Tree (10A)

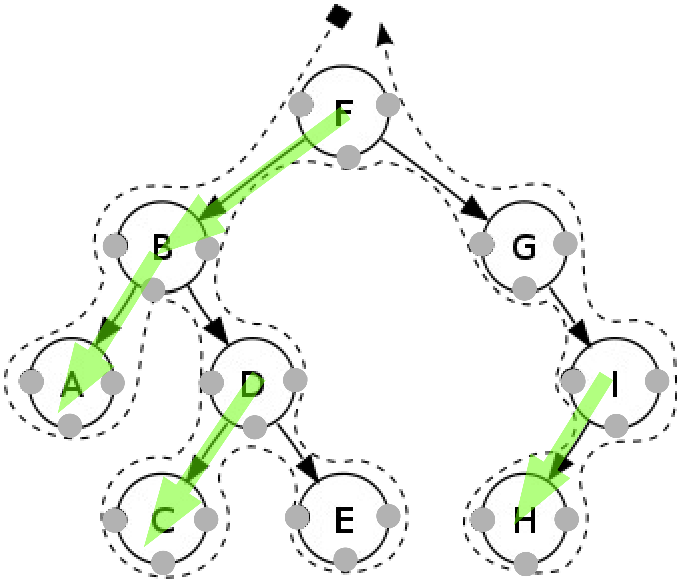
Young Won Lim
6/14/17

Please send corrections (or suggestions) to youngwlim@hotmail.com.

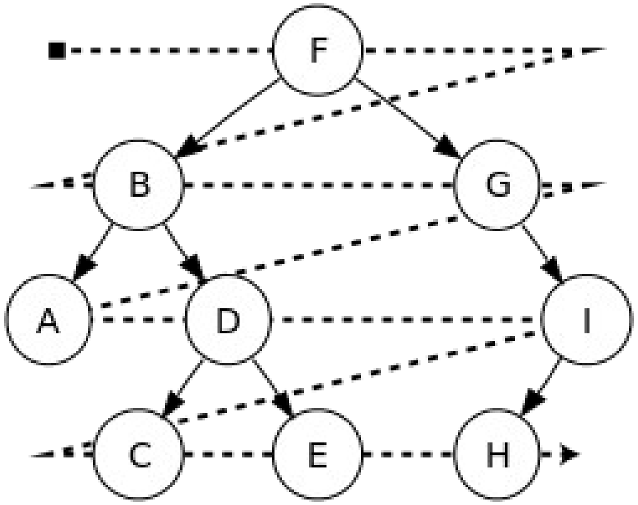
This document was produced by using OpenOffice and Octave.

