Beginner tree algorithms

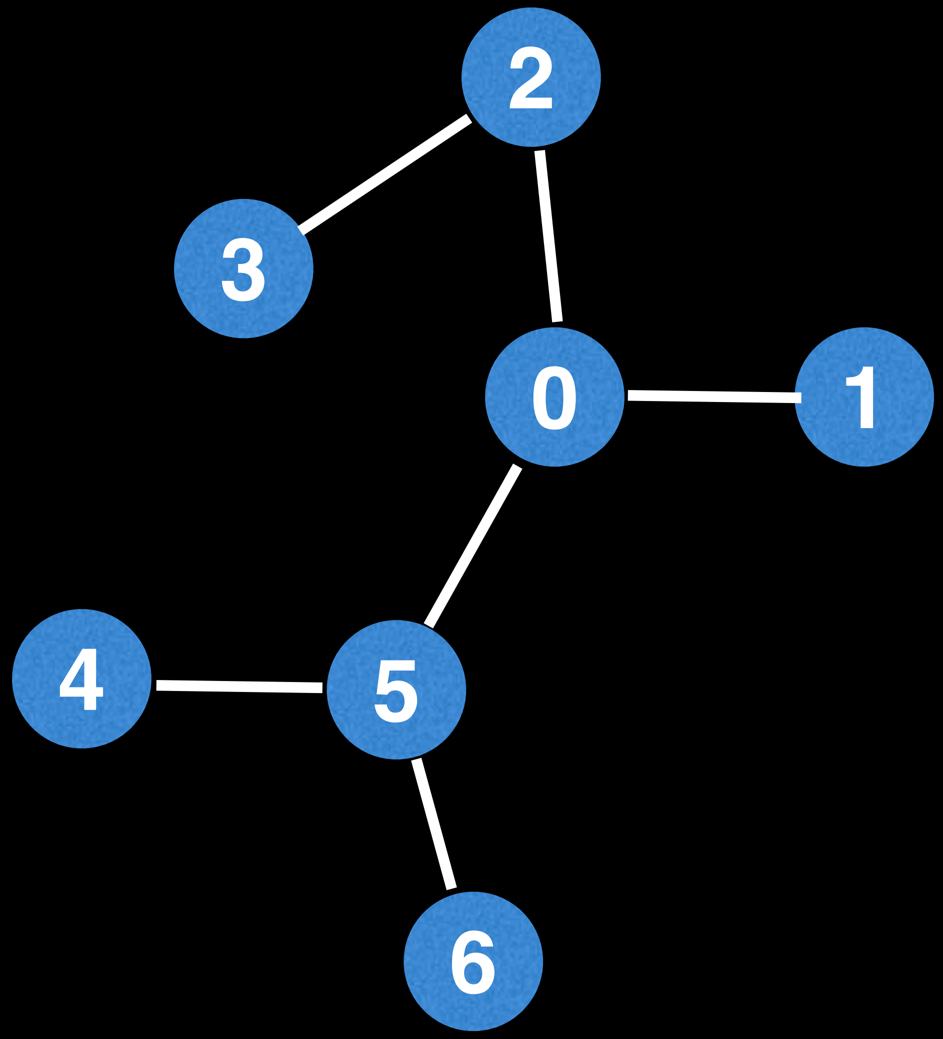
William Fiset

Rooting a tree

William Fiset

Rooting a tree

Sometimes it's useful to root an undirected tree to add structure to the problem you're trying to solve.

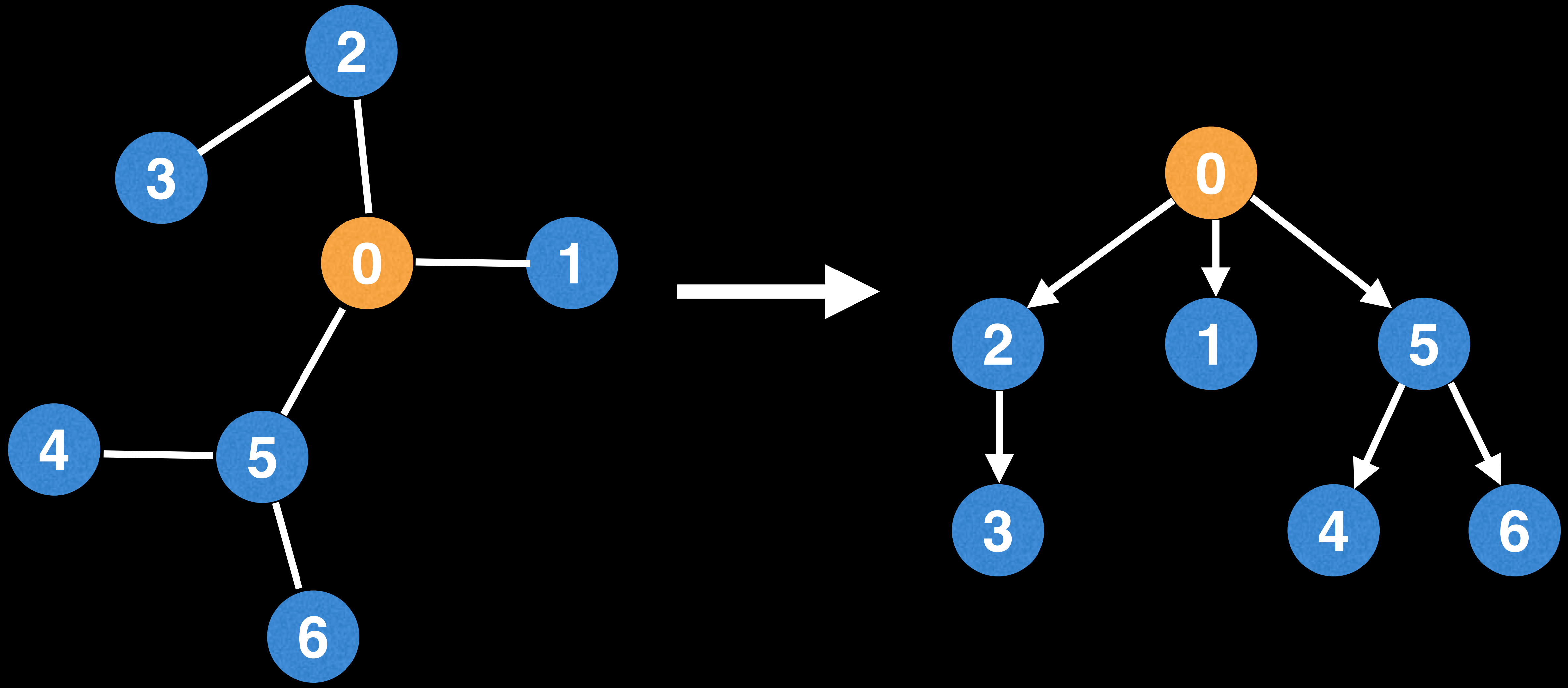


Undirected graph adjacency list:

```
0 -> [2, 1, 5]
1 -> [0]
2 -> [3, 0]
3 -> [2]
4 -> [5]
5 -> [4, 6, 0]
6 -> [5]
```

