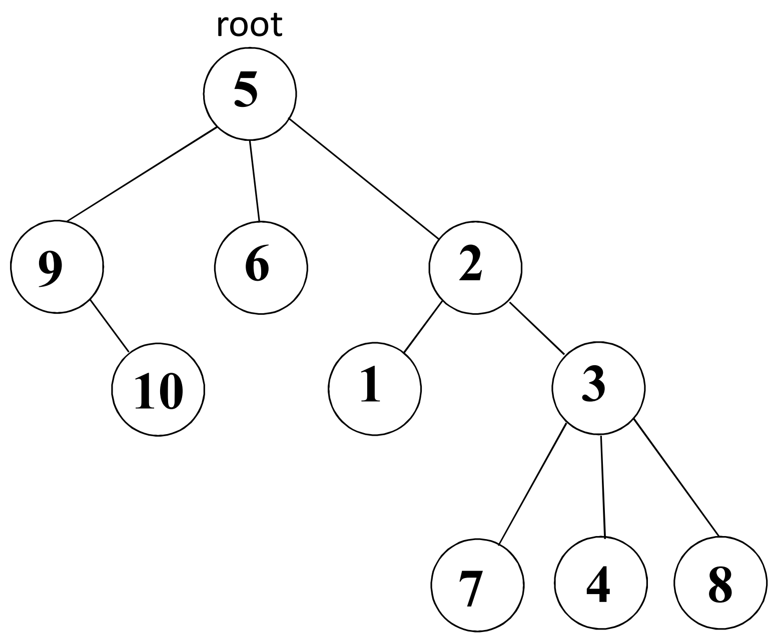
Rooted Tree Implementation and Traversal

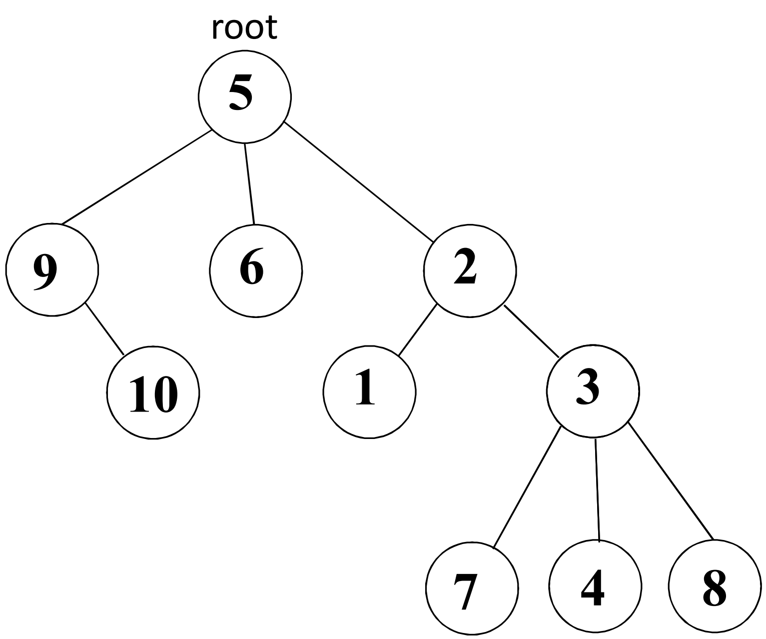
CSCI2100

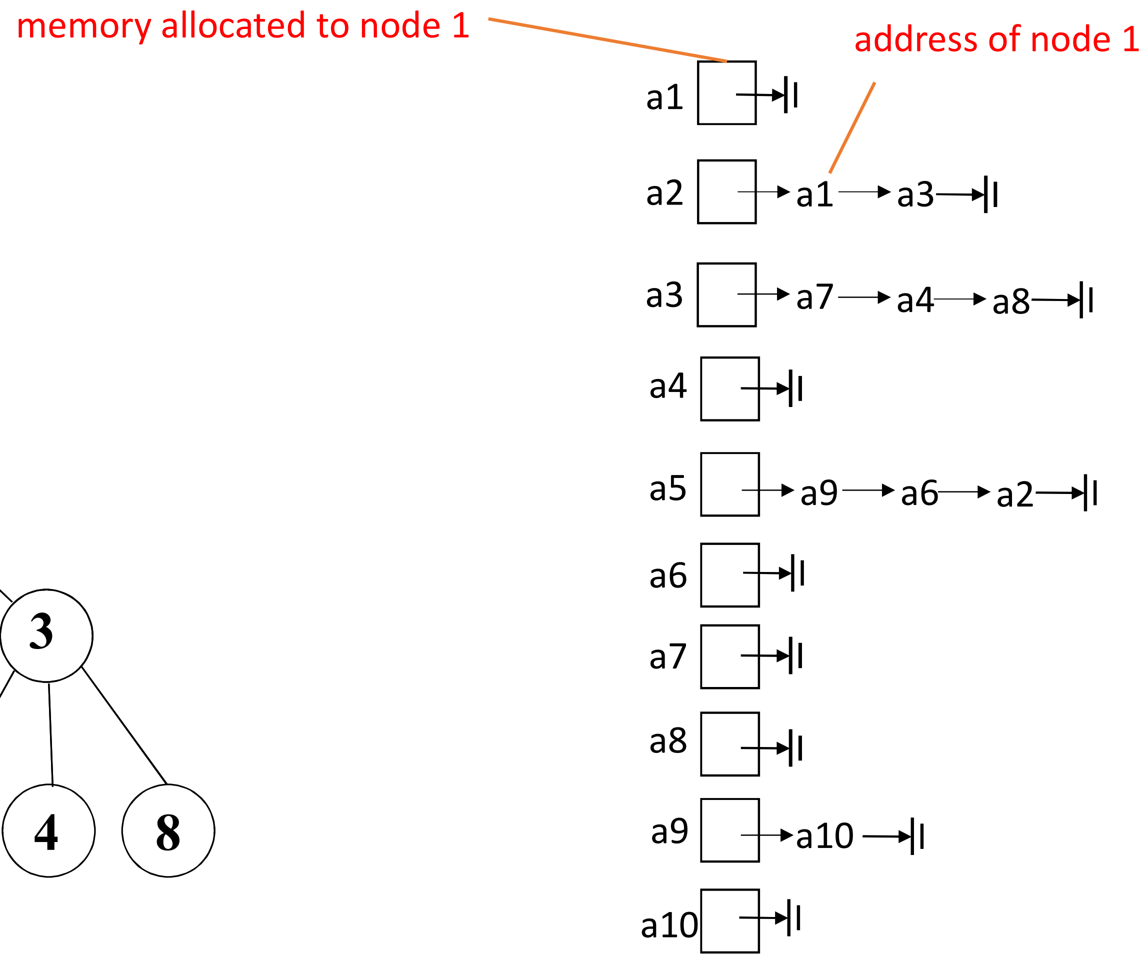
Tutorial 8

How to store a rooted tree in memory?



Storing a tree





For each node, create a linked list of pointers (one per child).

In general, the space of storing n nodes is $O(n)$.

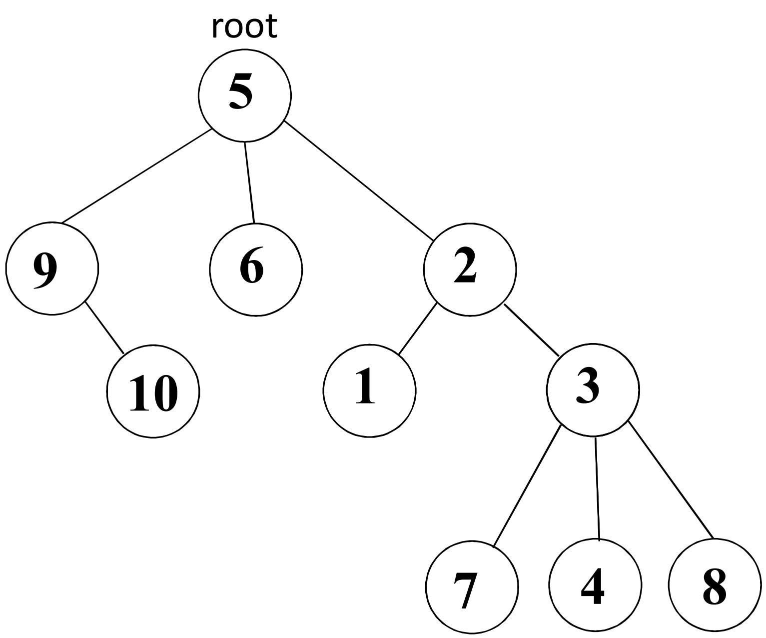
Rooted tree traversal

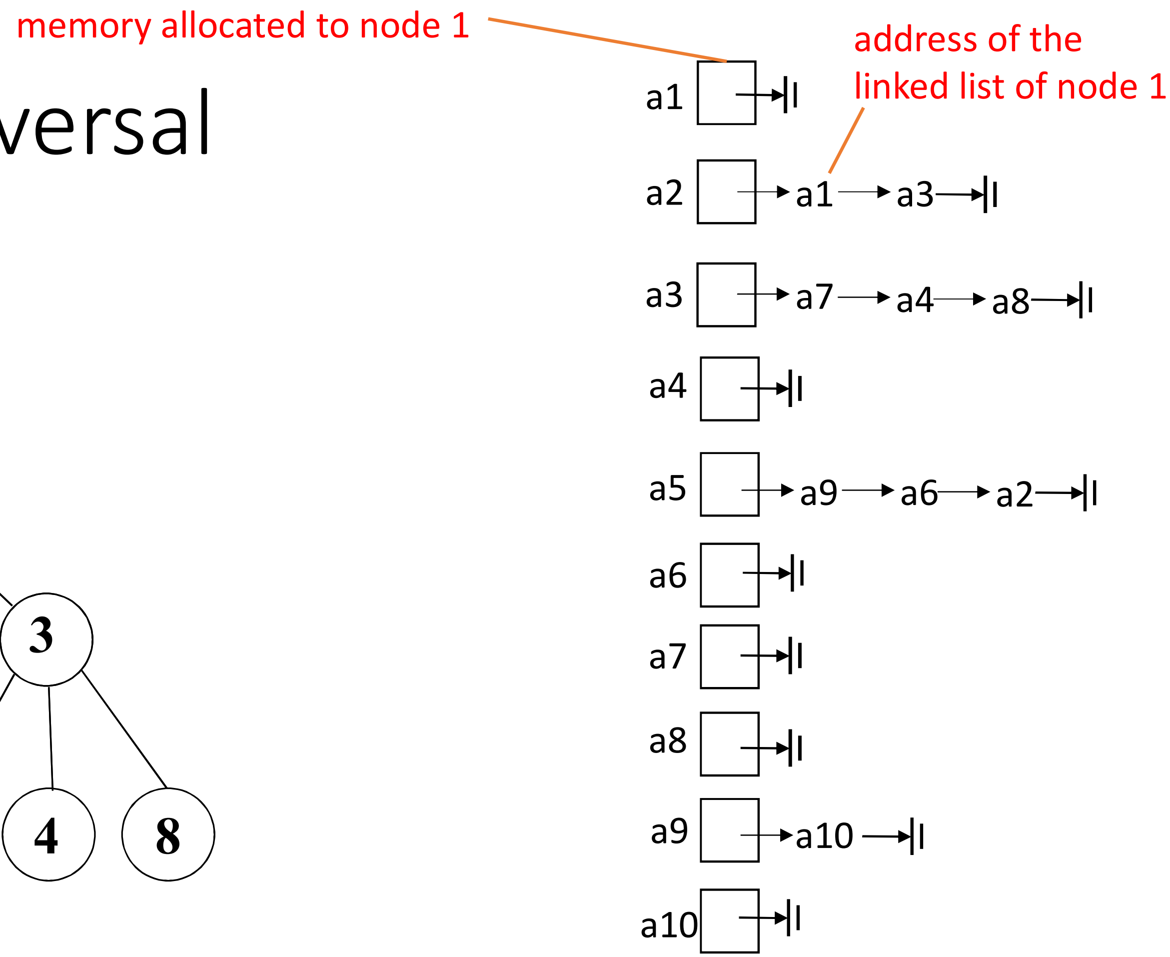
Problem: Given the root of a tree, count the number of nodes in the tree.

Goal: O(n) time.

We will achieve the purpose by giving an algorithm to traverse the tree.

