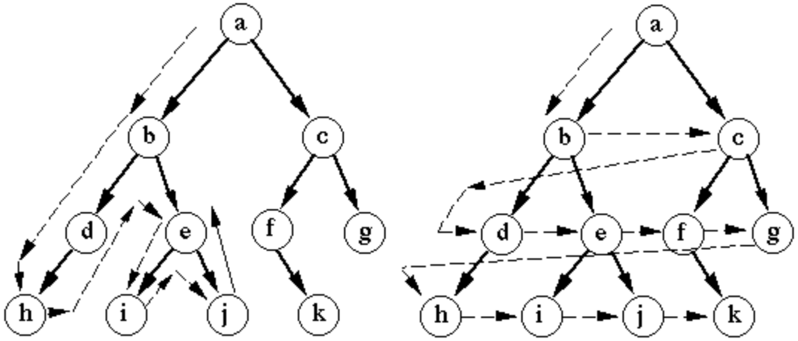
Traversal methods

> *Methodes to explore and process all the nodes in a binary tree*

- Because Trees are non-linear, there are multiple possible paths

Two main traversal strategies:



1. **Depth-First Traversal (DFS):**
    - visiting a node (sarting with the root)
    - then recursively traversing as deep as possible
    - then explore another branch.

2. **Breadth-First Traversal (BFS):**
    - visiting a node (sarting with the root)
    - explore all its neighbors (children)
    - then mode move to the children.

Depth-first

Pseudo-code for Depth-First Traversal:

1. Place the source node in the **stack**.
2. Remove the node from the top of the stack for processing.
3. Add all unexplored neighbors to the stack (at the top).
4. If the stack is not empty, go back to step 2.

In [60]:

```
def dfs(graph, start):
  stack = [start]
  while stack:
      vertex = stack.pop()
      print(vertex) # traitement
      stack.extend(graph[vertex])

graph = {'A': set(['B', 'C']),
         'B': set(['D', 'E', 'F']),
         'C': set([]),
         'D': set([]),
         'E': set([]),
         'F': set([])
        }

dfs(graph, 'A') # A B D F E C
```

A

B

D

E

F

C

Depth-first traversal: pre-order, in-order, and post-order.

For **depth-first traversal**, there are different types of processing: *pre-order*, *in-order*, and *post-order*.

- R = Root
- D = Right subtree
- G = Left subtree

There are three (main) types of traversal, observing the position of R:

- **Pre-order**: R G D
- **In-order**: G R D
- **Post-order**: G D R

Depth-first traversal: pre-order, in-order, and post-order.

For **depth-first traversal**, there are different types of processing: *pre-order*, *post-order*, and *in-order*.

```
def Preorder(R):
  if not empty(R):
    process(R)    # Root
    Preorder(left(R))   # Left
    Preorder(right(R))  # Right

def Inorder(R):
  if not empty(R):
    Inorder(left(R))   # Left
    process(R)         # Root
    Inorder(right(R))  # Right

def Postorder(R):
  if not empty(R):
    Postorder(left(R))  # Left
    Postorder(right(R)) # Right
    process(R)          # Rooot
```

Depth-first traversal: pre-order

Iterative implementation.

In [62]:

```
def iterative_inorder_traversal(root):
    stack = []
    current = root
    while current is not None or stack:
        while current is not None:
            stack.append(current)
            current = current.left
        current = stack.pop()
        print(current.value)
        current = current.right
```

Depth-first traversal: pre-order

Recursive implementation.

In [114]:

```
TT = {"dog": ["little", "very"],
     "little": ["the"],
     "the": [],
       "very": ["is", "cute"],
       "is": [],
       "cute": []
     }
```

In [117]:

```
def preorder(T, node):
    if node is not None:
        if len(T[node]) > 0:
            preorder(T, T[node][0])
        print(node)
        if len(T[node]) > 1:
            preorder(T, T[node][1])
```

In [118]:

```
preorder(TT, "dog")
```

```
the
little
dog
is
```

```
very
cute
```

In [68]:

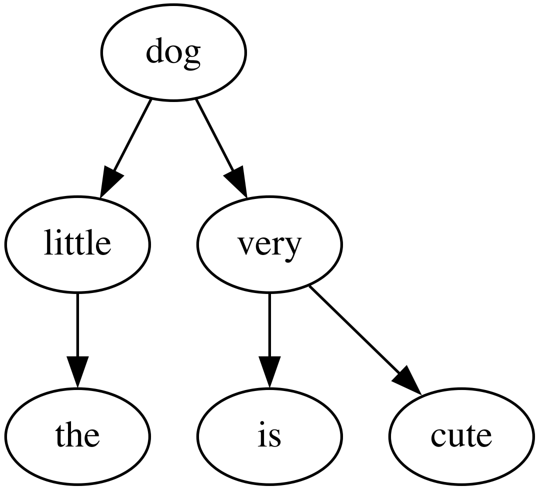
```python
def preorder_traversal(node):
    if node is not None:
        if node.left:
            preorder_traversal(node.left)
        print(node.value)
        if node.right:
            preorder_traversal(node.right)

root = Node("dog")
root.left = Node("little")
root.left.left = Node("the")
root.right = Node("very")
root.right.left = Node("is")
root.right.right = Node("cute")
preorder_traversal(root)
```

```
the
little
dog
is
very
cute
```

In [69]:

```python
visualize_oop(root)
```

```
Out[69]:
```



Breadth-first traversal

Visit all the nodes in a tree or graph level by level.

```
    1
   / \
  2   3
 / \
4   5
```

In [81]:

```python
def bfs_print(node):
    if node is None:
        return

    queue = [node]

    while queue:
        current_node = queue.pop(0)
        print(current_node.value, end=' ')

        if current_node.left:
            queue.append(current_node.left)

        if current_node.right:
            queue.append(current_node.right)
```

In [82]:

```python
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
bfs_print(root)
```

```
1 2 3 4 5
```

Utils

In [83]:

```
import graphviz
from graphviz import Digraph
from IPython.display import display
```

In [84]:

```
def visualize_oop(root):
    def build(node, dot=None):
        if dot is None:
            dot = graphviz.Digraph(format='png')

        if node is not None:
            dot.node(str(node.value))

            if node.left is not None:
                dot.edge(str(node.value), str(node.left.value))
                build(node.left, dot)

            if node.right is not None:
                dot.edge(str(node.value), str(node.right.value))
                build(node.right, dot)

        return dot

    return build(root)
```

In [ ]: