alpha ed/tech

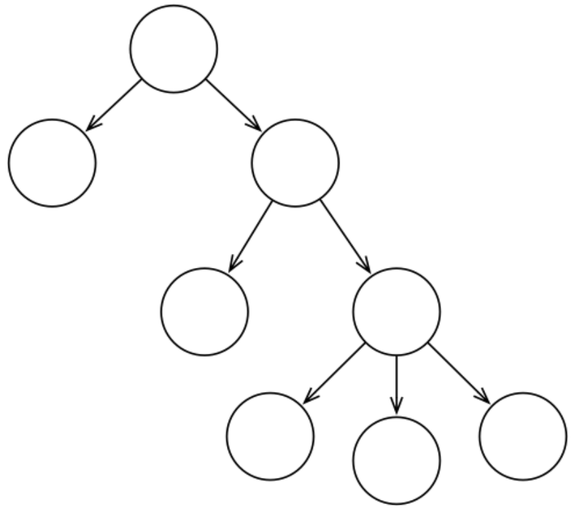
ESTRUTURA DE DADOS

MÓDULO 07 | AULA 05

alpha
<ed/tech>

ÁRVORES

Árvores são estruturas de dados hierárquicas. Basicamente, árvores são formadas por um conjunto de elementos, os quais chamamos nodos (ou vértices) conectados de forma específica por um conjunto de arestas. Um dos nodos, que dizemos estar no nível 0, é a raiz da árvore, e está no topo da hierarquia. A raiz está conectada a outros nodos, que estão no nível 1, que por sua vez estão conectados a outros nodos, no nível 2, e assim por diante.



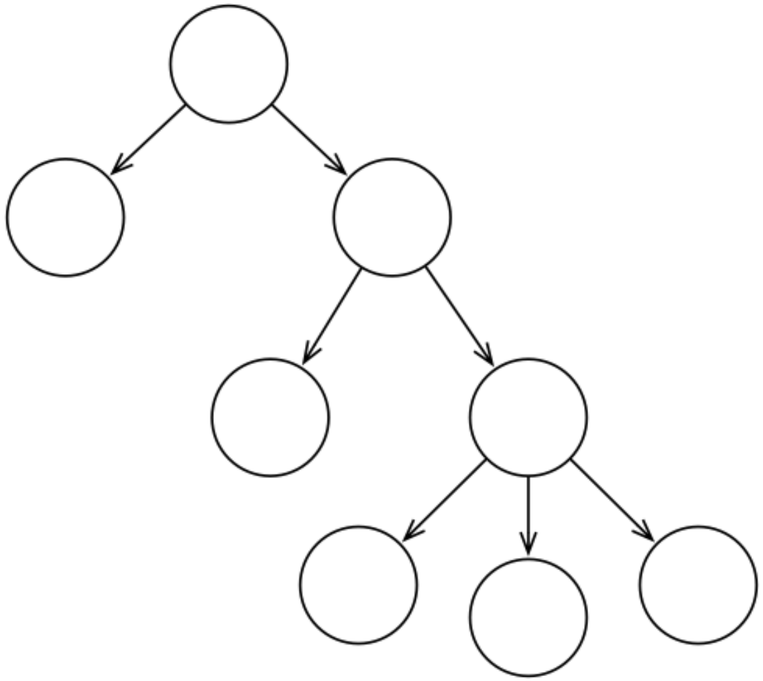
As conexões entre os nodos de uma árvore seguem uma nomenclatura genealógica. Um nodo em um dado nível está conectado a seus filhos (no nível abaixo) e a seu pai (no nível acima). A raiz da árvore, que está no nível 0, possui filhos mas não possui pai.

Árvores podem ser desenhadas de muitas formas, mas a convenção em Computação é desenhá-las com a raíz no topo, apesar de isso ser um pouco contra-intuitivo de acordo com nossa noção de árvore do cotidiano.

alpha
<ed/tech>

ÁRVORES BINÁRIAS

Árvores binárias são árvores nas quais cada nodo pode ter no máximo dois filhos, conforme mostrado na figura abaixo.



Uma árvore binária pode ser definida de forma recursiva, de acordo com o raciocínio a seguir. A raiz da árvore possui dois filhos, um à direita e outro à esquerda, que por sua vez são raizes de duas sub-árvores. Cada uma dessas sub-árvores possui uma sub-árvore esquerda e uma sub-árvore direita, seguindo esse mesmo raciocínio.

alpha
<ed/tech>

ÁRVORES BINÁRIAS

Representação de Árvores Binárias

Na prática, os nodos de uma árvore binária possuem um valor (chamado de chave) e dois apontadores, um para o filho da esquerda e outro para o filho da direita. Esses apontadores representam as ligações (arestas) de uma árvore.

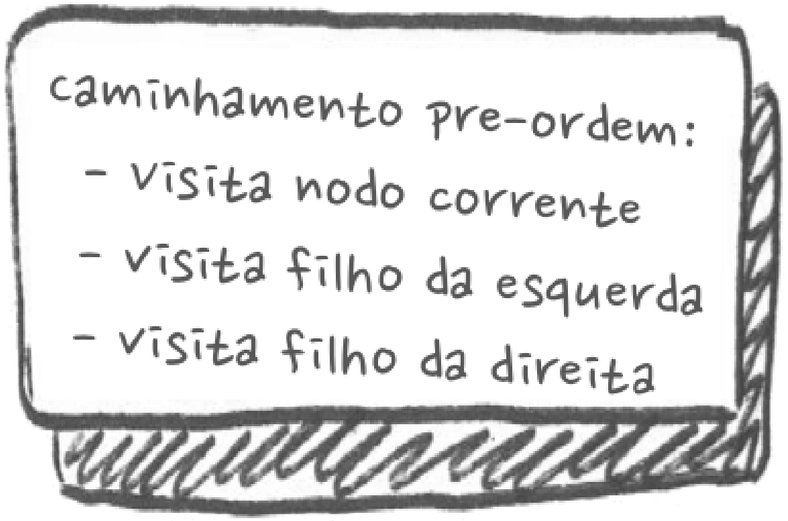
```
class NodoArvore:
    def __init__(self, chave=None, esquerda=None, direita=None):
        self.chave = chave
        self.esquerda = esquerda
        self.direita = direita

    def __repr__(self):
        return '%s <- %s -> %s' % (self.esquerda and self.esquerda.chave,
                                   self.chave,
                                   self.direita and self.direita.chave)
```

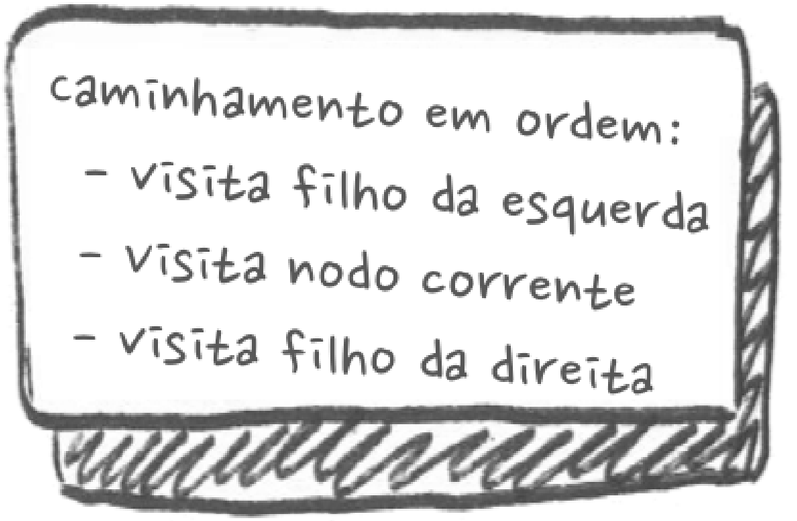
```
raiz = NodoArvore(3)
raiz.esquerda = NodoArvore(5)
raiz.direita  = NodoArvore(1)
print("Árvore: ", raiz)
```

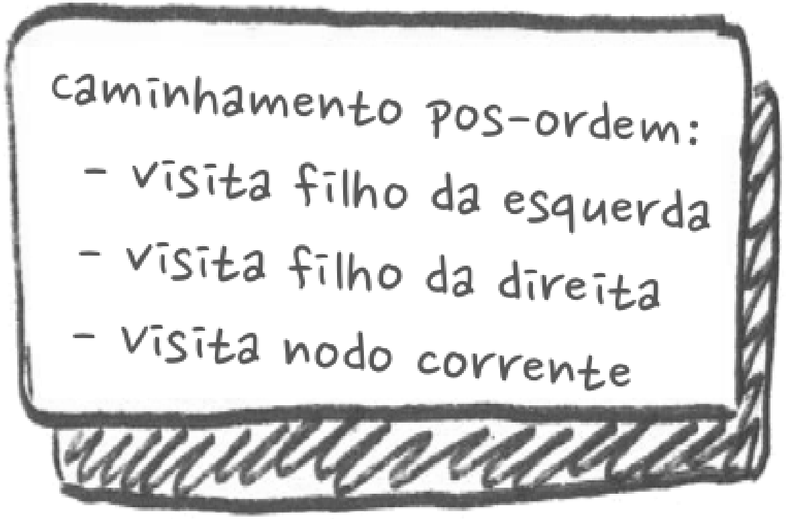
alpha <ed/tech>

ÁRVORES BINÁRIAS

Caminhamento em Árvores Binárias





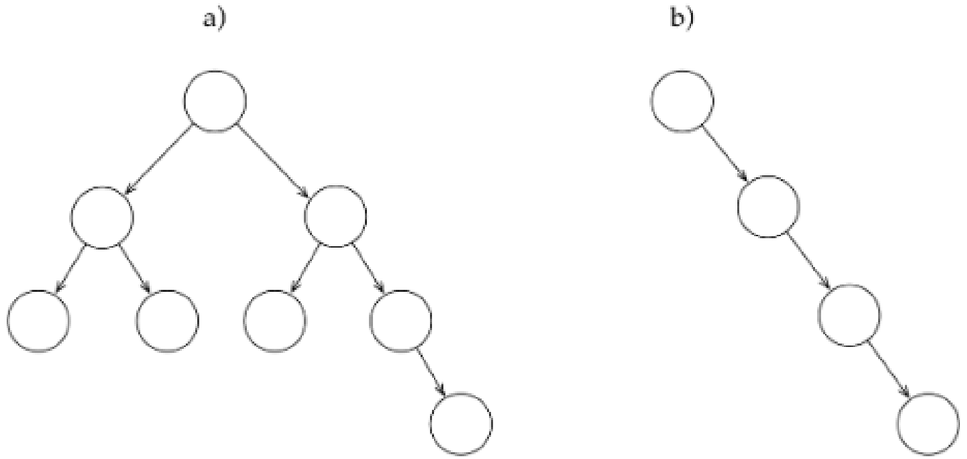


alpha
<ed/tech>

ÁRVORES BINÁRIAS

Definição: Uma árvore binária é balanceada se a diferença da profundidade de duas folhas quaisquer é no máximo 1. A profundidade de um nodo é o número de níveis da raiz até aquele nodo.

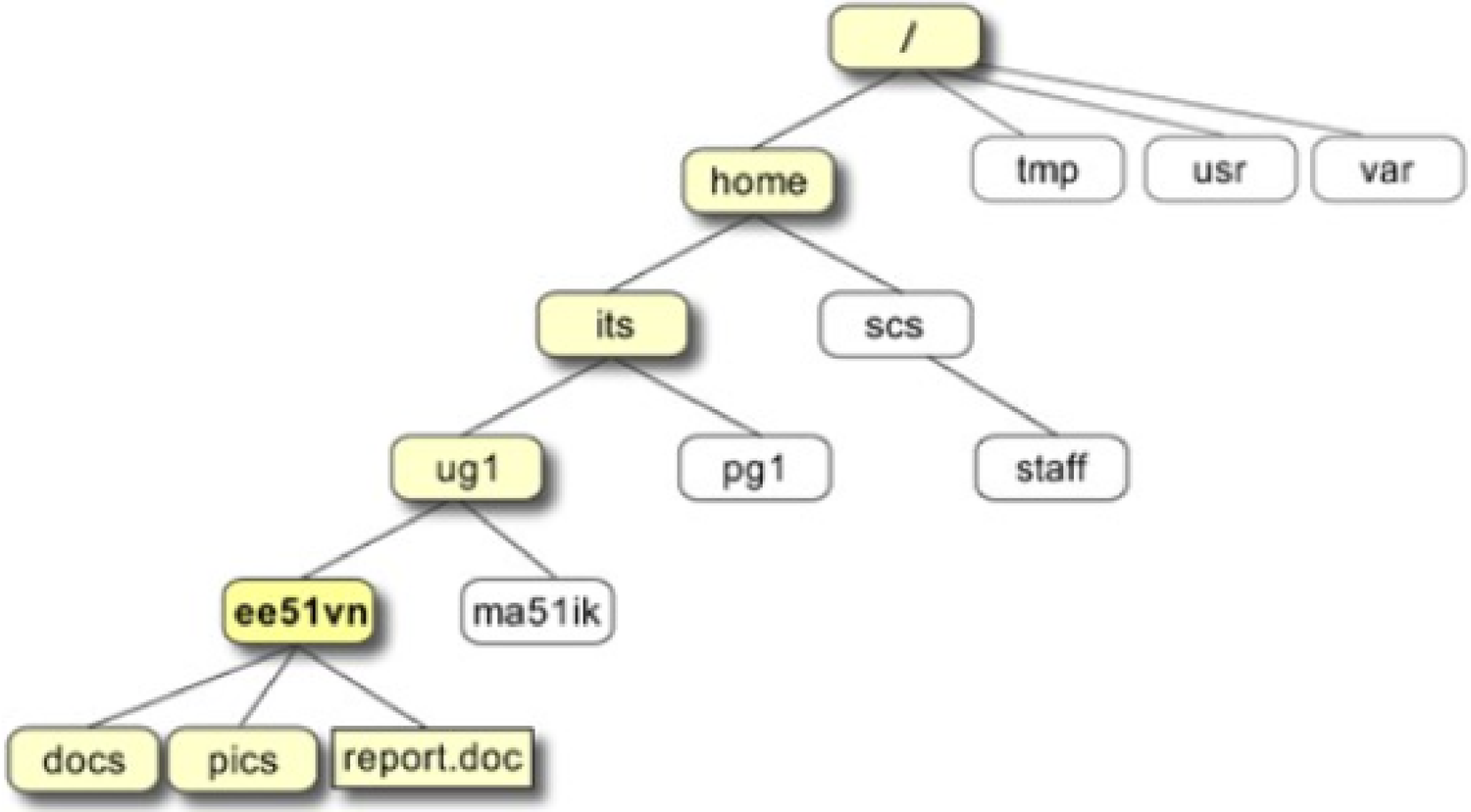
Na figura ao lado, a árvore a) é balanceada, e a árvore b) não é balanceada.



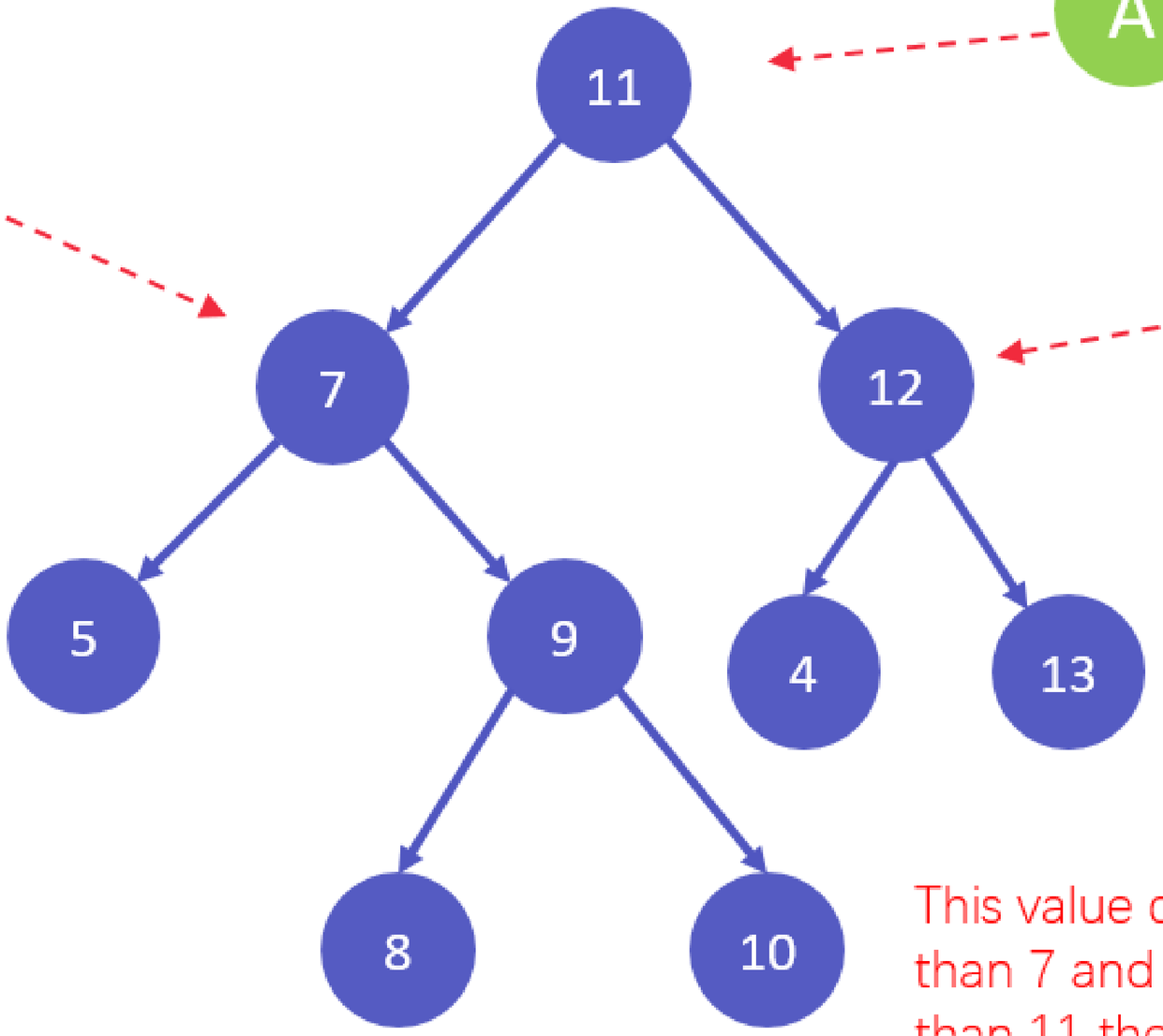
alpha <ed/tech>

ÁRVORES BINÁRIAS

Um exemplo da implementação de uma árvore binária é o Sistema de Organização de Arquivos.



CONCEPT DIAGRAM



A

This is the main node or top-level node. It has left and right sub-trees

B

Right sub-tree has value greater than node

C

Left sub-tree has value less than node

This value cannot be 8, or 9 or 10 because then it will be greater than 7, hence not fulfilling the condition of left sub tree

This value can be greater than 7 and 9 but not greater than 11 the top node

Search Operation

A

Elements to be searched in the tree 10

B

10 < 12 so move to the left sub-tree

No need to search in right sub-tree

C

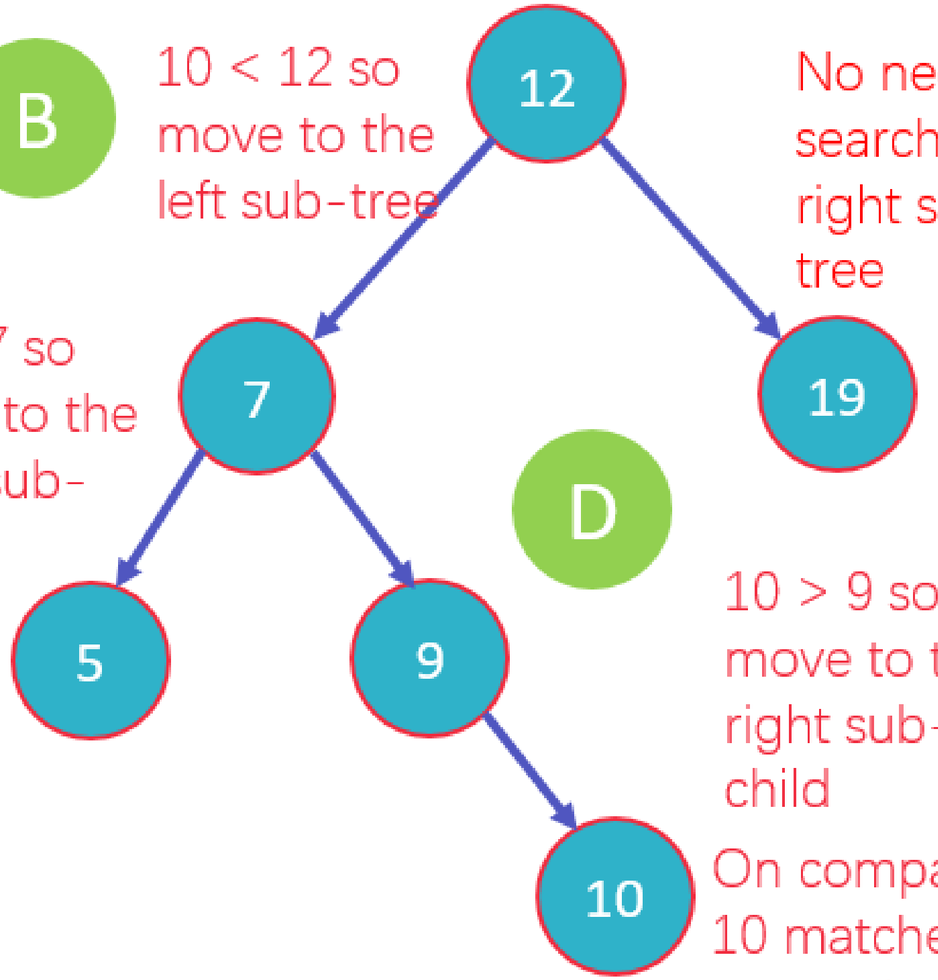
10 > 7 so move to the right sub-tree

D

10 > 9 so move to the right sub-tree child

E

On comparison 10 matches, return the value



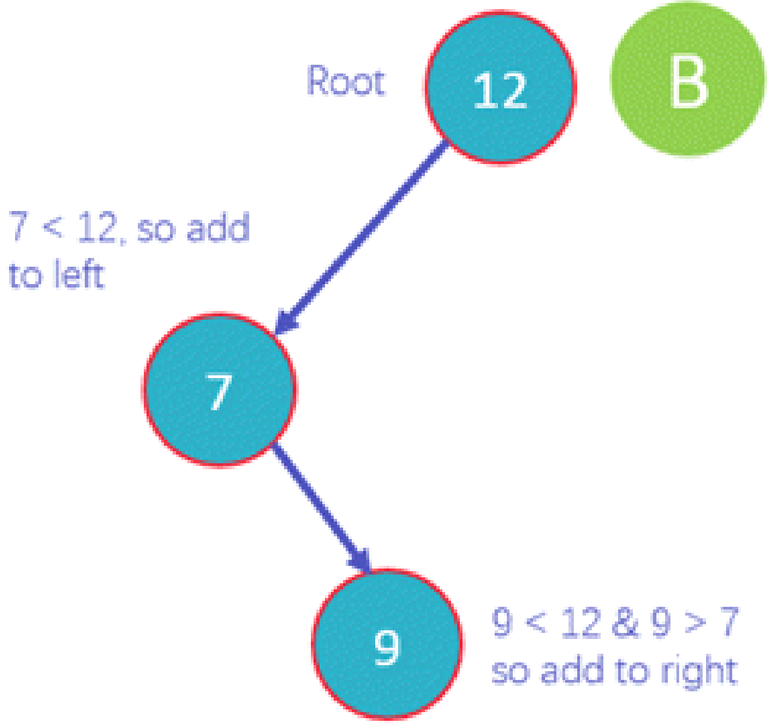
Insert Operation

A

Elements to be inserted in the tree from left to right:
12, 7, 9, 19, 5, 10

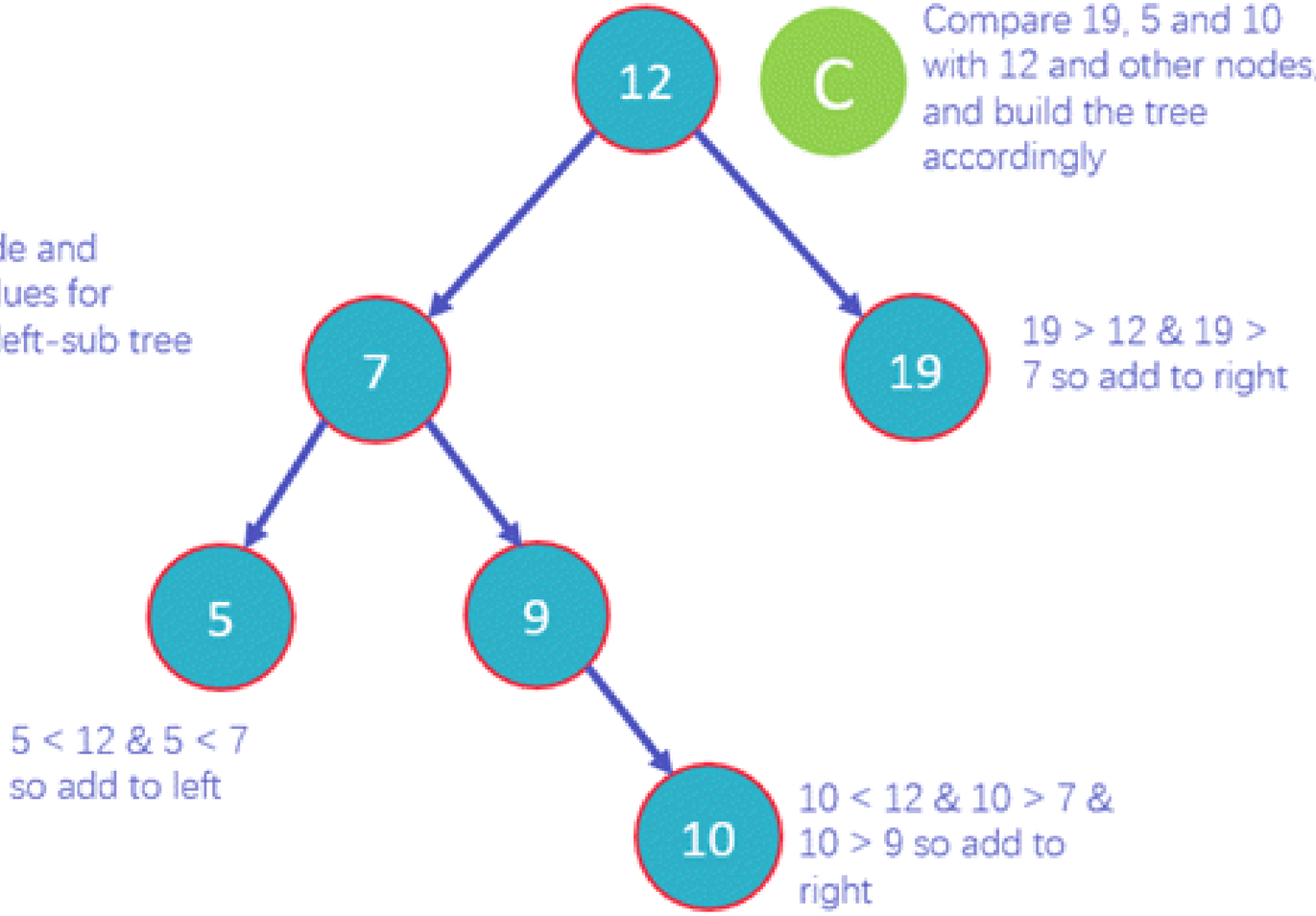
B

Insert 12 as root node and compare 7 and 9 values for inserting to right or left-sub tree

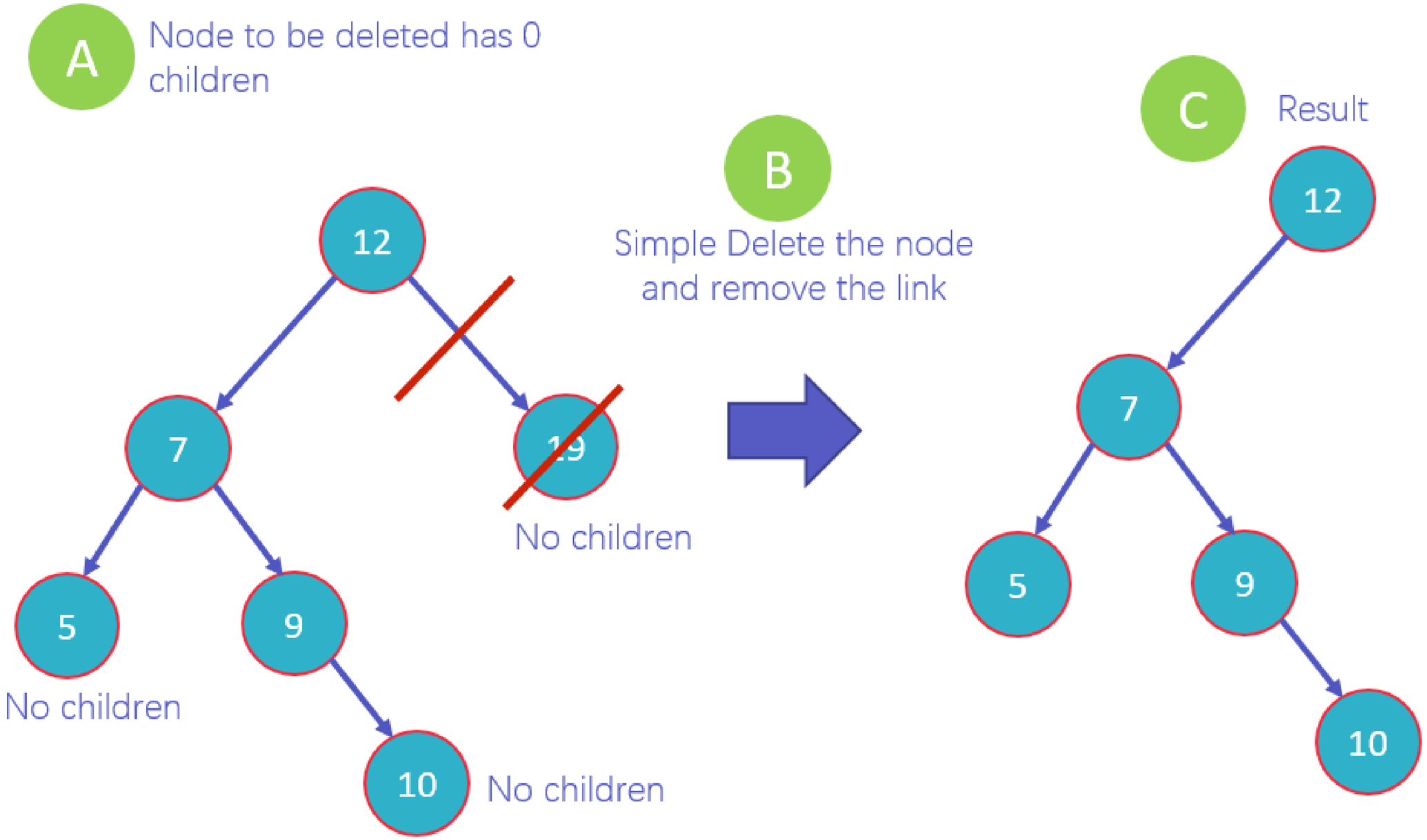


C

Compare 19, 5 and 10 with 12 and other nodes, and build the tree accordingly



Delete Operation – Case 1

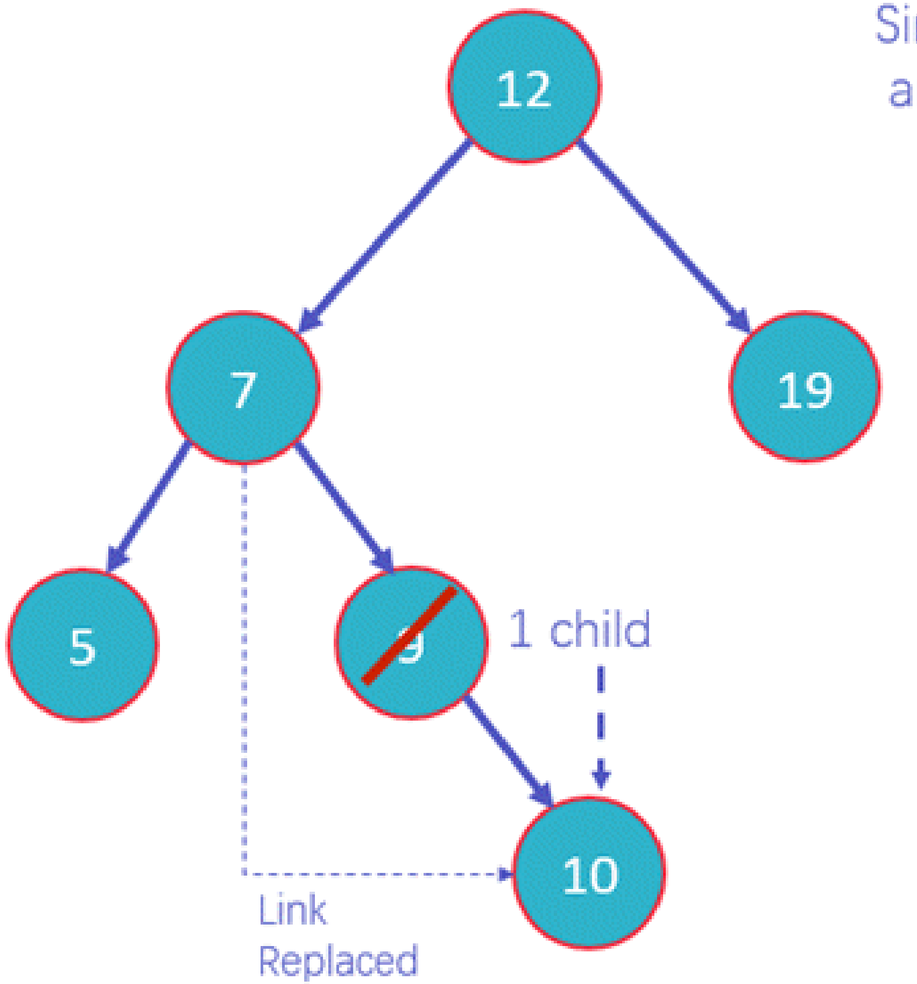


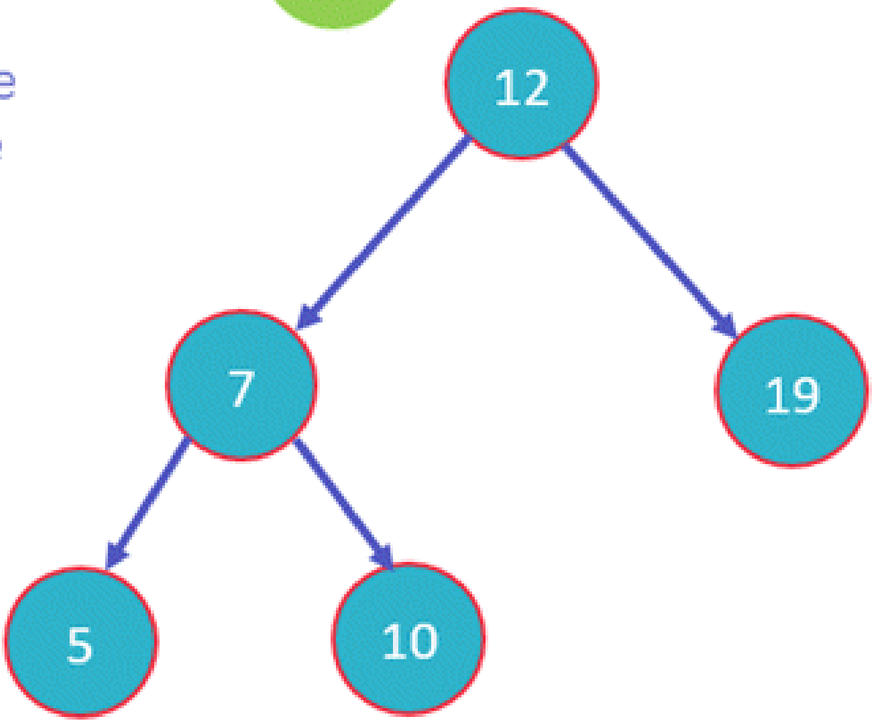