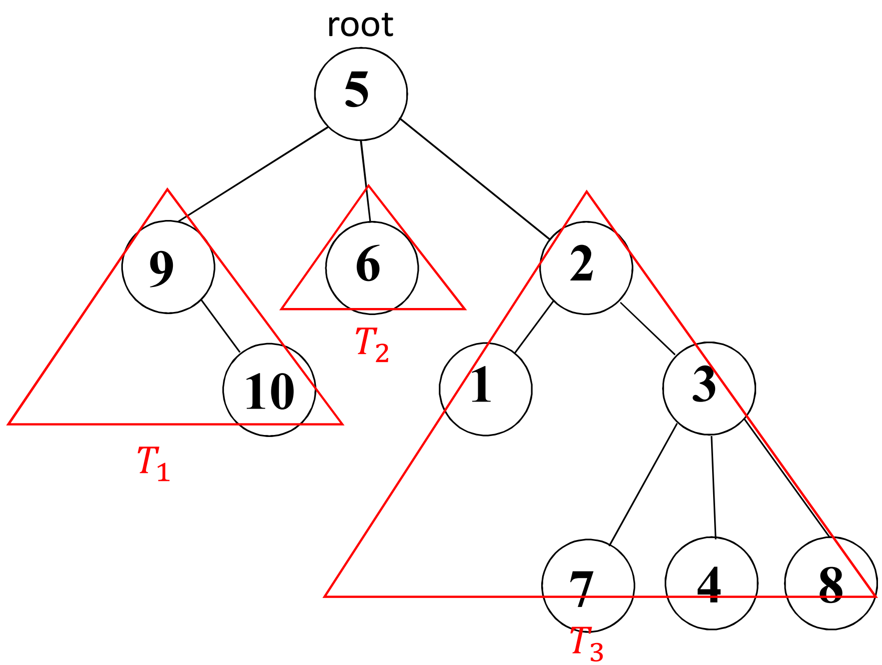
Rooted tree traversal





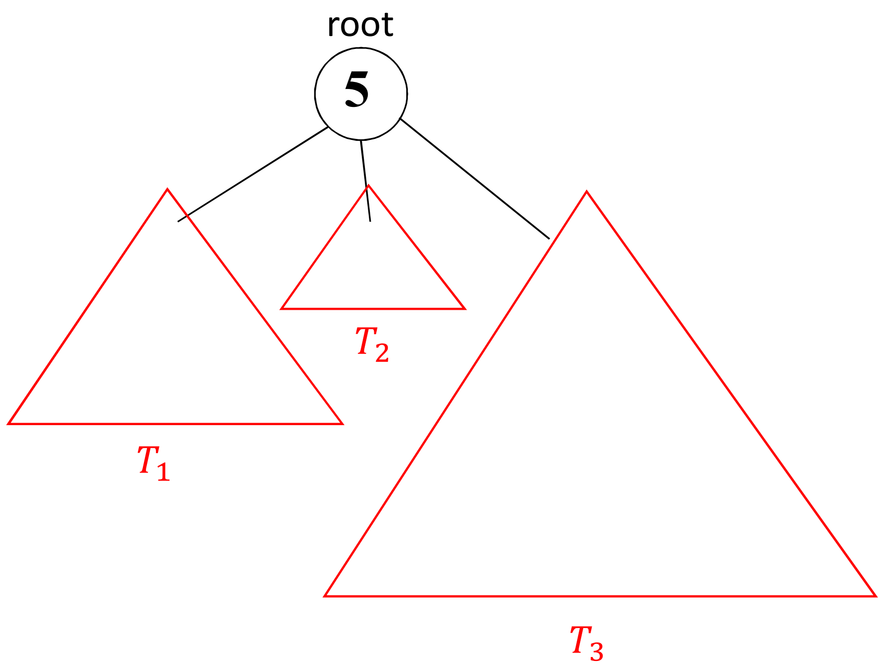
Given a5, how do we find the number of nodes in the tree?

A recursive view



- Recursively count the subtree of each child of the root.

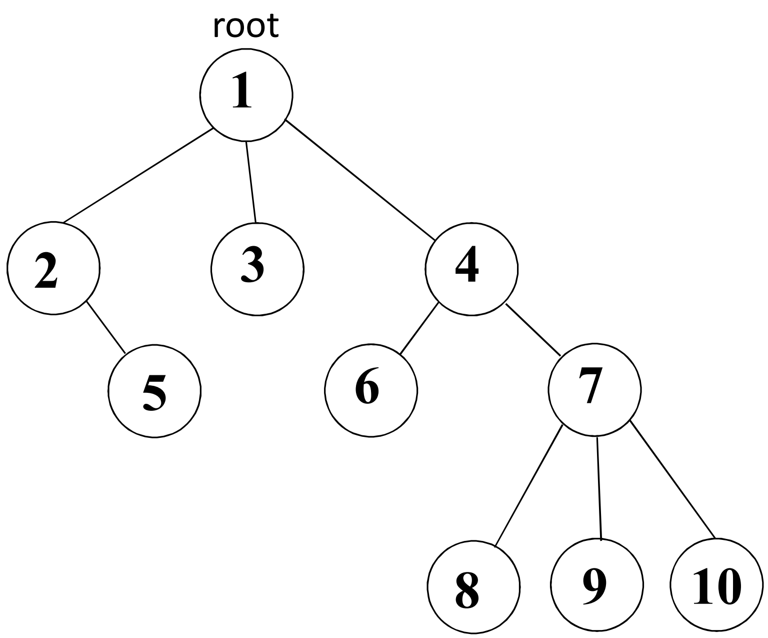
A recursive view



- Recursively count the subtree of each child of the root.

$\boldsymbol{count}(r)$:
  $result = 1$
  for each child u of r:
    $result = result + \boldsymbol{count}(u)$
  return $result$

Analysis – the smart way

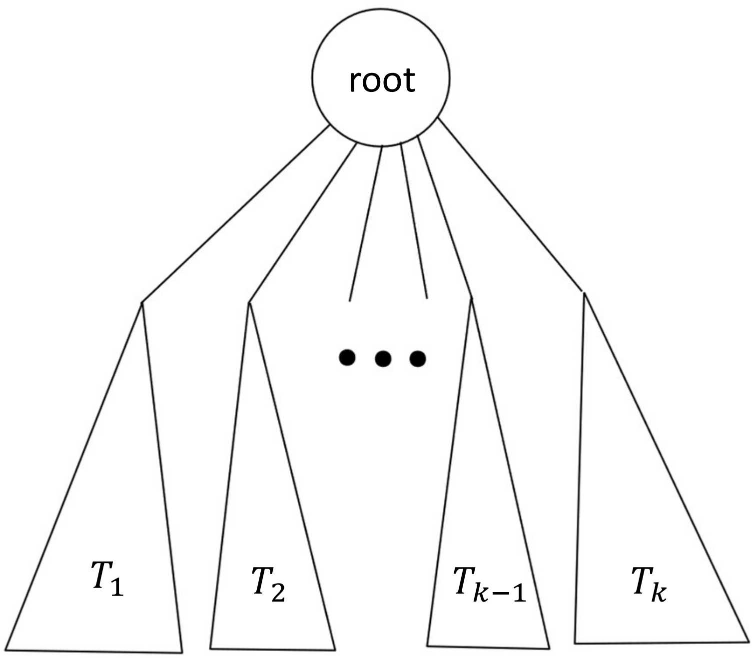


- Intuitively, we visit each edge twice (descending once and ascending once).

- So the cost is
  $$O(\#nodes\ +\ \#\ \text{edges}) = O(n)$$

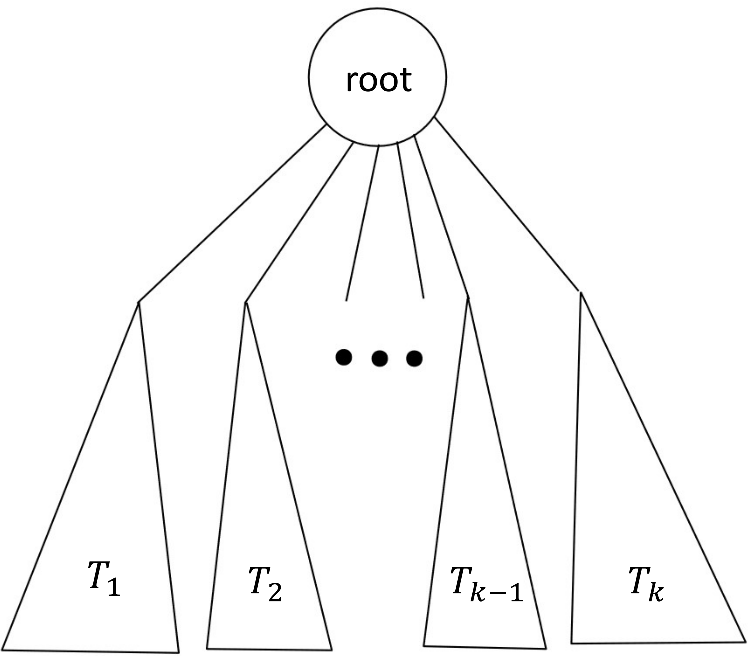
Analysis – the "standard" way

Let $f(n)$ denote the running time on a tree T of n nodes (we denote $|T|$ as the number of nodes in T).



$\boldsymbol{count}(r)$:
    $result = 1$
    for each child u of r:
        $result = result + \boldsymbol{count}(u)$
    return $result$

Analysis – the “standard” way



For $n = 1$, we have:

$$f(1) = O(1)$$

For $n \geq 2$:

$$f(n) \leq f(|T_1|) + \cdots + f(|T_k|) + O(k+1)$$

Where $T_1, T_2, \ldots, T_k$ are the subtrees at the child nodes of the root.

We can prove:

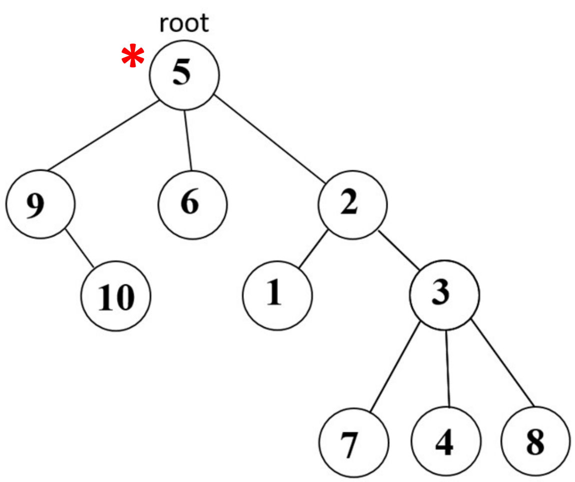
$$f(n) = O(n)$$

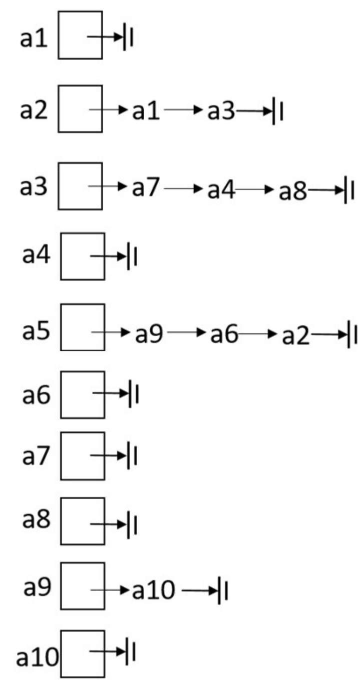
by the substitution method (left to you).

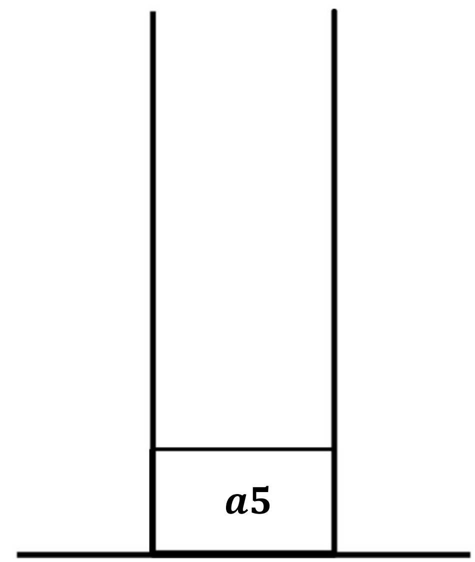
The recursive implementation may not work in today's operating systems

- Every operating system today limits the depth of recursion
  - Typically at the order of hundreds.
- Our earlier program will crash if the tree is too tall.
- Next, we will see a non-recursive implementation based on a stack.

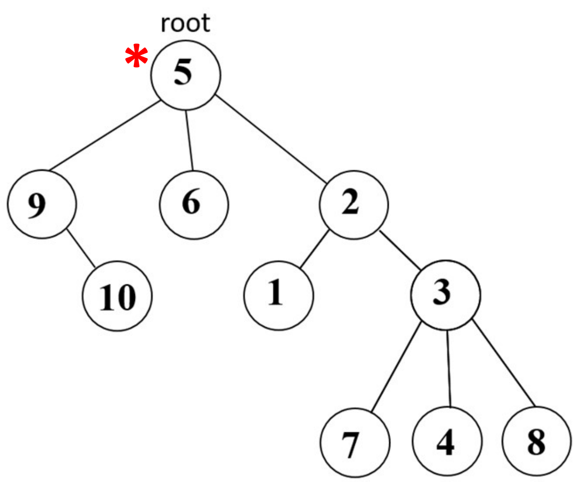
A stack-based implementation

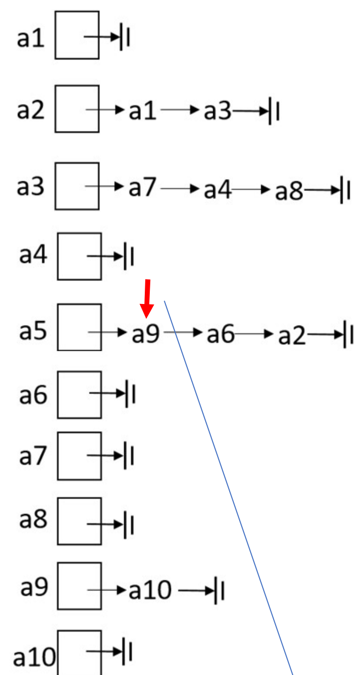


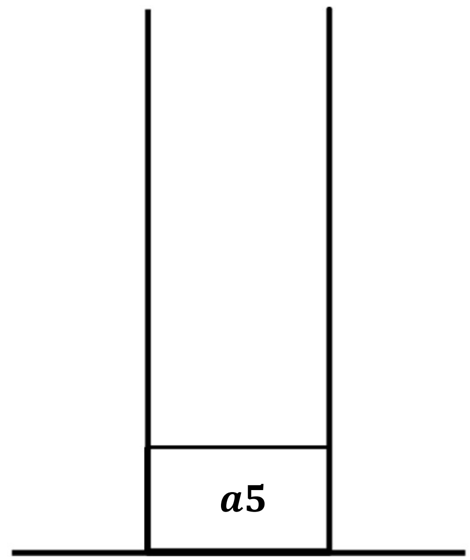




A stack-based implementation

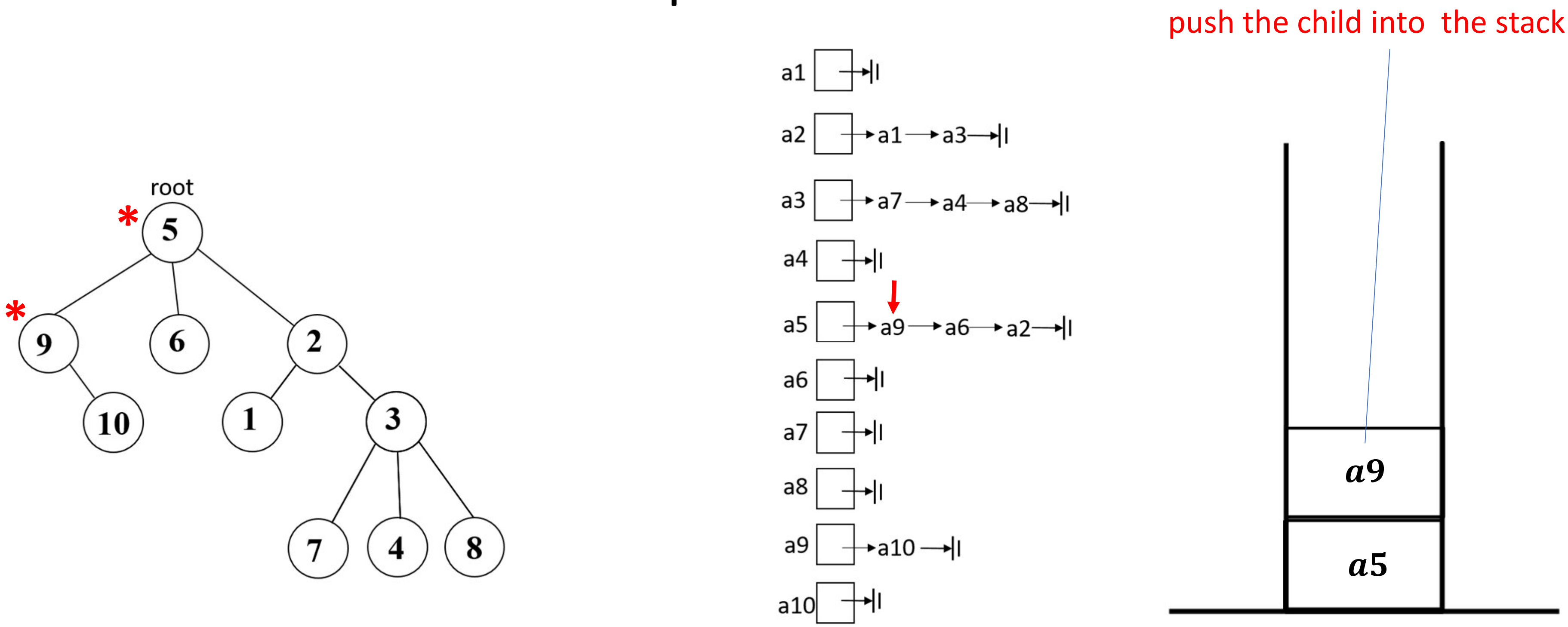




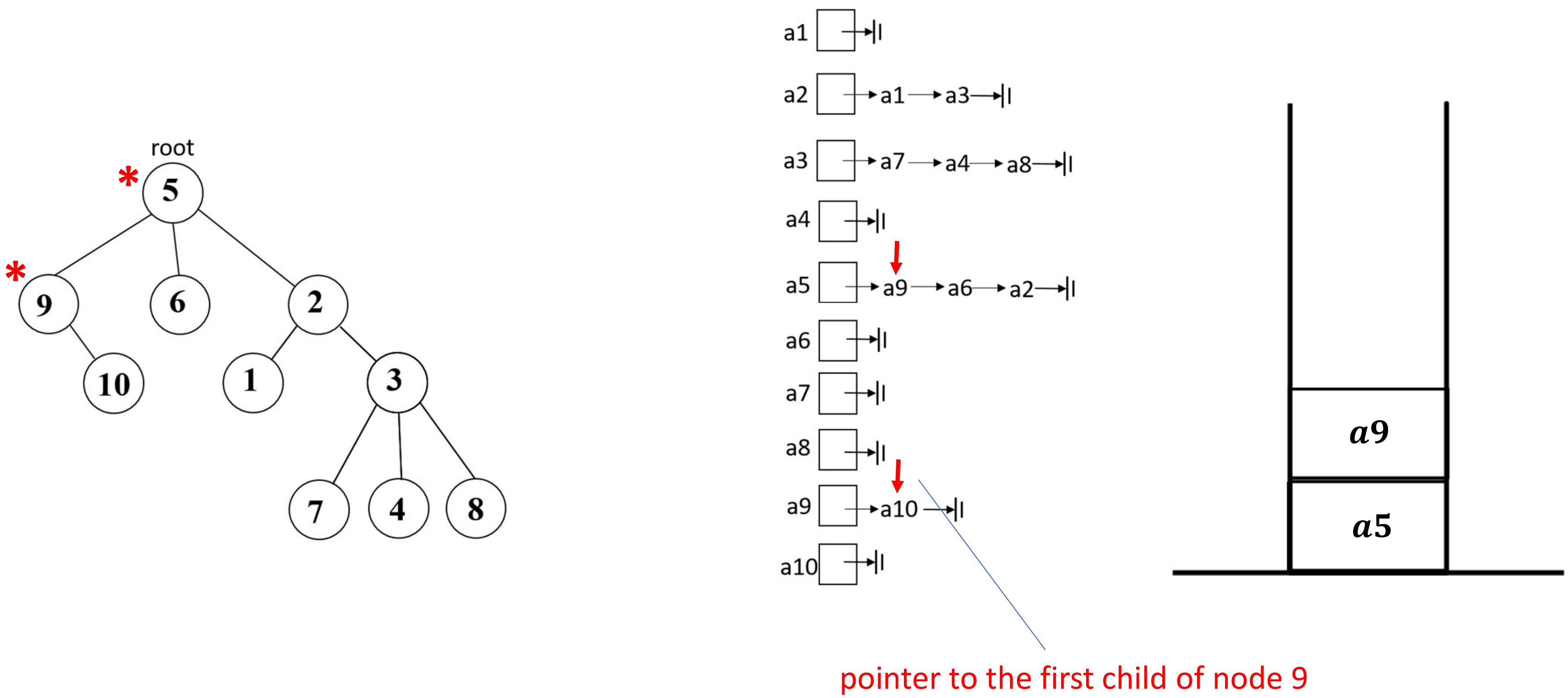


a pointer remembering the child under processing

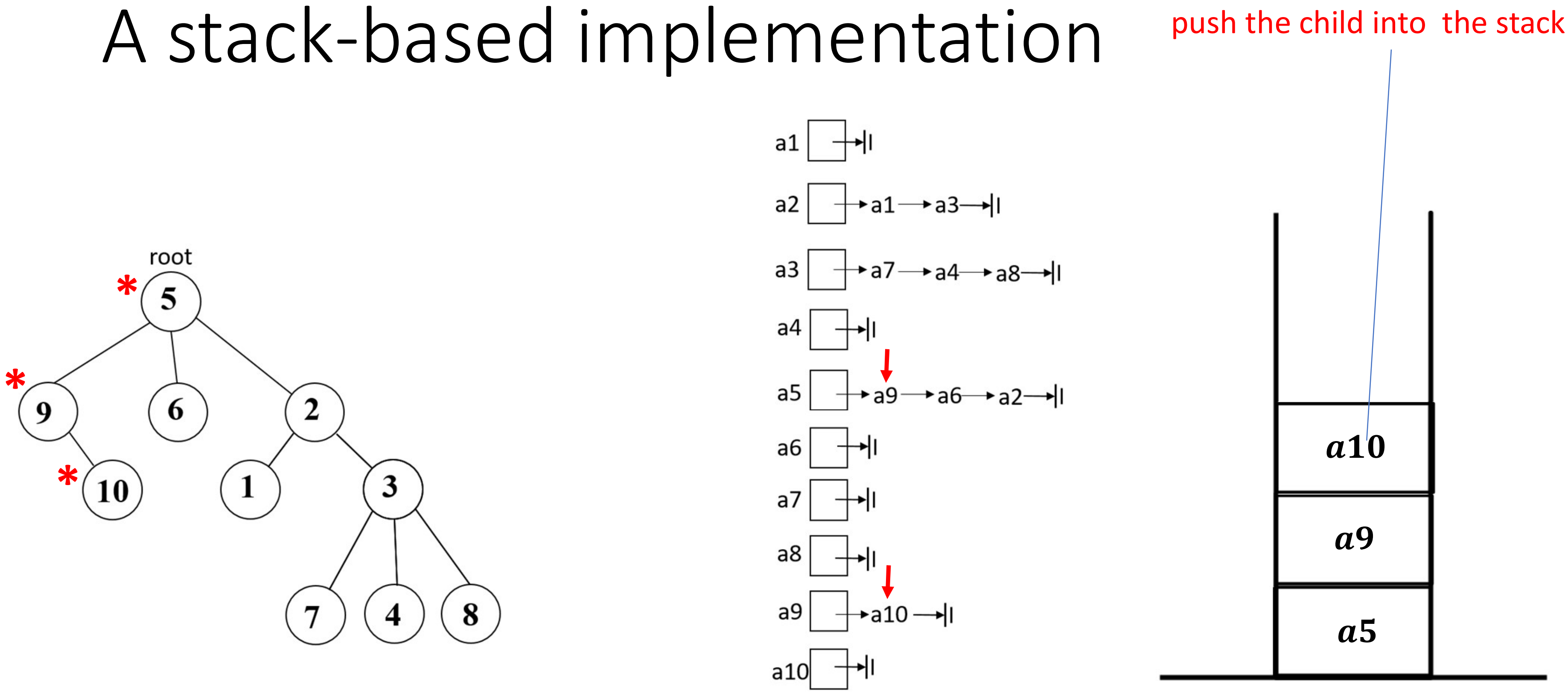
A stack-based implementation



