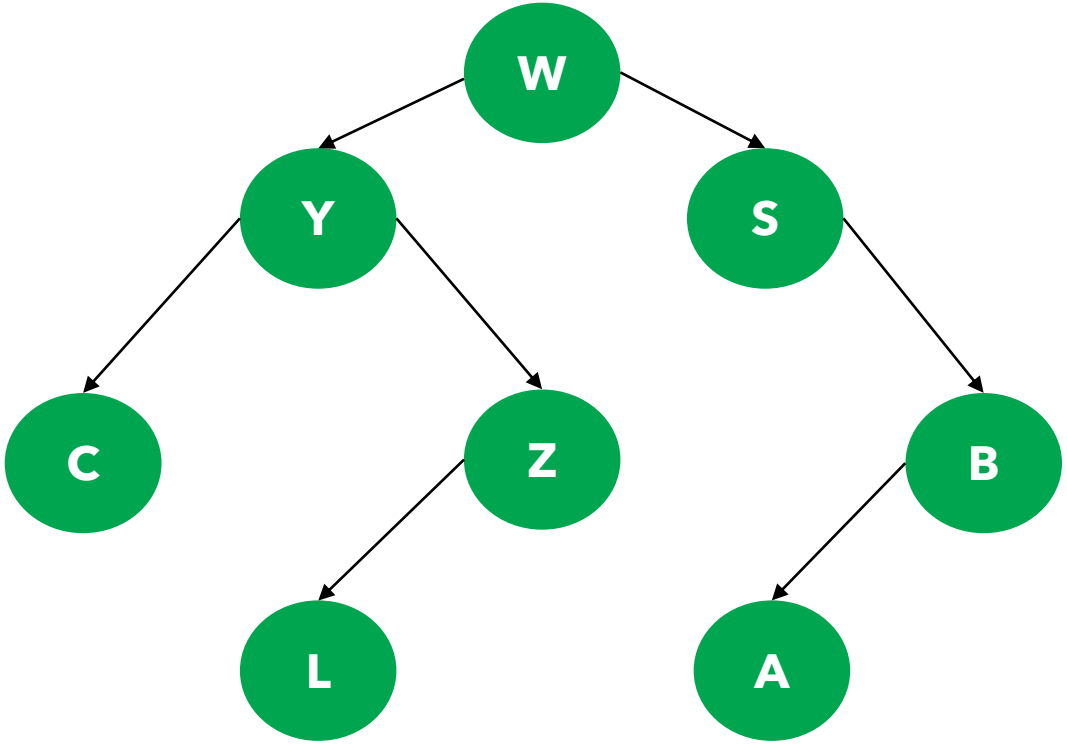
Breadth-first search (BFS)




```
enqueue(S.right);

Q: [S, C, Z, B]
```

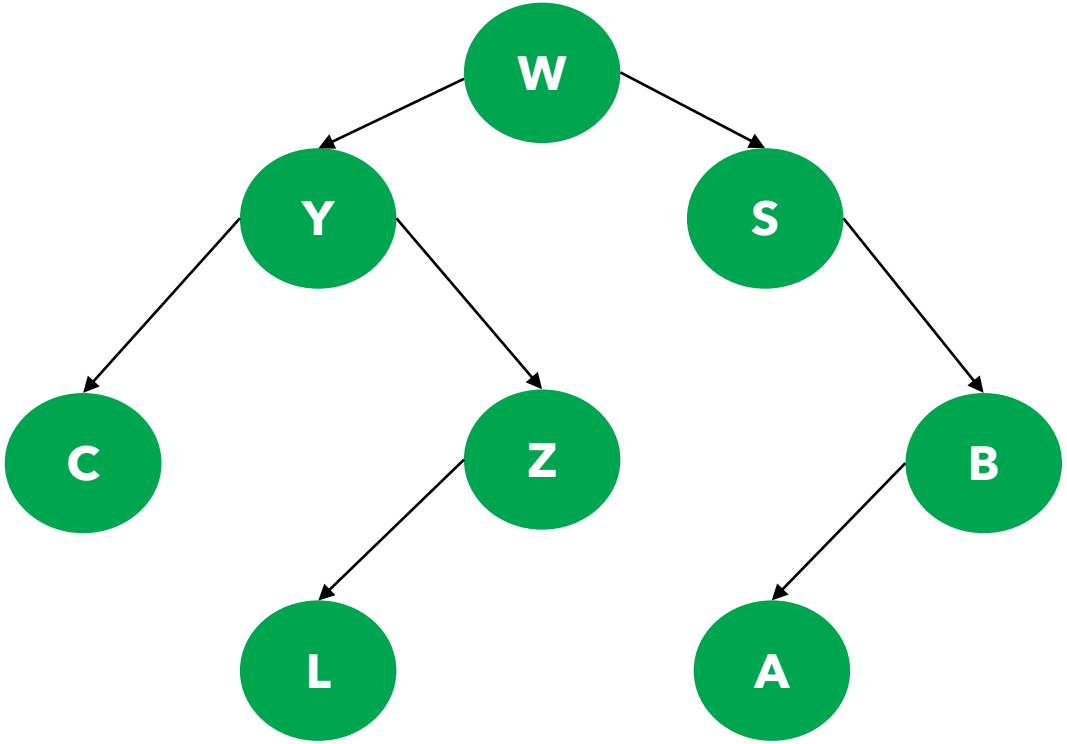
Breadth-first search (BFS)