Delete Operation – Case 2

A Node to be deleted has 1 child

B Simple Delete the node and replace it with the child node

C Result





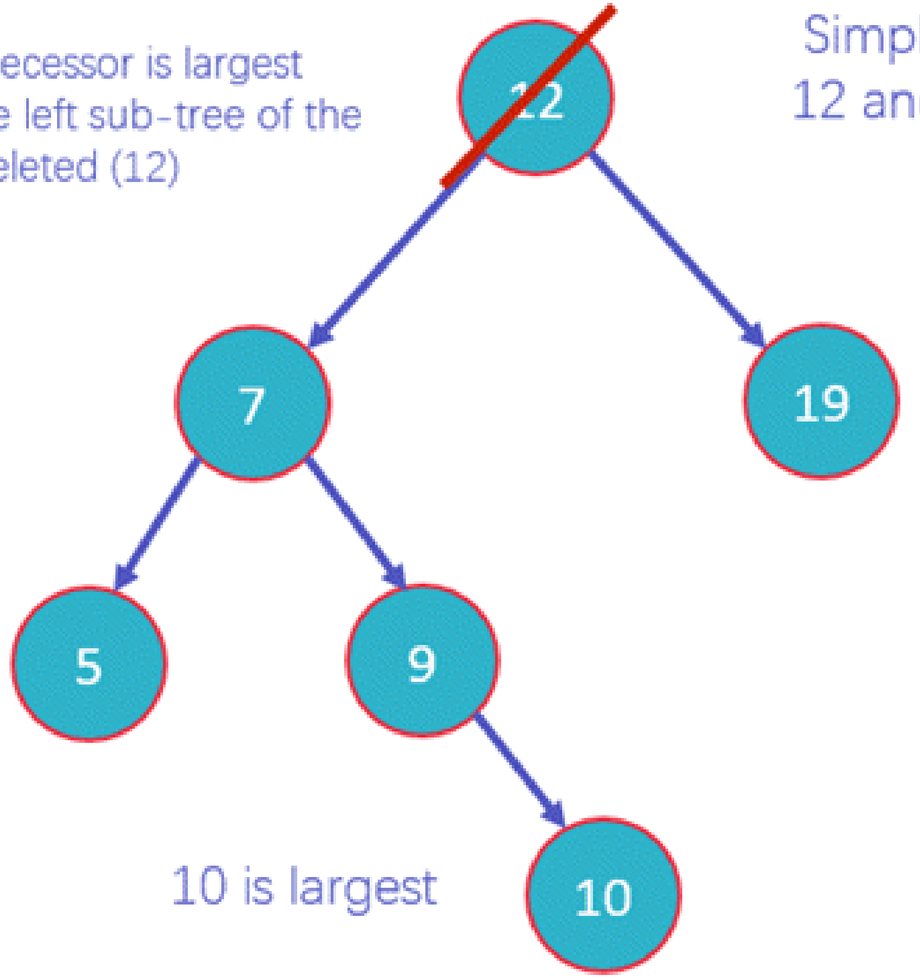
Delete Operation – Case 3 (a)