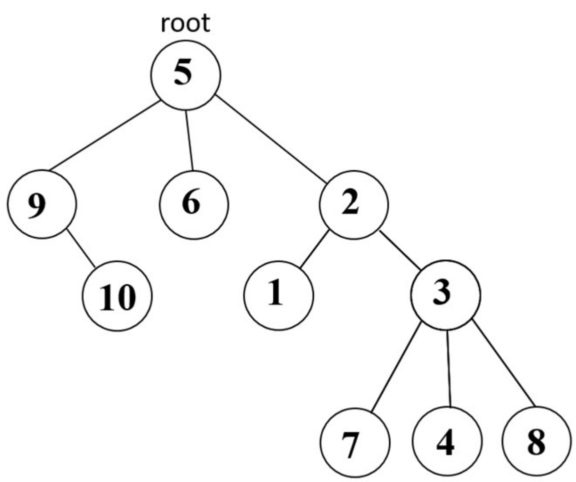
A stack-based implementation

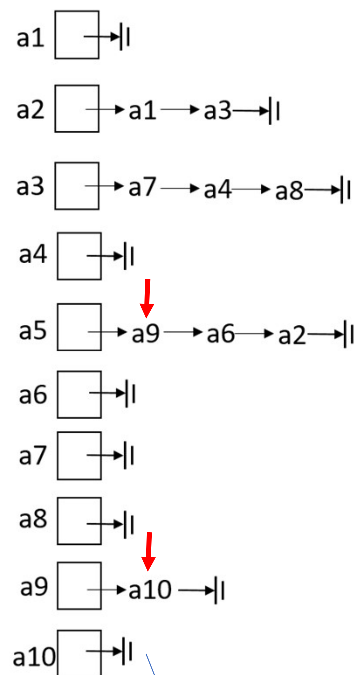


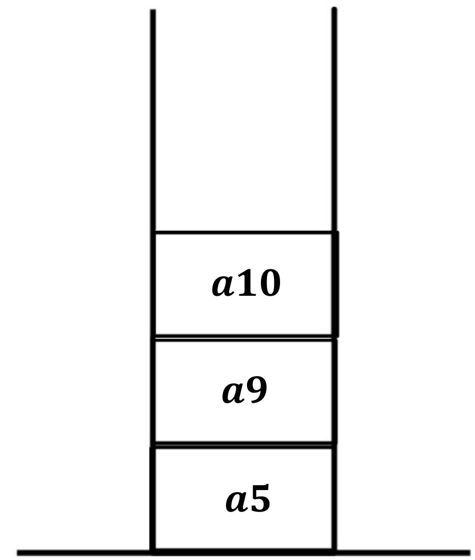
A stack-based implementation



A stack-based implementation

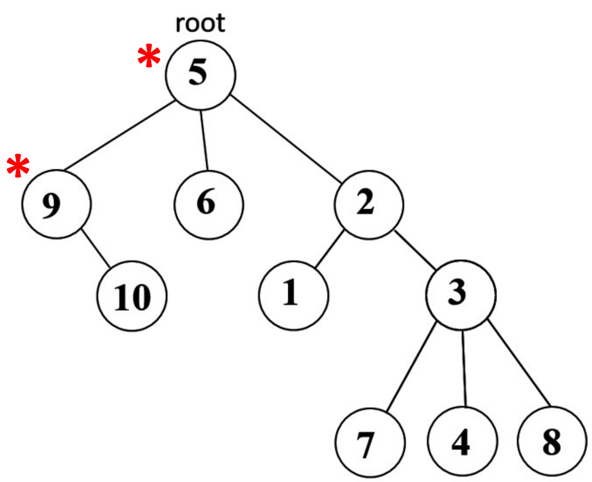


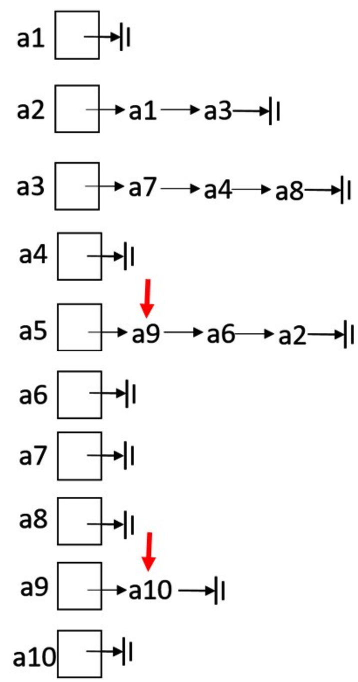




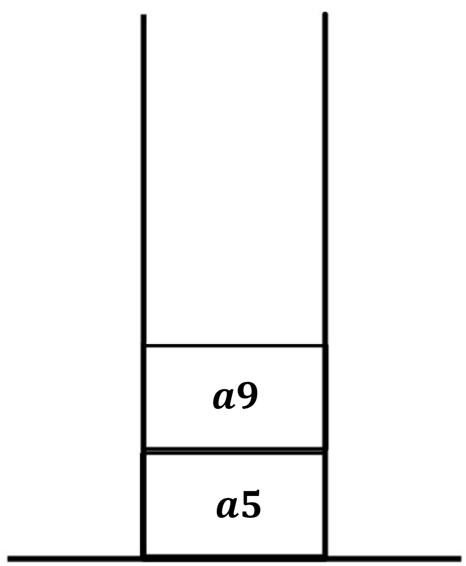
node 10 has no children.