Tree Traversal

Depth First Search



Breadth First Search

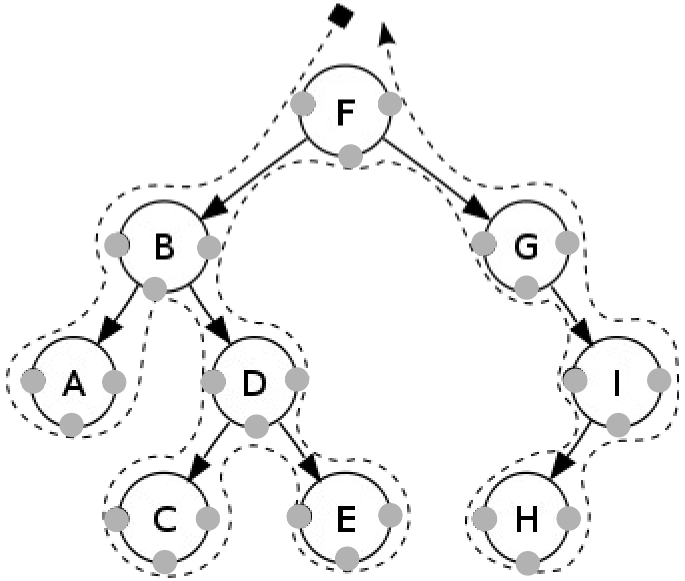


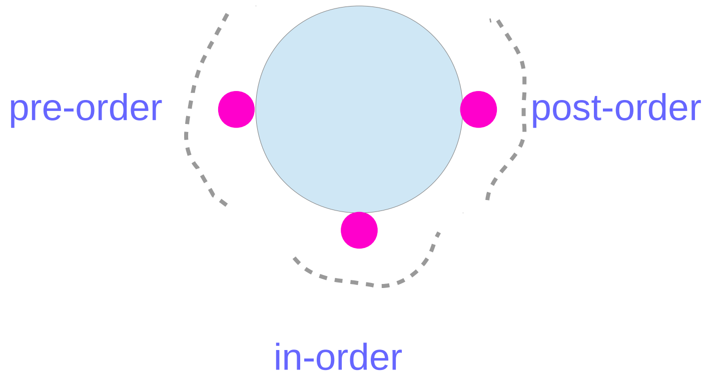
https://en.wikipedia.org/wiki/Morphism

Tree Traversal

Depth First Search
- Pre-Order
- In-order
- Post-Order

Breadth First Search





https://en.wikipedia.org/wiki/Morphism

Pre-Order

pre-order function

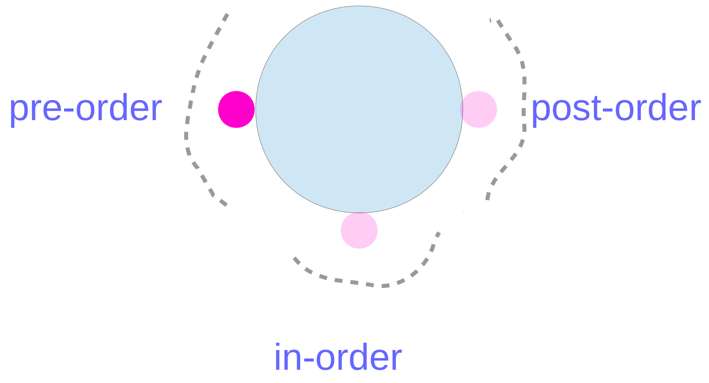
Check if the current node is empty / null.

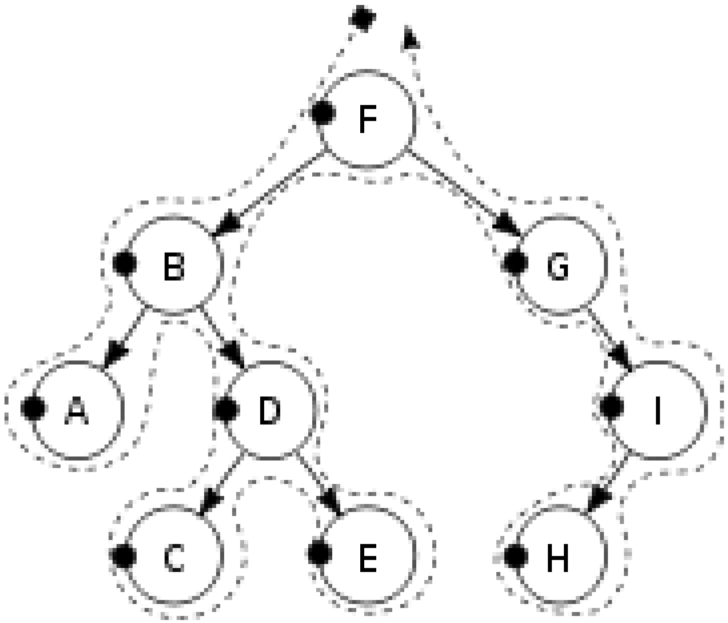
<u>Display</u> the data part of the root (or current node).

Traverse the **left** subtree by recursively calling the **pre-order** function.

Traverse the **right** subtree by recursively calling the **pre-order** function.

FBADCEGIH





https://en.wikipedia.org/wiki/Morphism

In-Order

in-order function

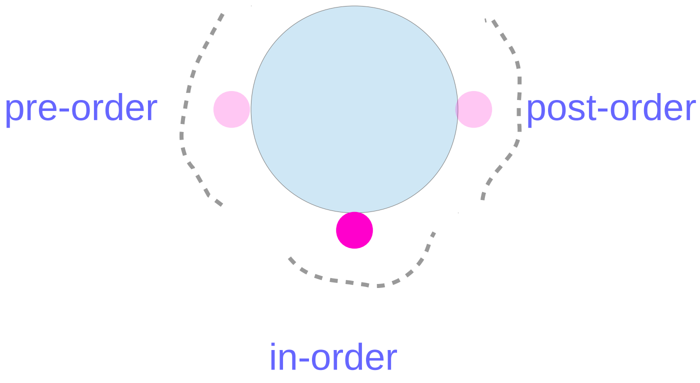
Check if the current node is empty / null.

Traverse the left subtree by recursively calling the **in-order** function.

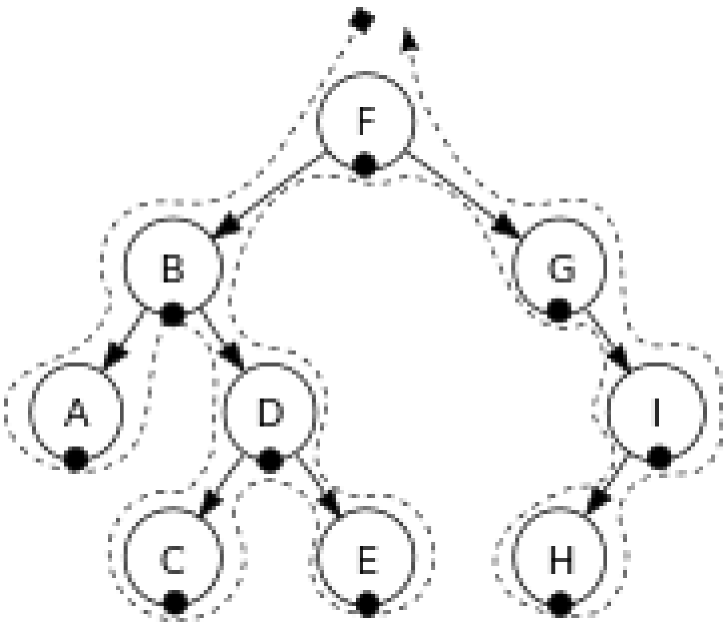
Display the data part of the root (or current node).

Traverse the right subtree by recursively calling the **in-order** function.

ABCDEFGHI



https://en.wikipedia.org/wiki/Morphism



Post-Order

post-order function

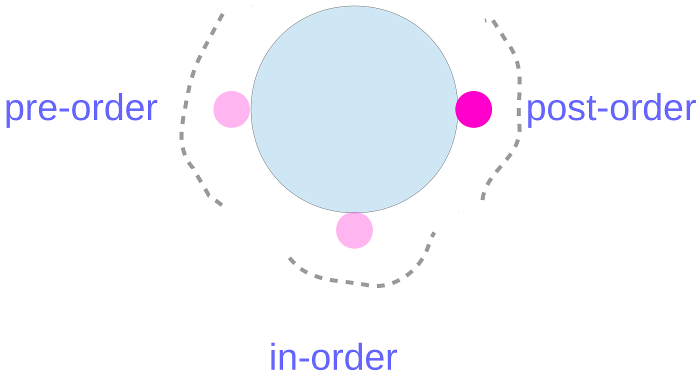
Check if the current node is empty / null.