A

Node to be deleted has 2 child
Replace Situation: In Order Predecessor

In Order predecessor is largest element in the left sub-tree of the node to be deleted (12)

B

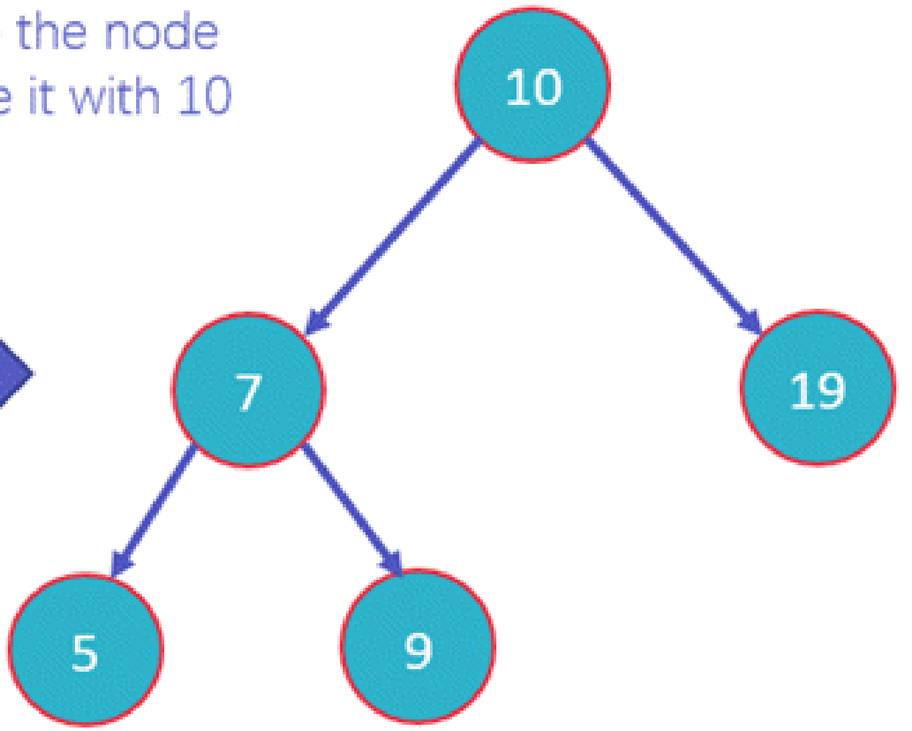


C

Simple Delete the node 12 and replace it with 10

D

Result



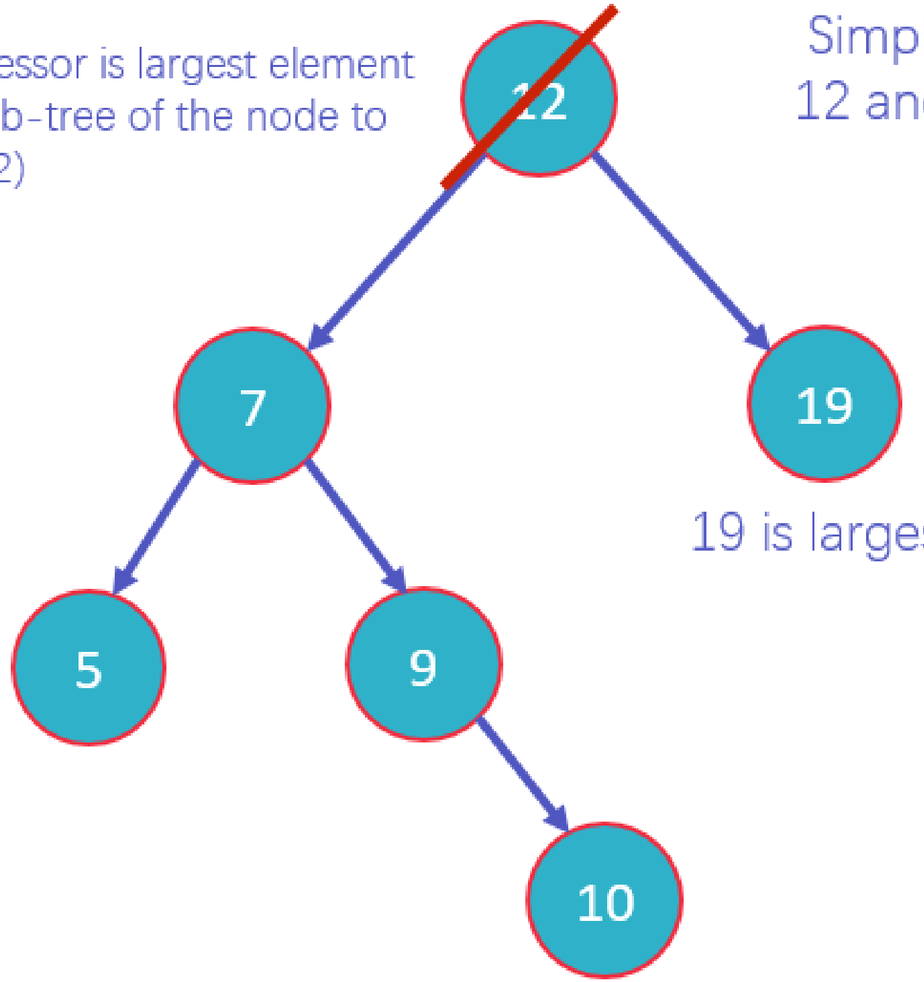
Delete Operation – Case 3 (b)

A

Node to be deleted has 2 child
Replace Situation: In Order Successor

B

In Order successor is largest element in the right sub-tree of the node to be deleted (12)



19 is largest

C

Simple Delete the node 12 and replace it with 19

D

Result