```
dequeue();

Q: [C, Z, B]
```

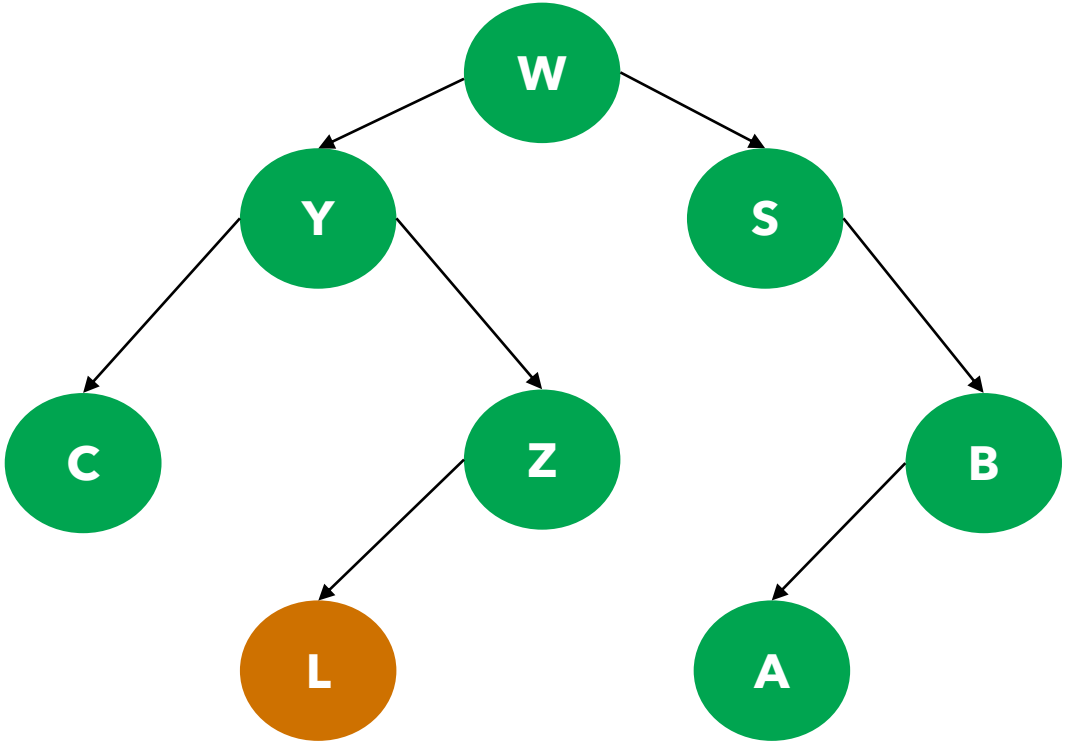
Breadth-first search (BFS)




```
dequeue();

Q: [Z, B]
```

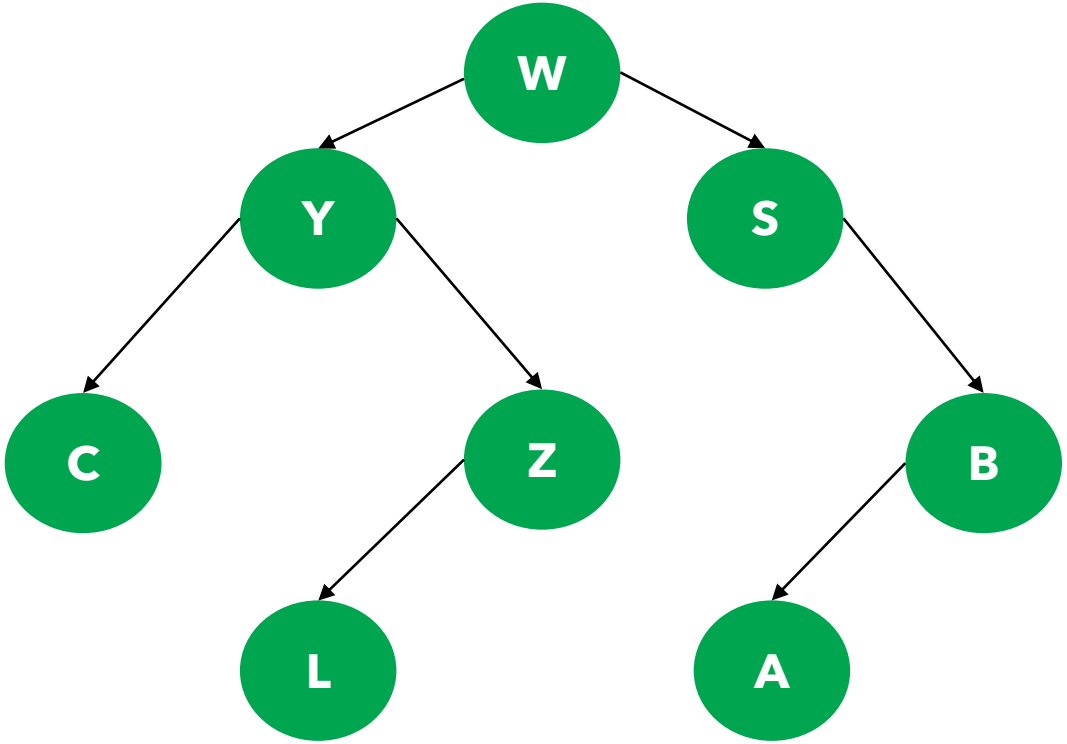
Breadth-first search (BFS)




```
enqueue(Z.left);

Q: [Z, B, L]
```

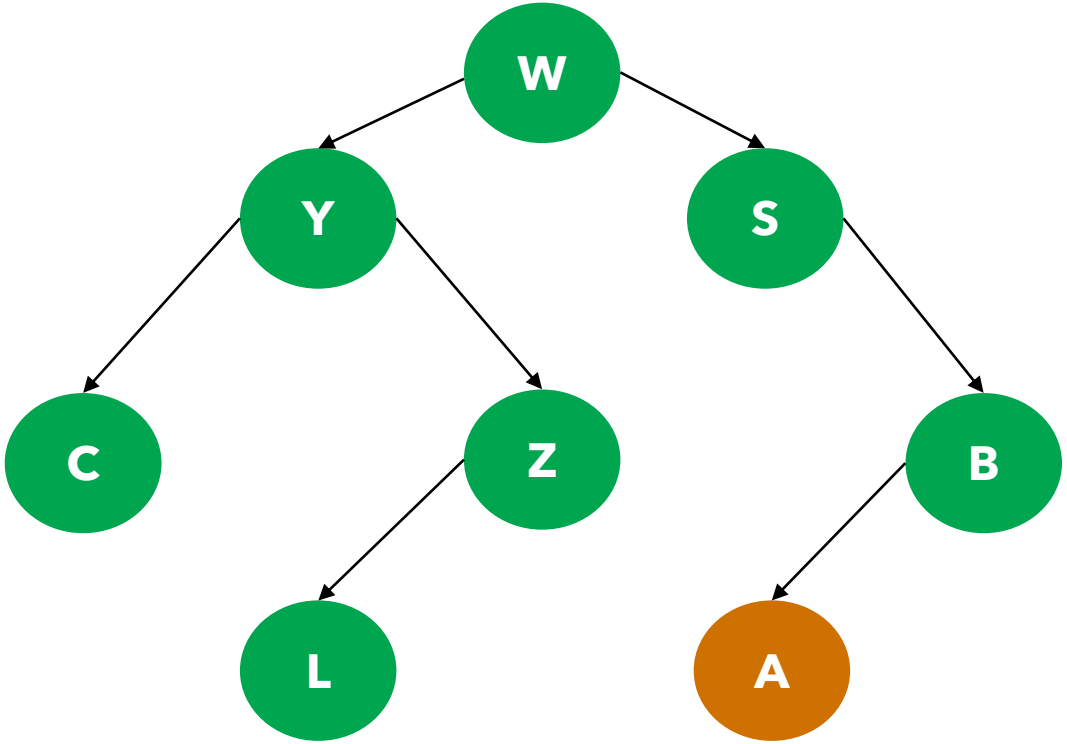
Breadth-first search (BFS)




```
dequeue();

Q: [B, L]
```

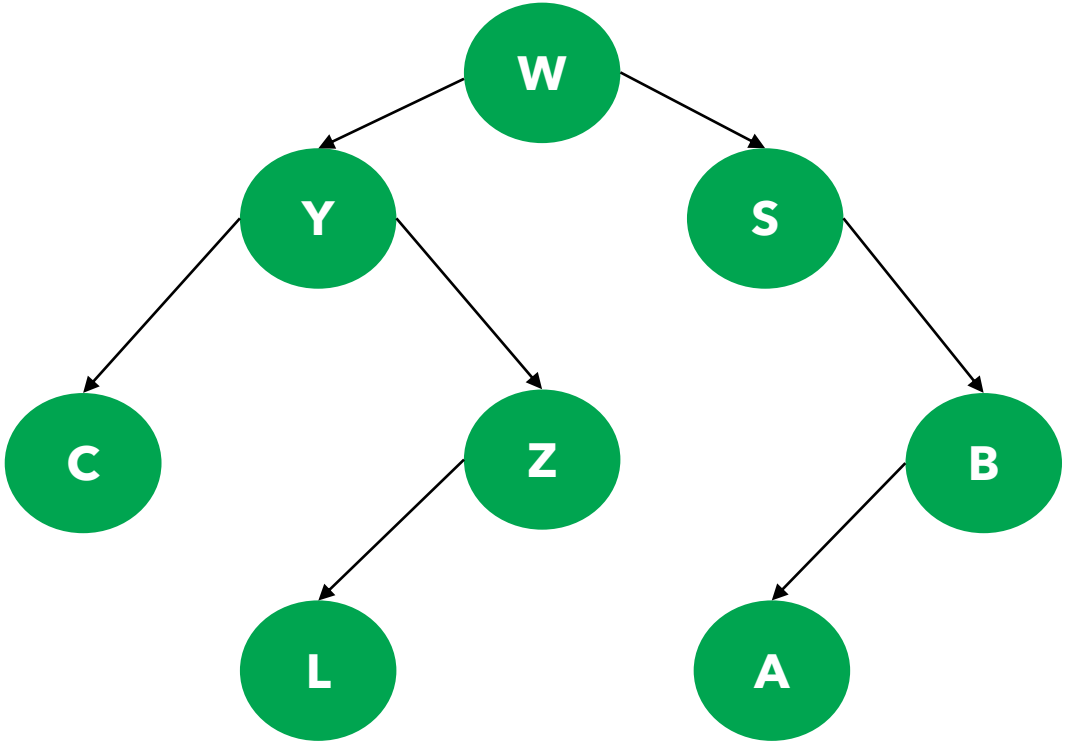
Breadth-first search (BFS)




```
enqueue(B.left);

Q: [B, L, A]
```

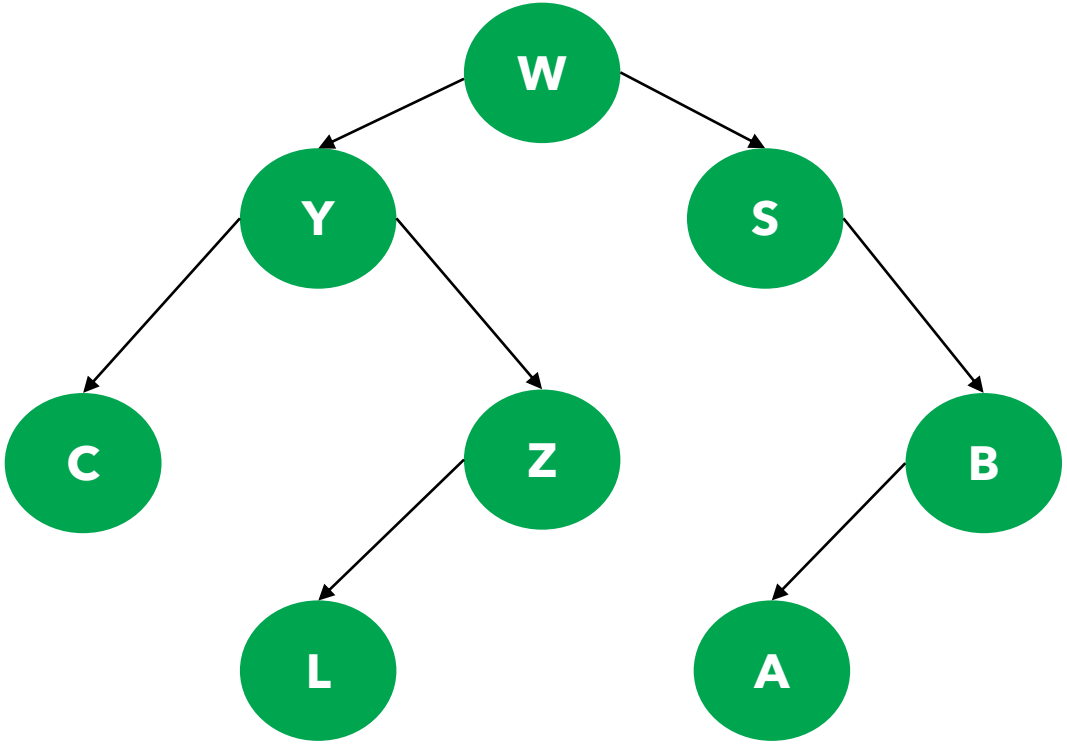
Breadth-first search (BFS)




```
dequeue();

Q: [L, A]
```

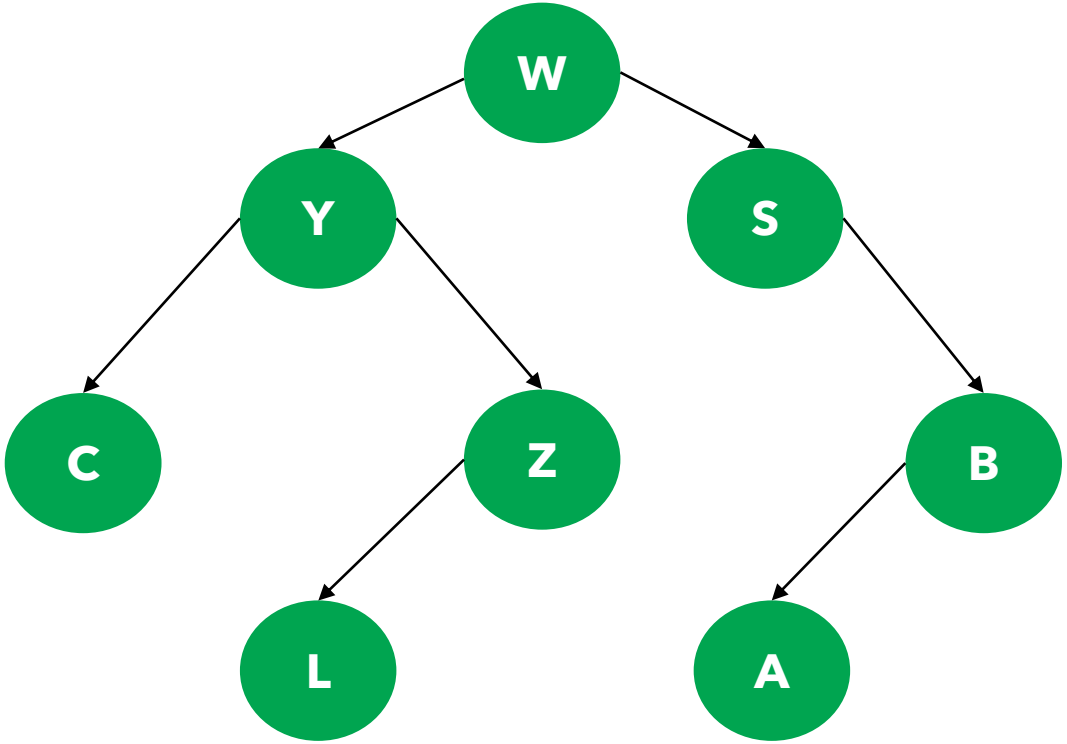
Breadth-first search (BFS)




```
dequeue();

Q: [A]
```

Breadth-first search (BFS)




```
dequeue();

Q: []
```