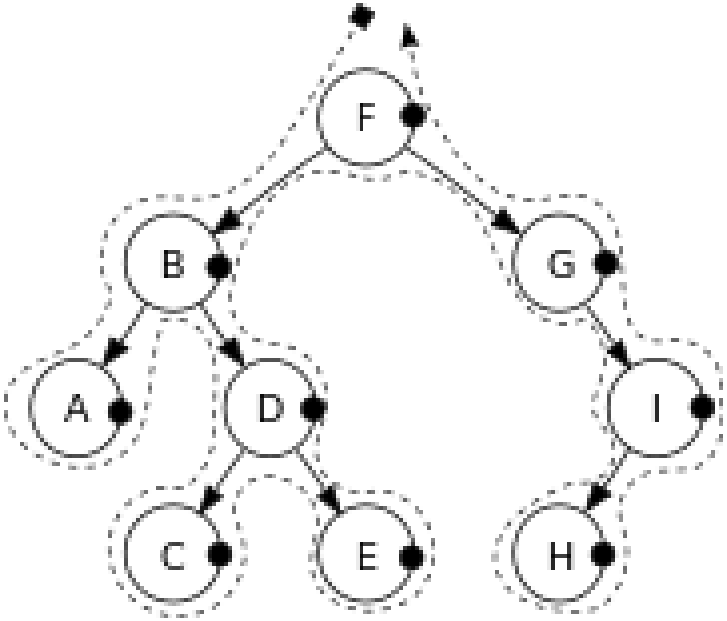
Traverse the left subtree by recursively calling the **post-order** function.

Traverse the right subtree by recursively calling the **post-order** function.

Display the data part of the root (or current node).

ACEDBHIGH





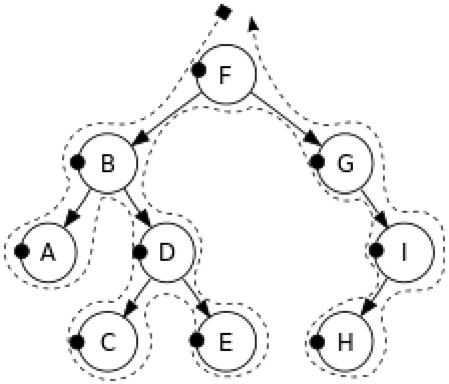
https://en.wikipedia.org/wiki/Morphism

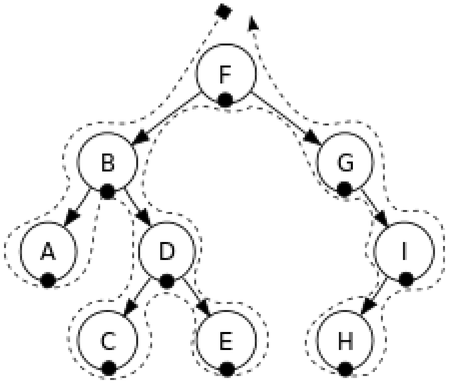
Recursive Algorithms

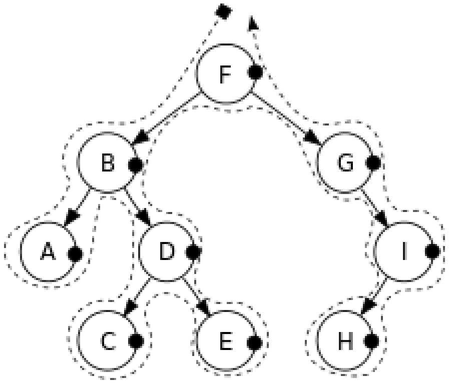
```
preorder(node)
  if (node = null)
    return
  visit(node)
  preorder(node.left)
  preorder(node.right)
```

```
inorder(node)
  if (node = null)
    return
  inorder(node.left)
  visit(node)
  inorder(node.right)
```

```
postorder(node)
  if (node = null)
    return
  postorder(node.left)
  postorder(node.right)
  visit(node)
```