A stack-based implementation

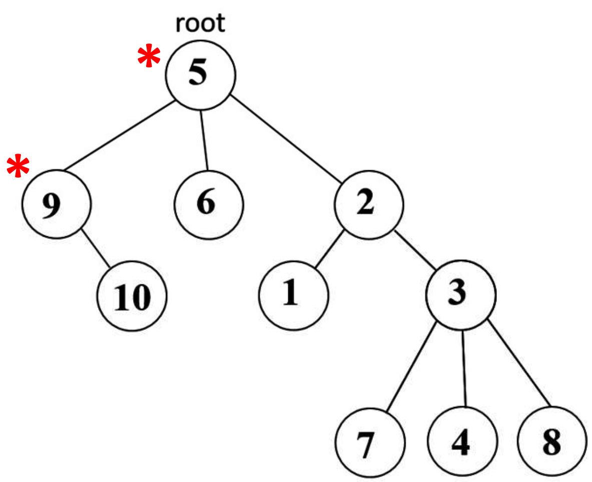


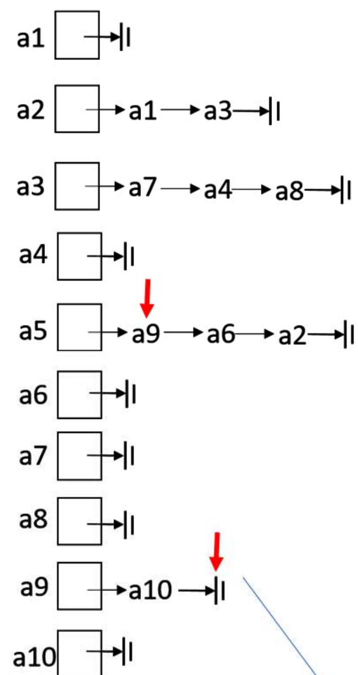


a10 has been popped

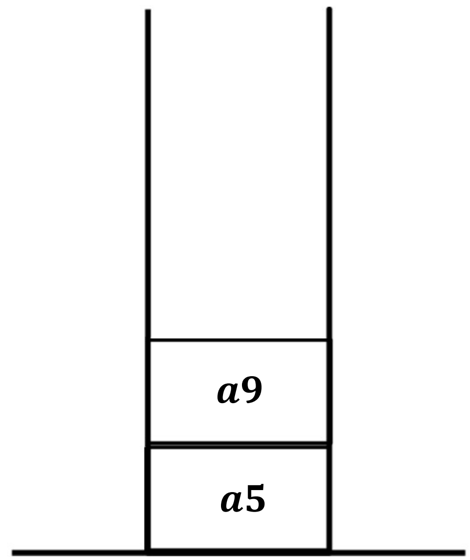


A stack-based implementation





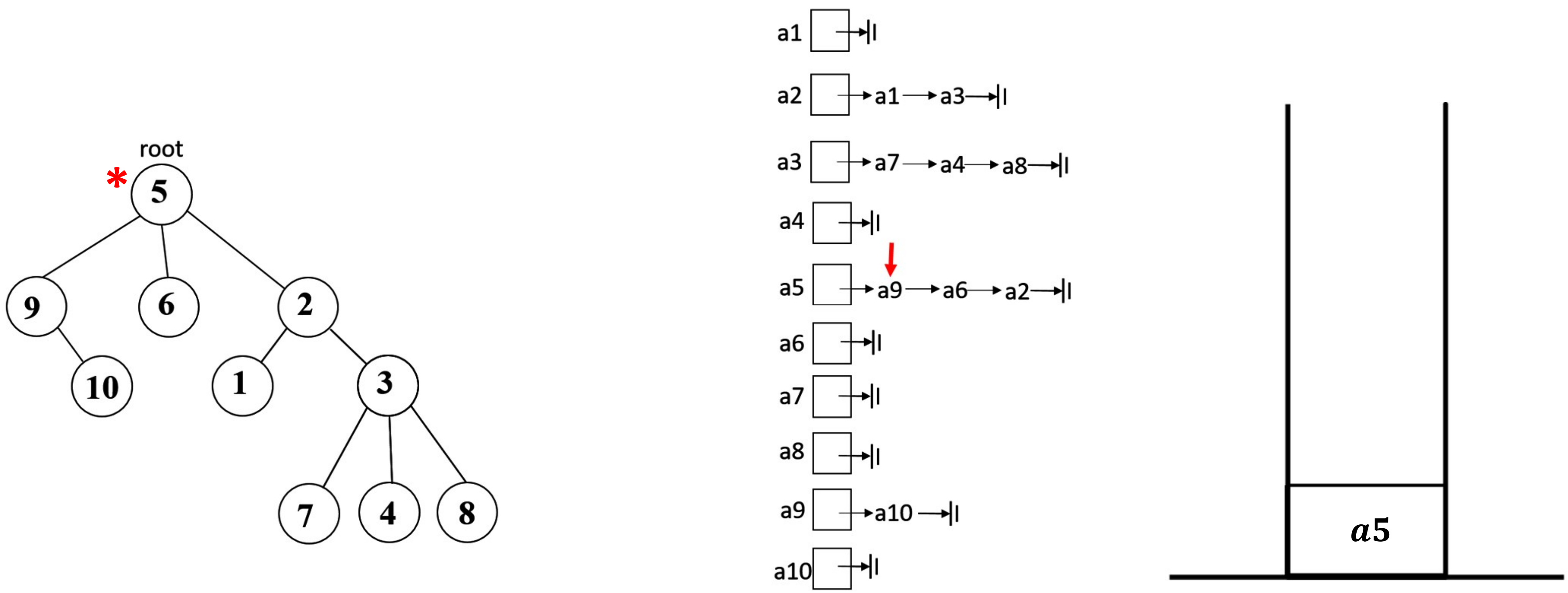
a10 has been popped



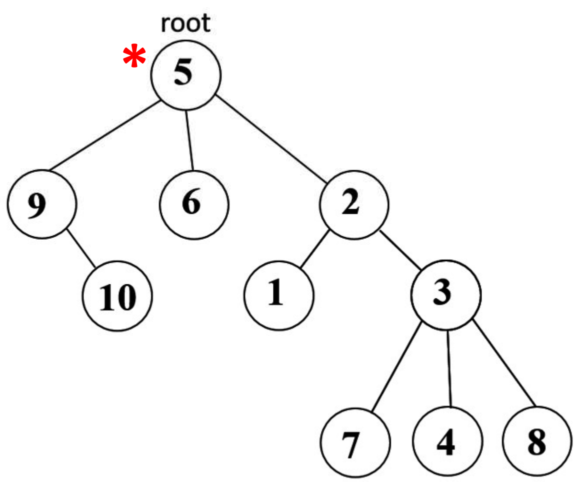
node 9 has no more children.

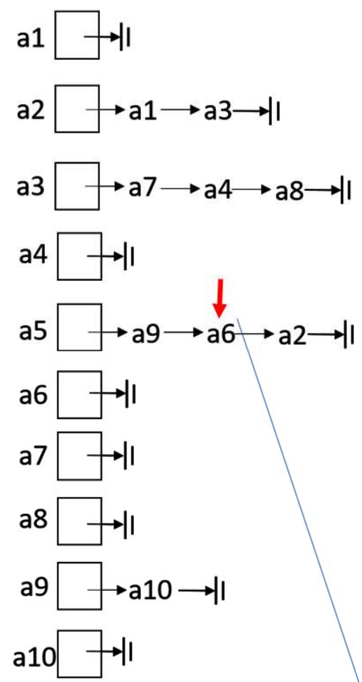
A stack-based implementation

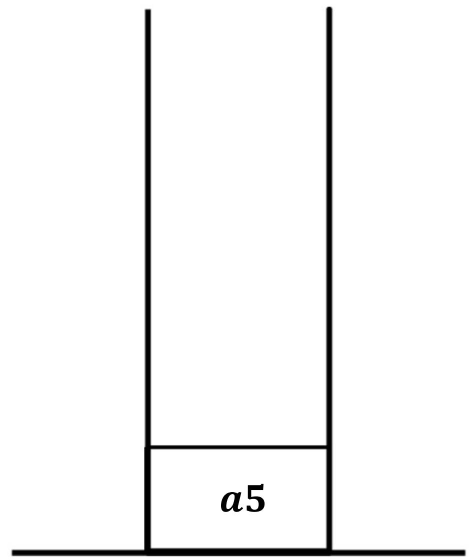
a9 has been popped



A stack-based implementation

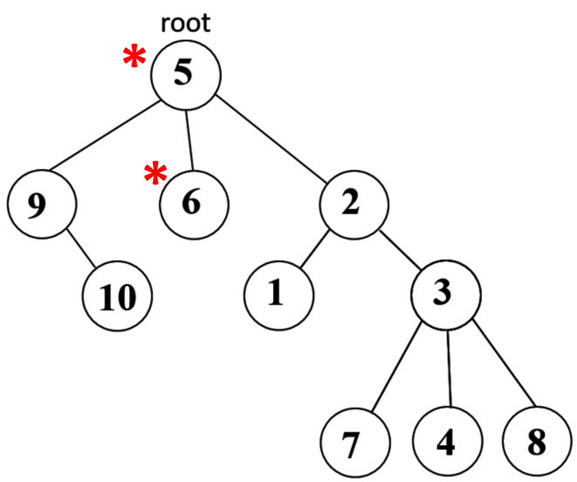


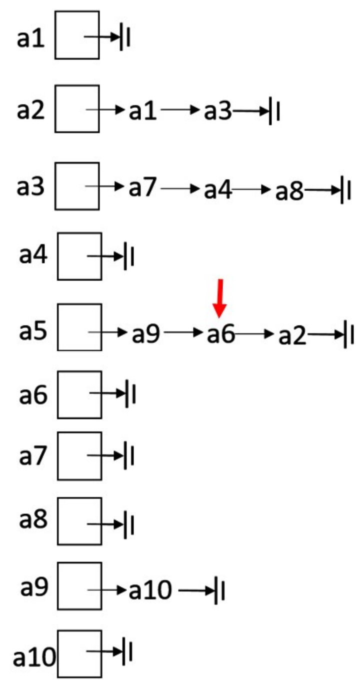




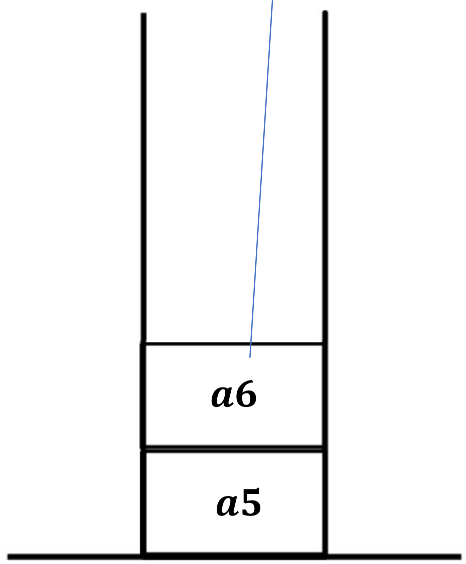
moving the pointer to the next child of node 5

A stack-based implementation



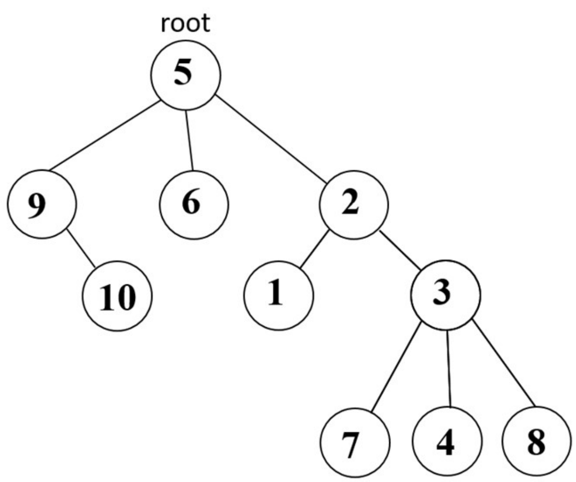


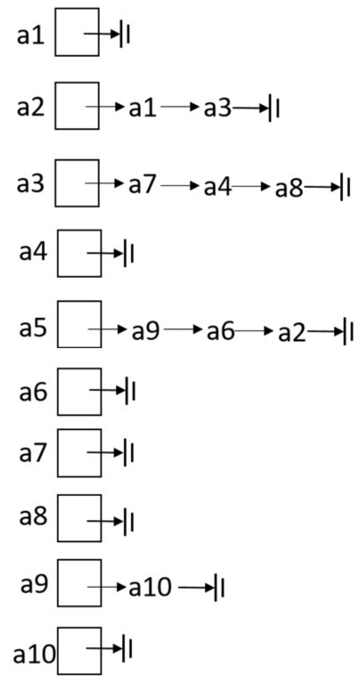
push the next child into the stack



The algorithm then continues in the same fashion.

A stack-based implementation





- Running time = O(*n*) because very node in the linked lists is pushed once and popped once.