https://en.wikipedia.org/wiki/Tree_traversal

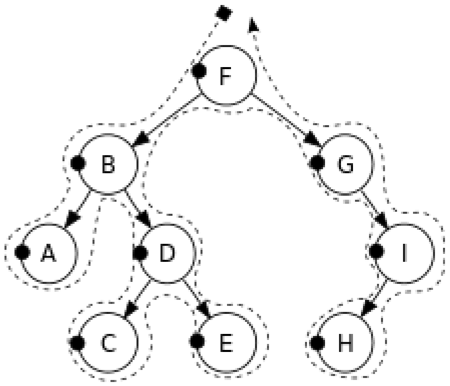
Iterative Algorithms

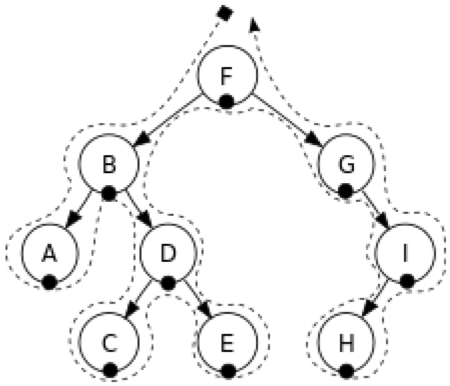
```
iterativePreorder(node)
  if (node = null)
    return
  s ← empty stack
  s.push(node)

  while (not s.isEmpty())
    node ← s.pop()
    visit(node)
    // right child is pushed first
    // so that left is processed first
    if (node.right ≠ null)
      s.push(node.right)
    if (node.left ≠ null)
      s.push(node.left)
```

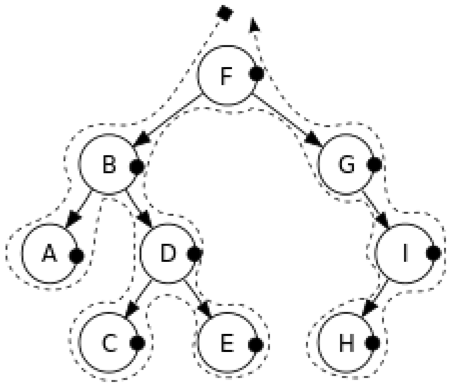
https://en.wikipedia.org/wiki/Tree_traversal

```
iterativeInorder(node)
  s ← empty stack

  while (not s.isEmpty() or
           node ≠ null)
    if (node ≠ null)
      s.push(node)
      node ← node.left
    else
      node ← s.pop()
      visit(node)
      node ← node.right
```

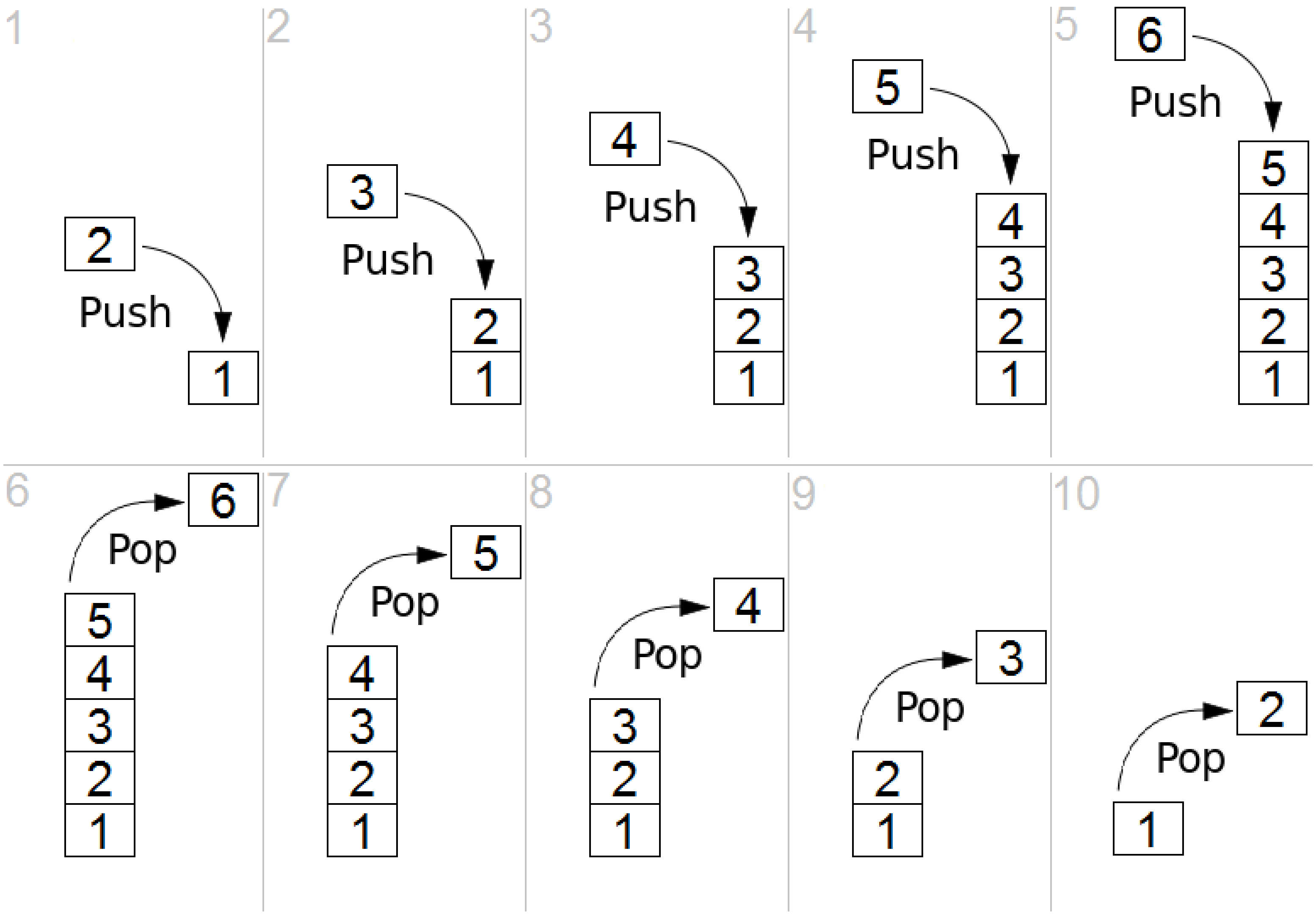
```
iterativePostorder(node)
  s ← empty stack
  lastNodeVisited ← null

  while (not s.isEmpty() or node ≠ null)
    if (node ≠ null)
      s.push(node)
      node ← node.left
    else
      peekNode ← s.peek()
      // if right child exists and traversing
      // node from left child, then move right
      if (peekNode.right ≠ null and
         lastNodeVisited ≠ peekNode.right)
        node ← peekNode.right
      else
        visit(peekNode)
        lastNodeVisited ← s.pop()
```





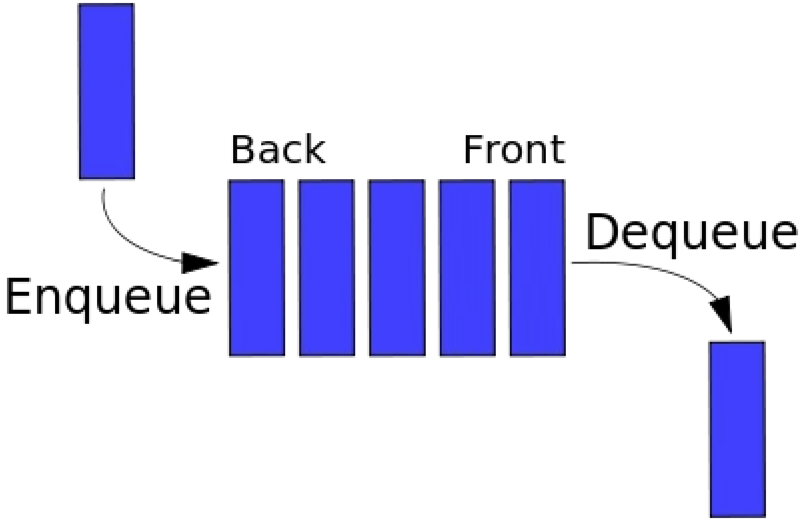


Stack



https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

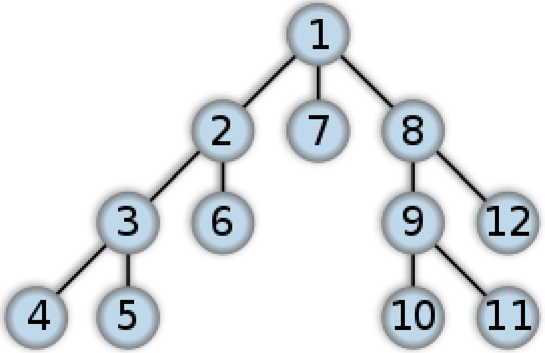
Queue



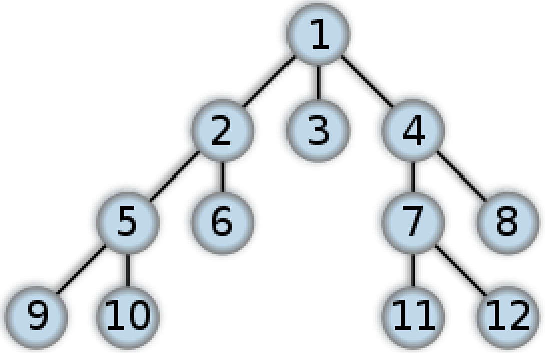
https://en.wikipedia.org/wiki/Queue_(abstract_data_type)#/media/File:Data_Queue.svg

Search Algorithms

DFS (Depth First Search)



BFS (Breadth First Search)



https://en.wikipedia.org/wiki/Breadth-first_search, /Depth-first_search

DFS Algorithm

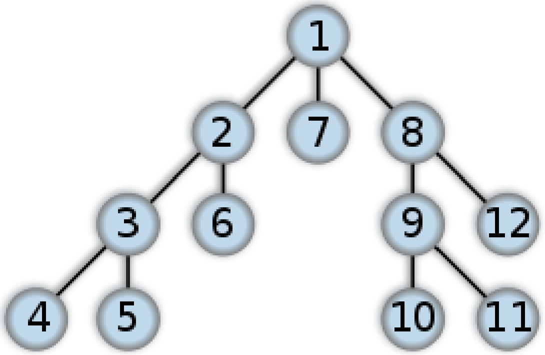
A recursive implementation of DFS:

```
procedure DFS(G,v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```

A non-recuUrsive implementation of DFS:

```
procedure DFS-iterative(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.pop()
        if v is not labeled as discovered:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

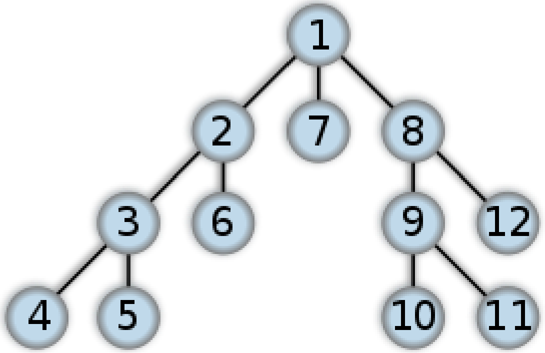
DFS (Depth First Search)



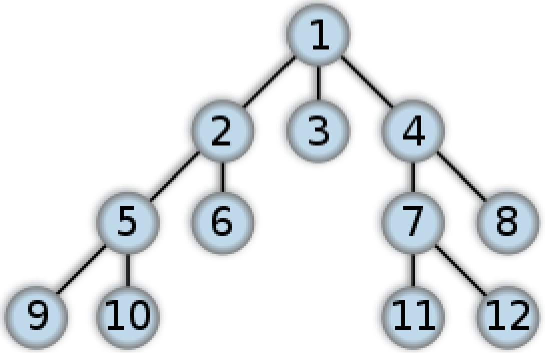
https://en.wikipedia.org/wiki/Breadth-first_search, /Depth-first_search

Search Algorithms

DFS (Depth First Search)



BFS (Breadth First Search)

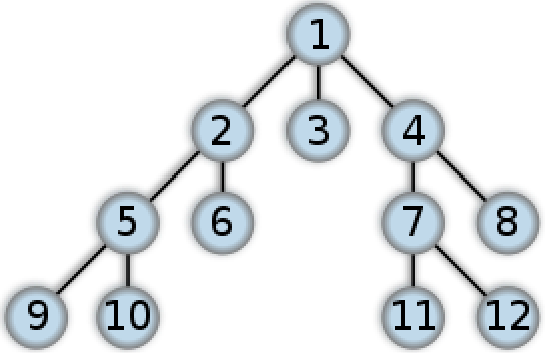


https://en.wikipedia.org/wiki/Breadth-first_search, /Depth-first_search

BFS Algorithm

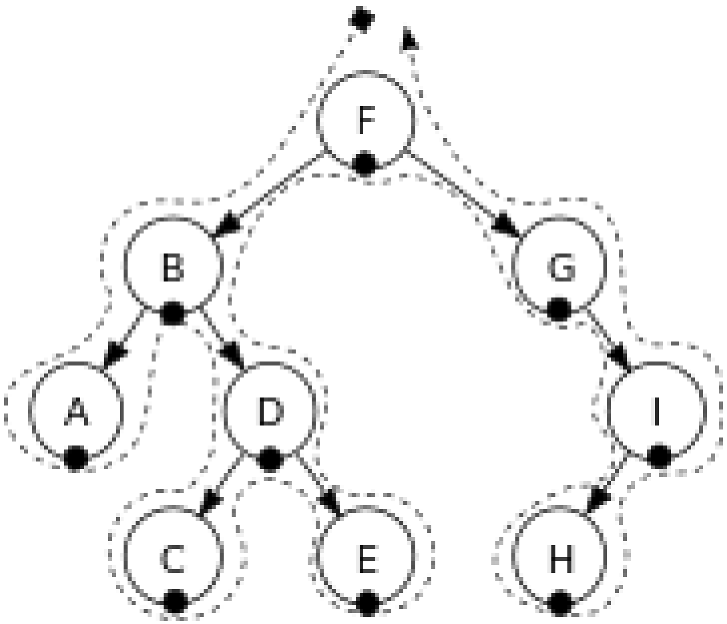
```
Breadth-First-Search(Graph, root):

    create empty set S
    create empty queue Q

    add root to S
    Q.enqueue(root)

    while Q is not empty:
        current = Q.dequeue()
        if current is the goal:
            return current
        for each node n that is adjacent to current:
            if n is not in S:
                add n to S
                n.parent = current
                Q.enqueue(n)
```

BFS (Breadth First Search)



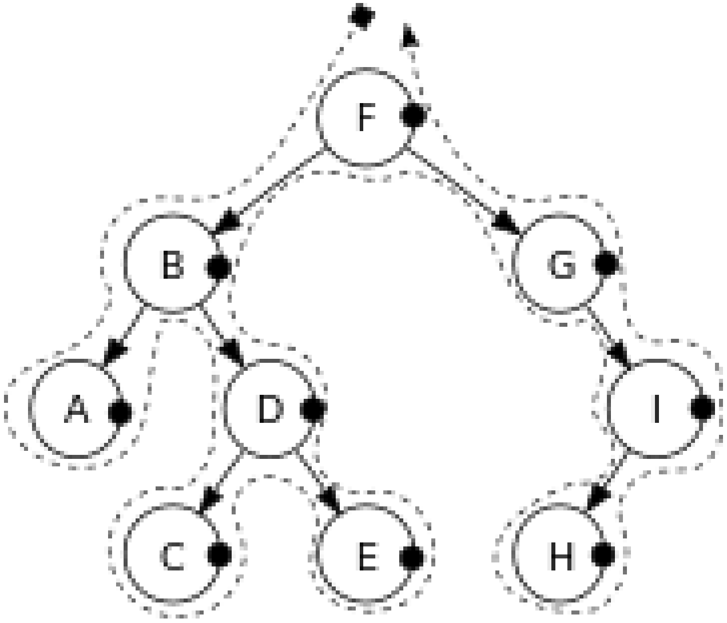
https://en.wikipedia.org/wiki/Breadth-first_search, /Depth-first_search

In-Order



https://en.wikipedia.org/wiki/Morphism

Post-Order



https://en.wikipedia.org/wiki/Morphism

References

[1] http://en.wikipedia.org/
[2]