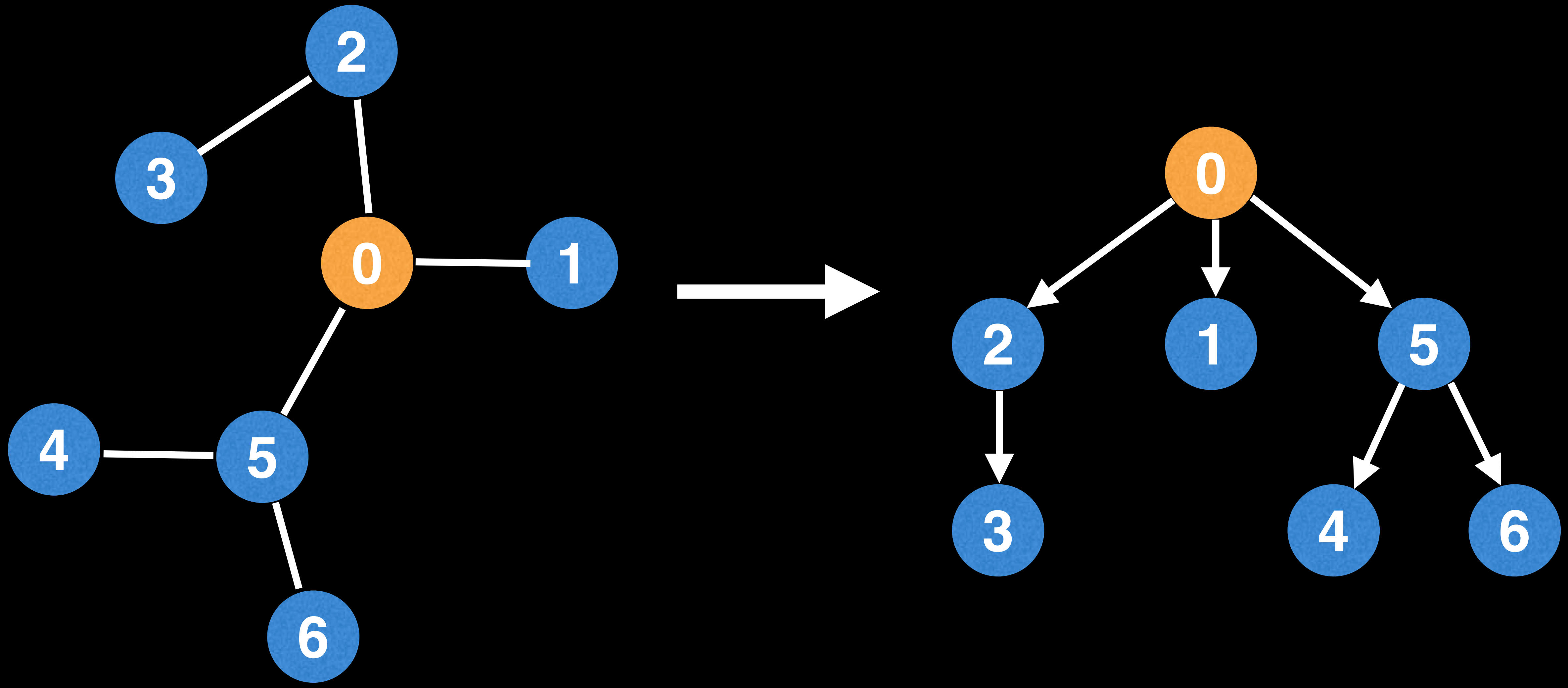
Rooting a tree

Sometimes it’s useful to root an undirected tree to add structure to the problem you’re trying to solve.



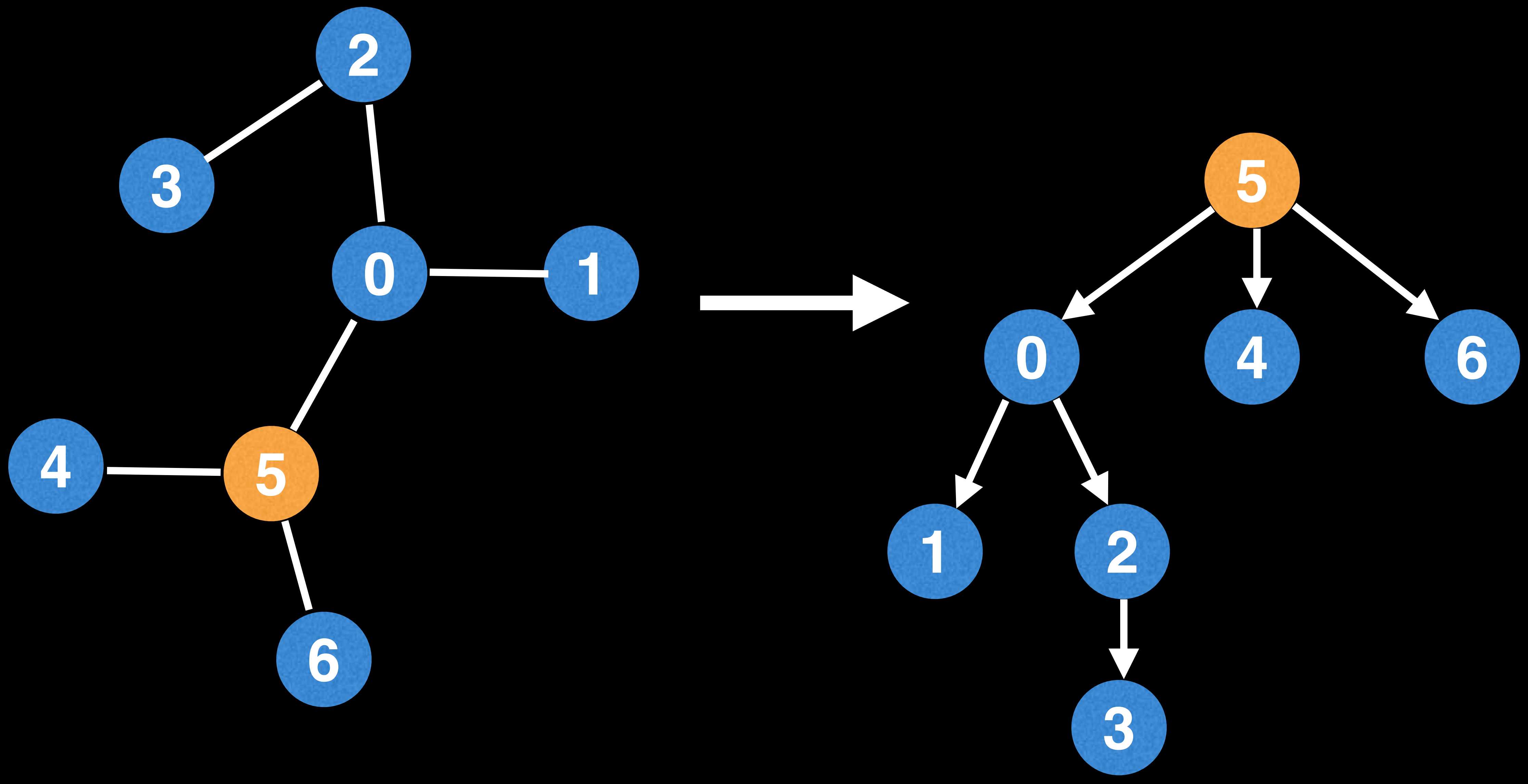
Rooting a tree

Conceptually this is like "picking up" the tree by a specific node and having all the edges point downwards.

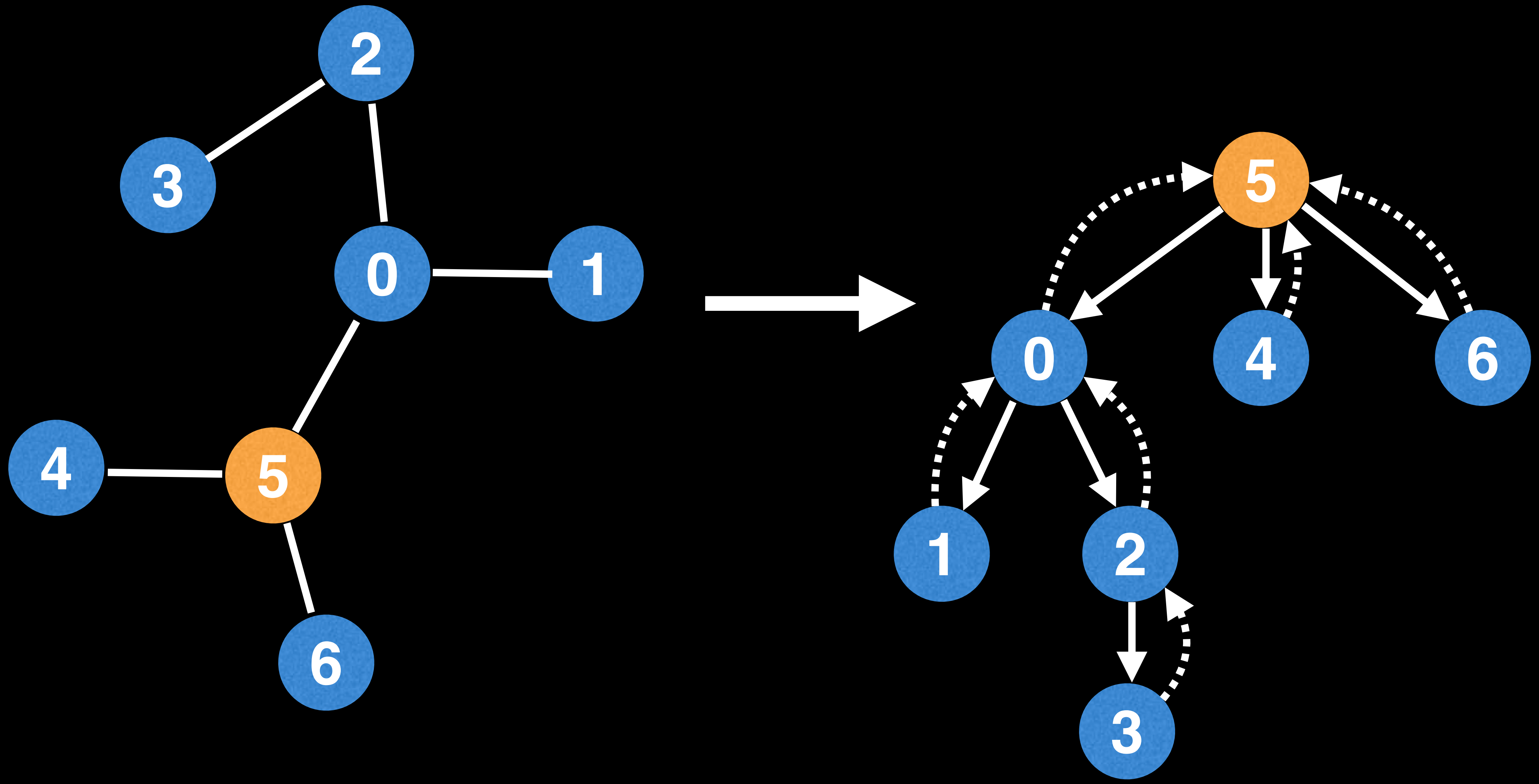


Rooting a tree

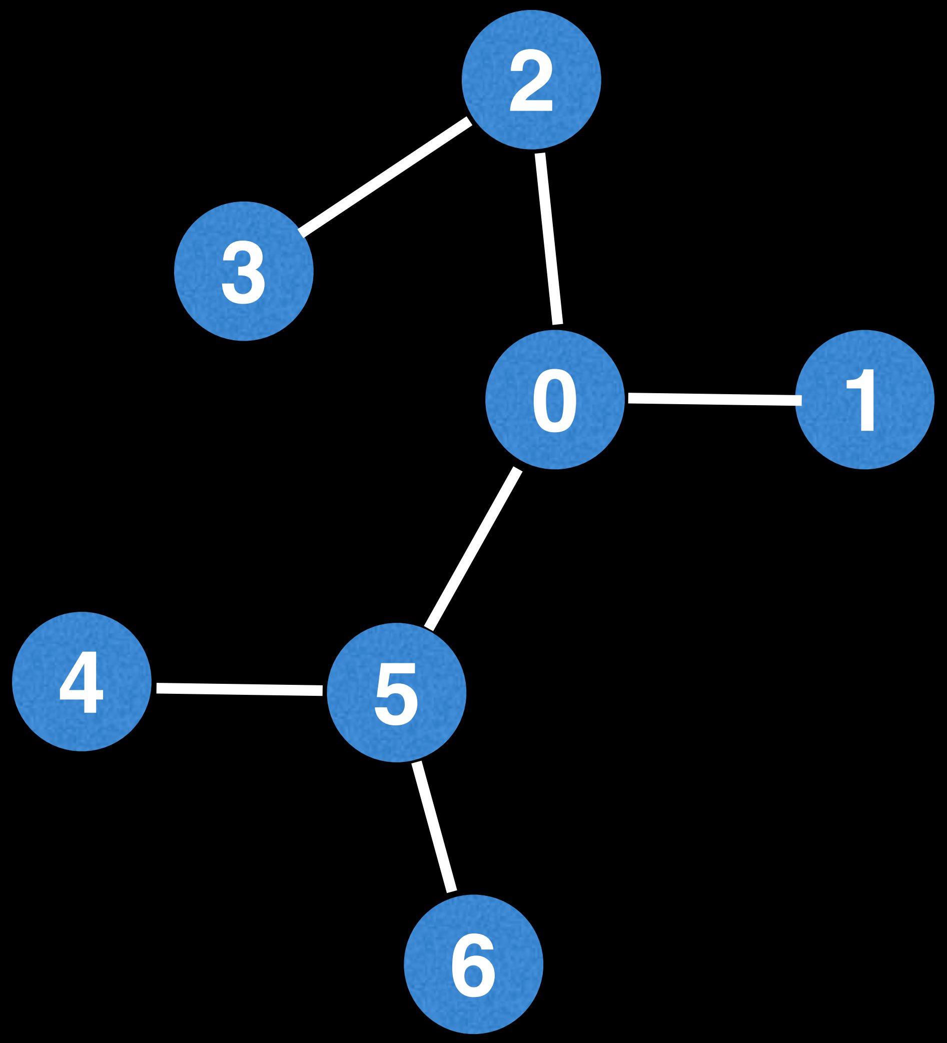
You can root a tree using any of its nodes.



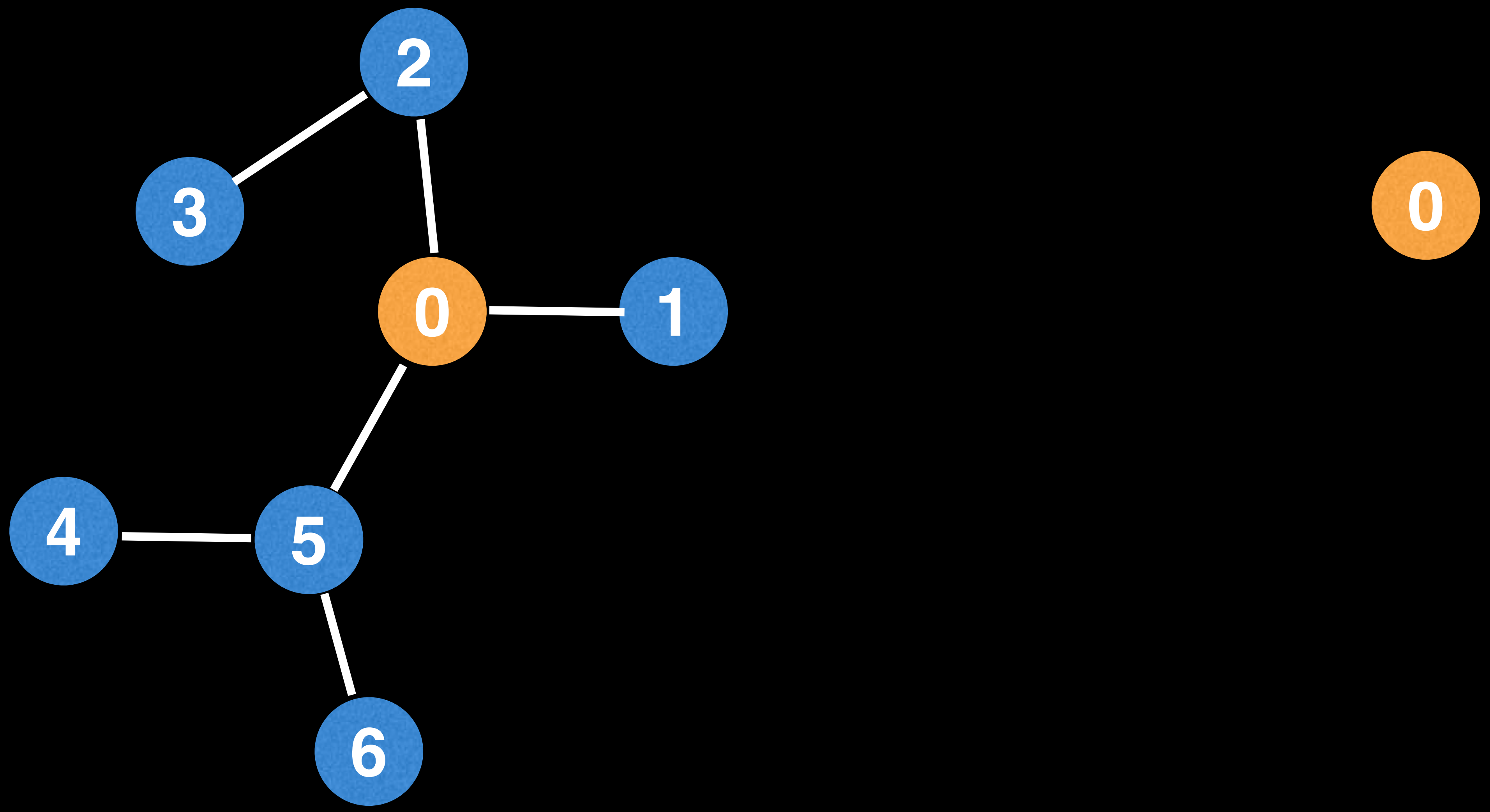
In some situations it's also useful to keep have a reference to the parent node in order to walk up the tree.



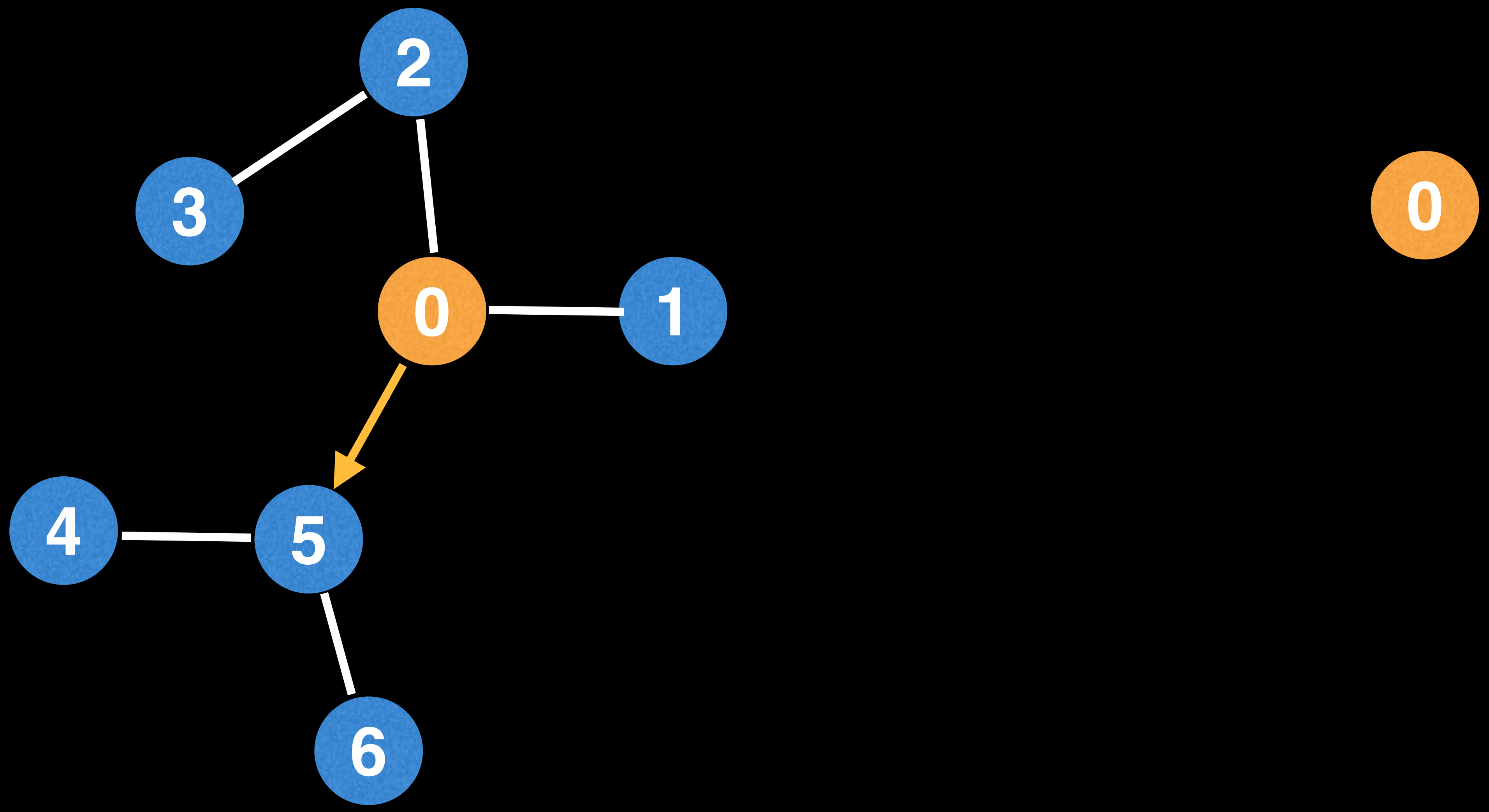
Rooting a tree is easily done depth first.



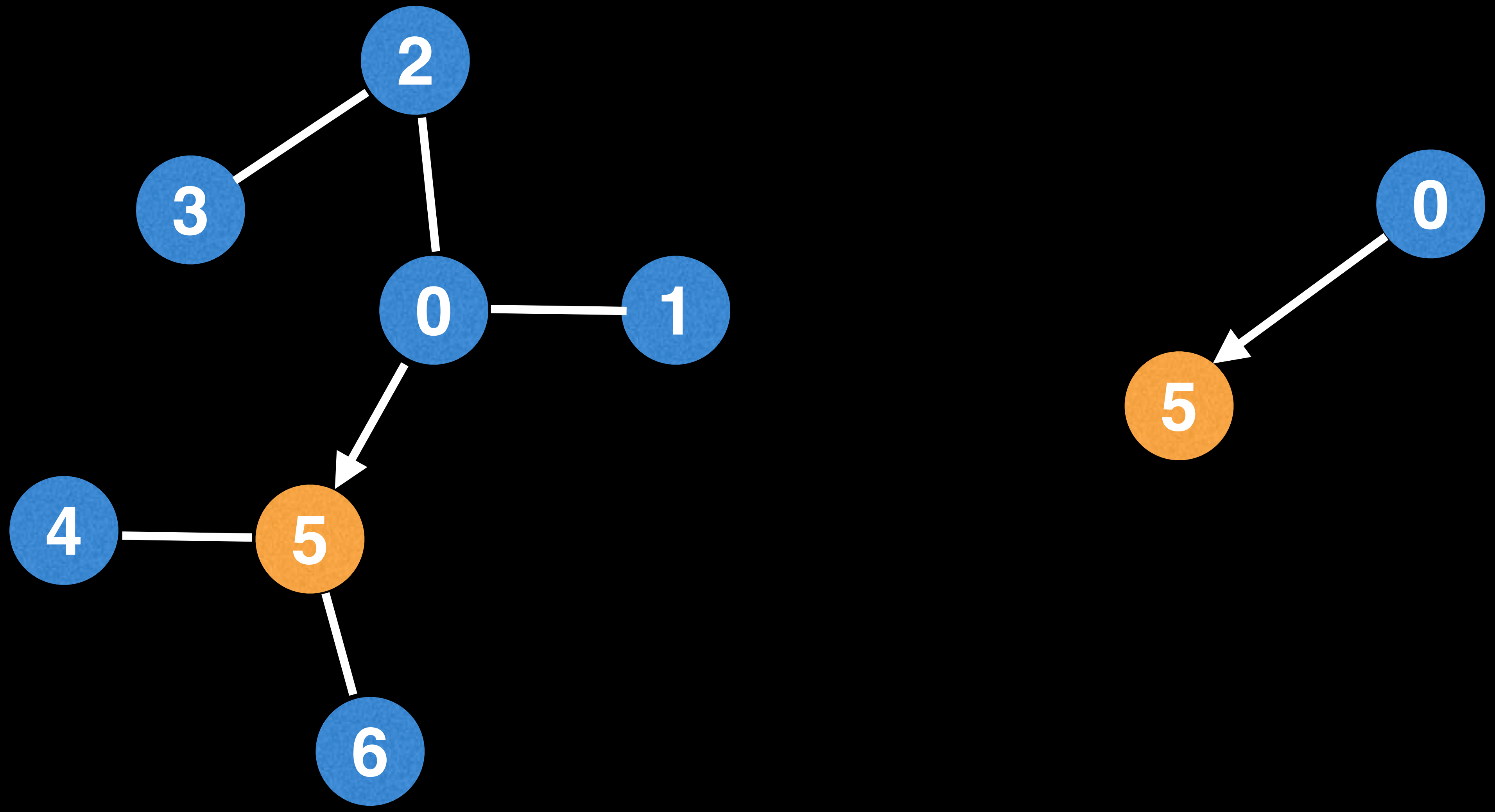
Rooting a tree is easily done depth first.



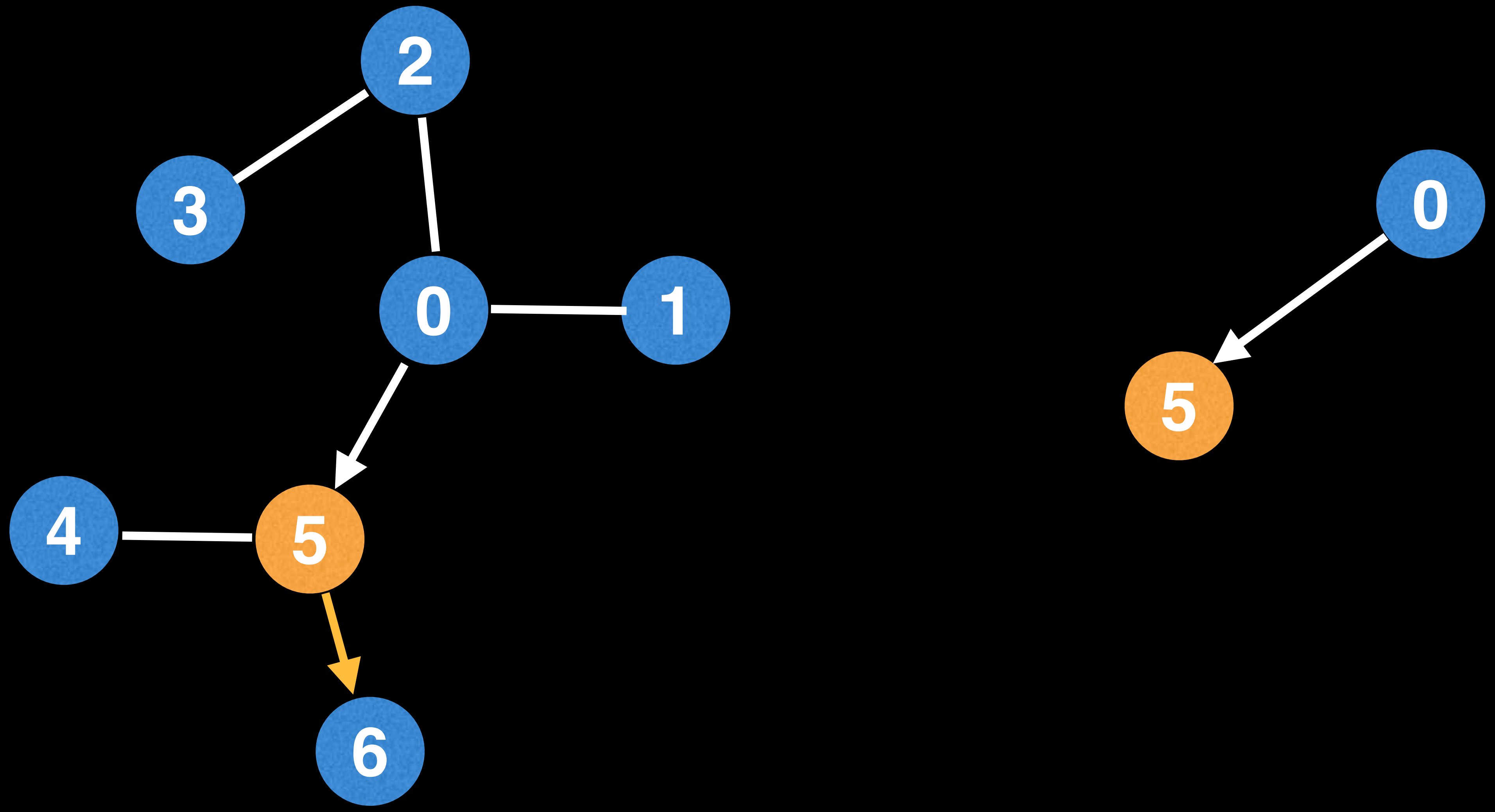
Rooting a tree is easily done depth first.



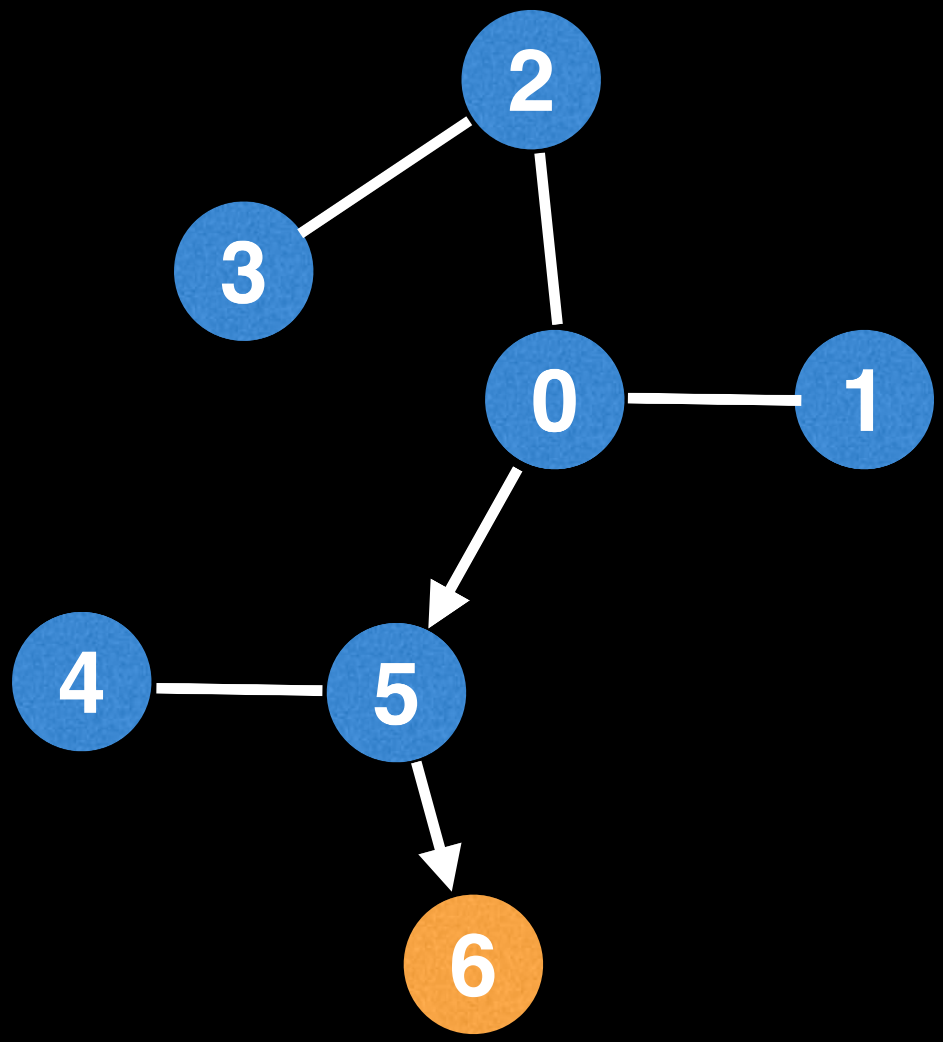
Rooting a tree is easily done depth first.

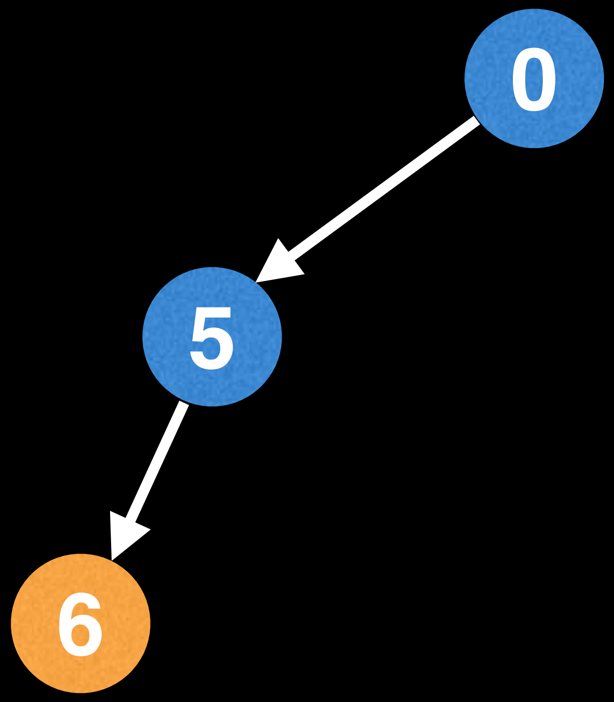


Rooting a tree is easily done depth first.

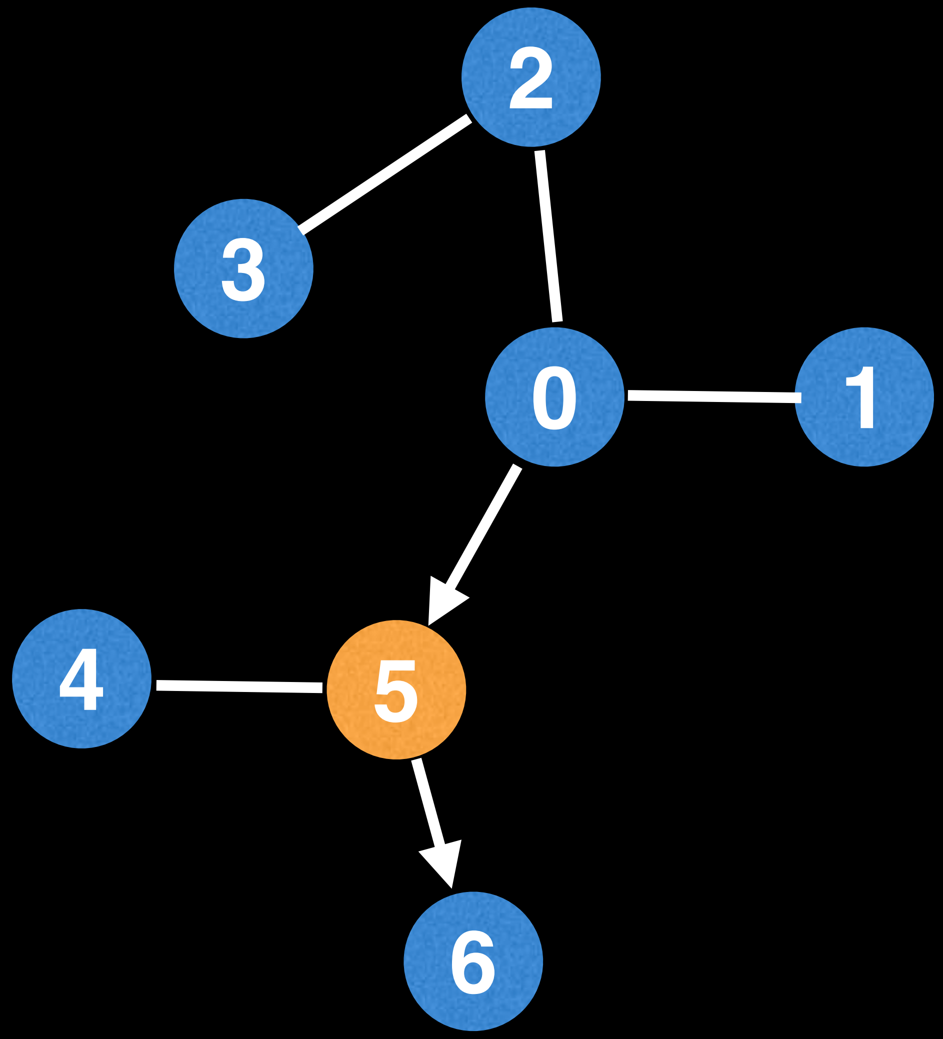


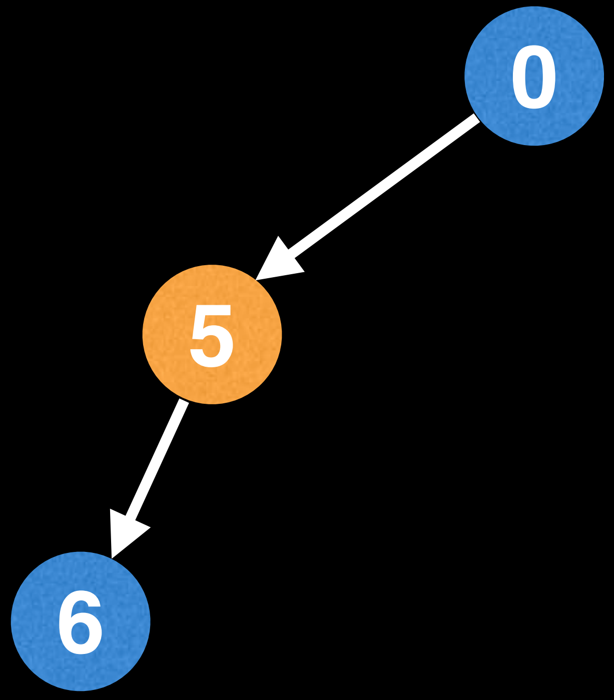
Rooting a tree is easily done depth first.



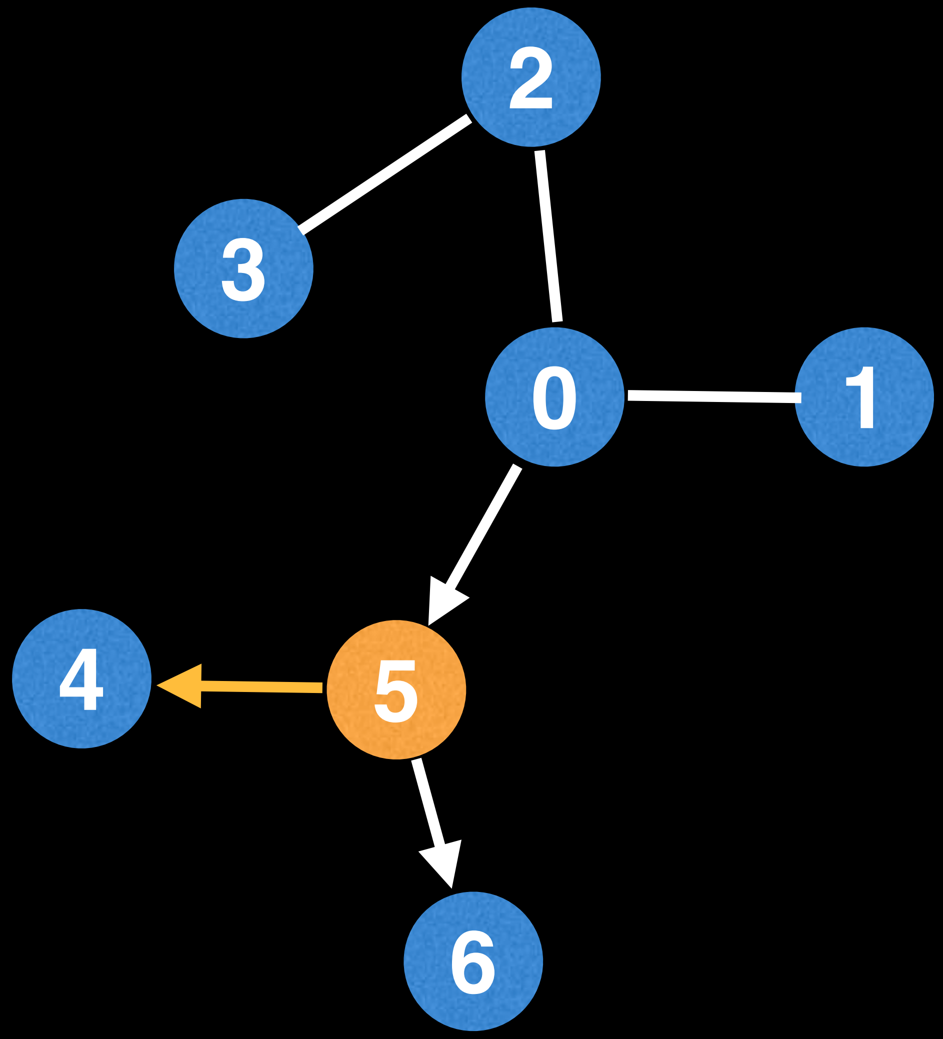


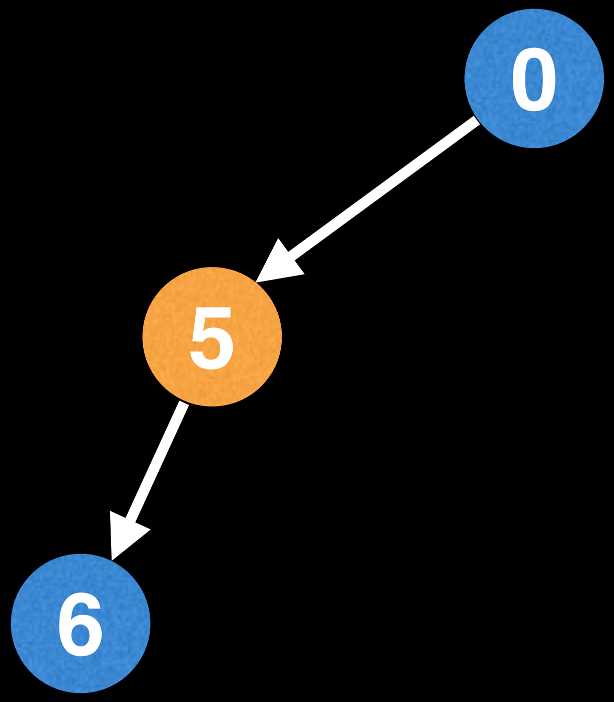
Rooting a tree is easily done depth first.



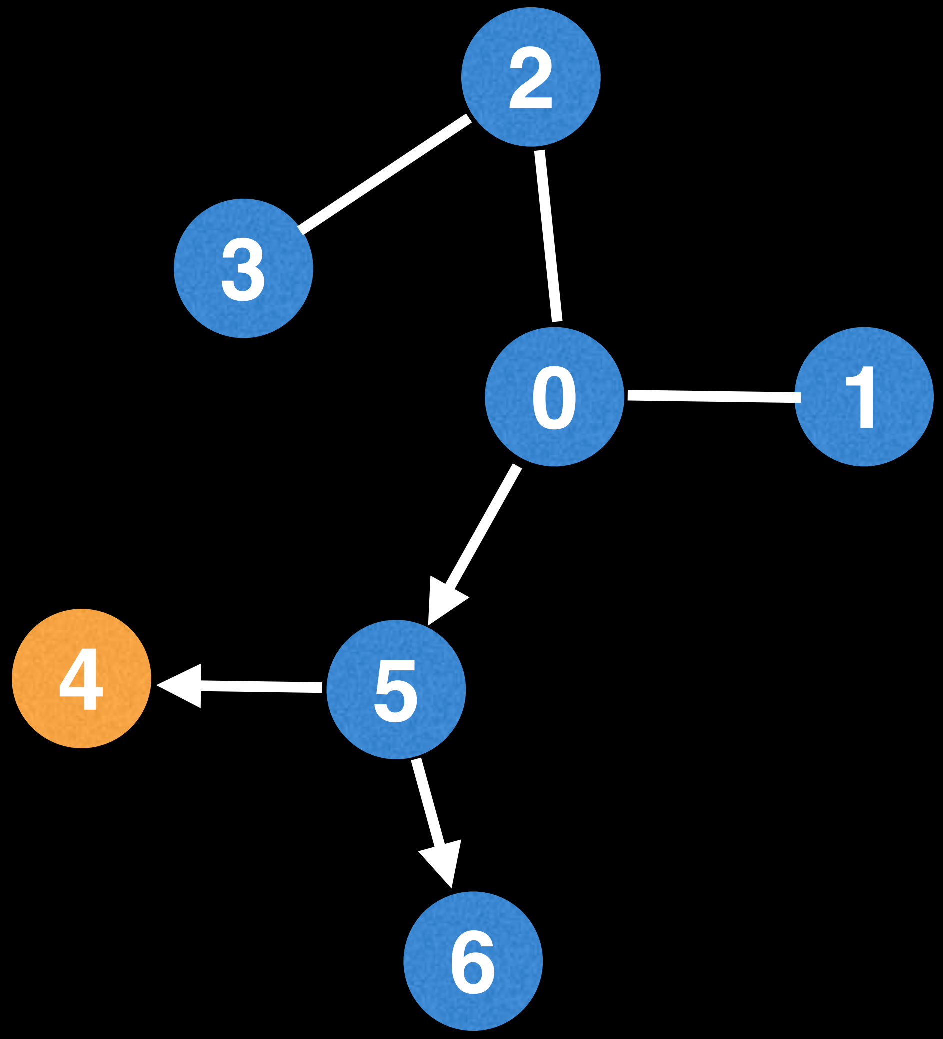


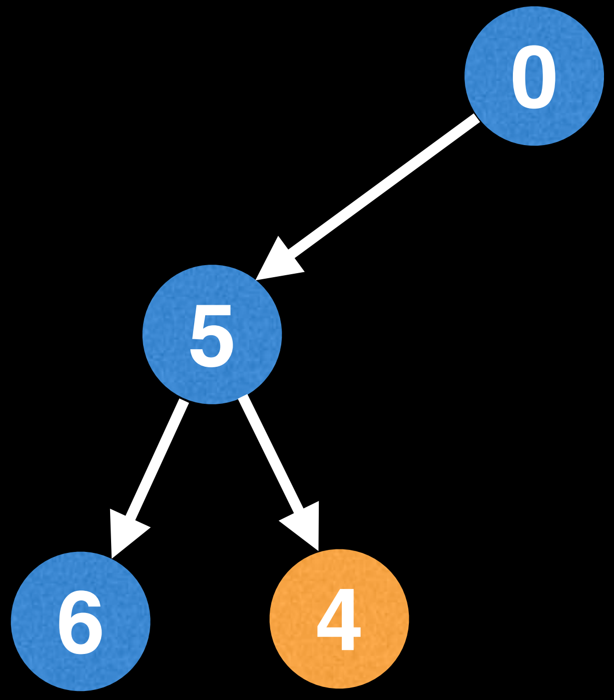
Rooting a tree is easily done depth first.



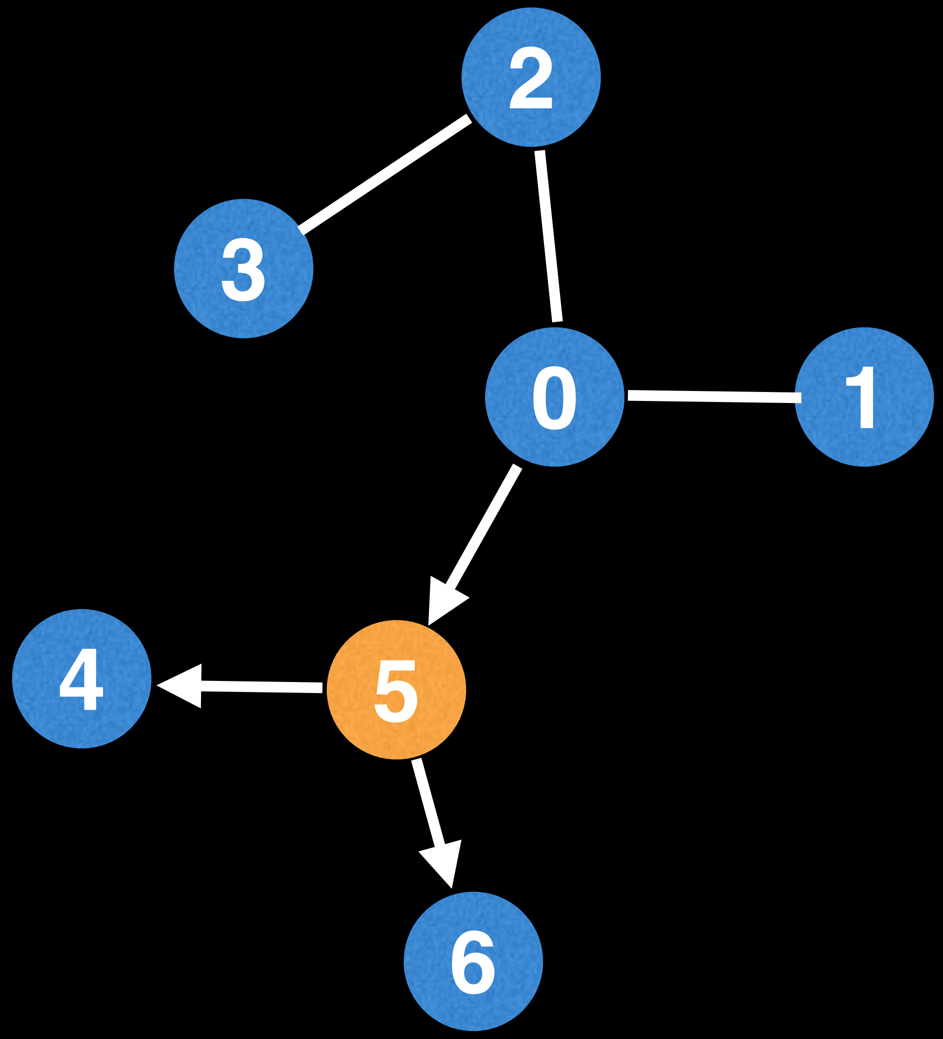


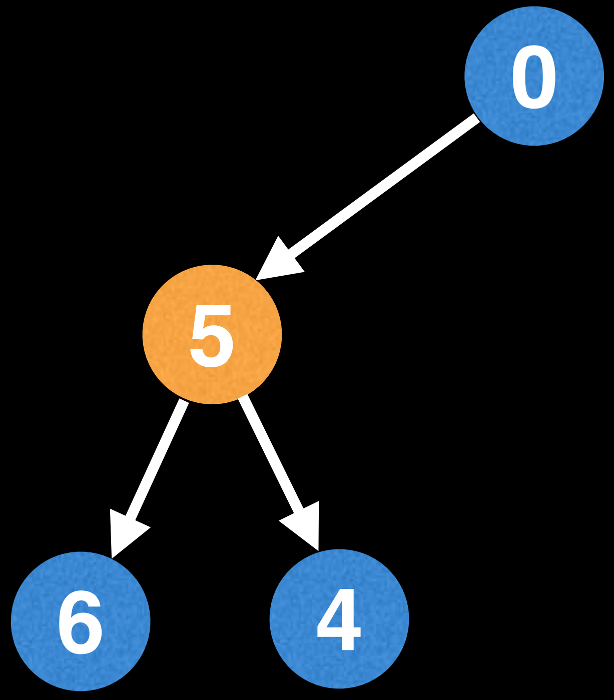
Rooting a tree is easily done depth first.



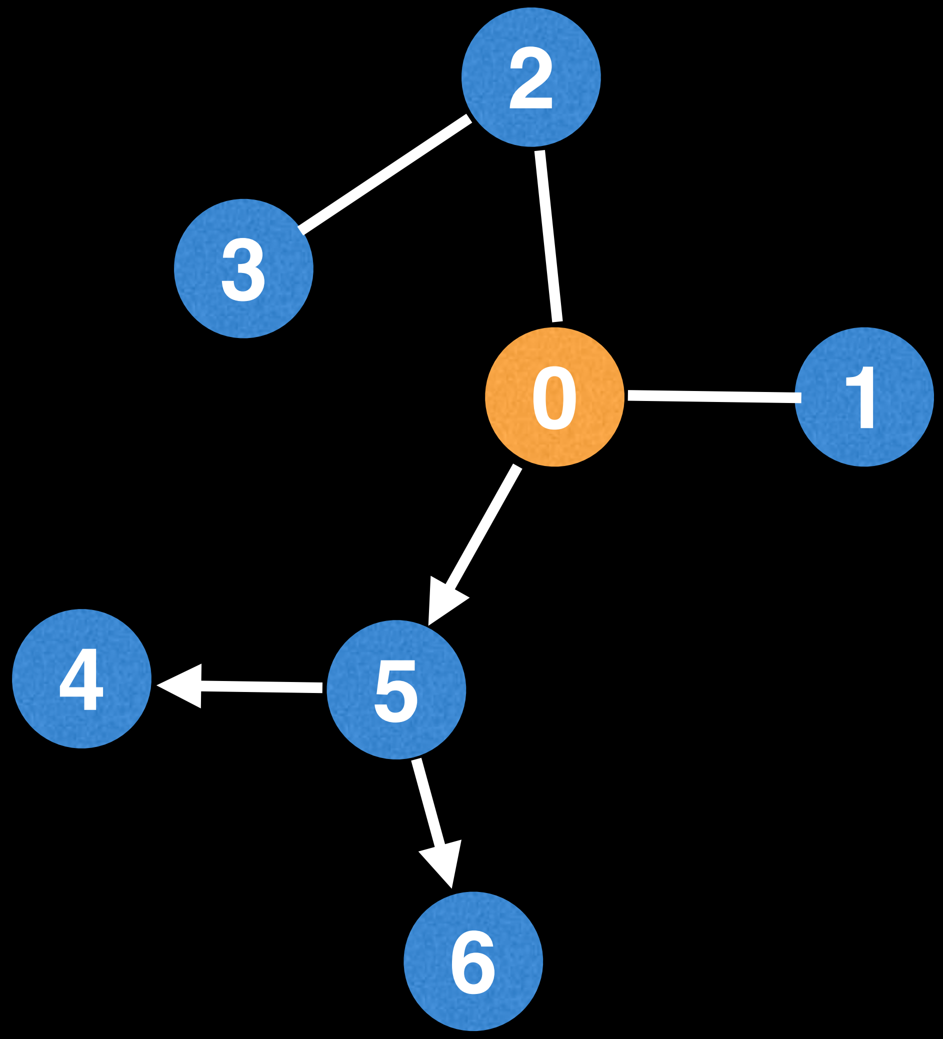


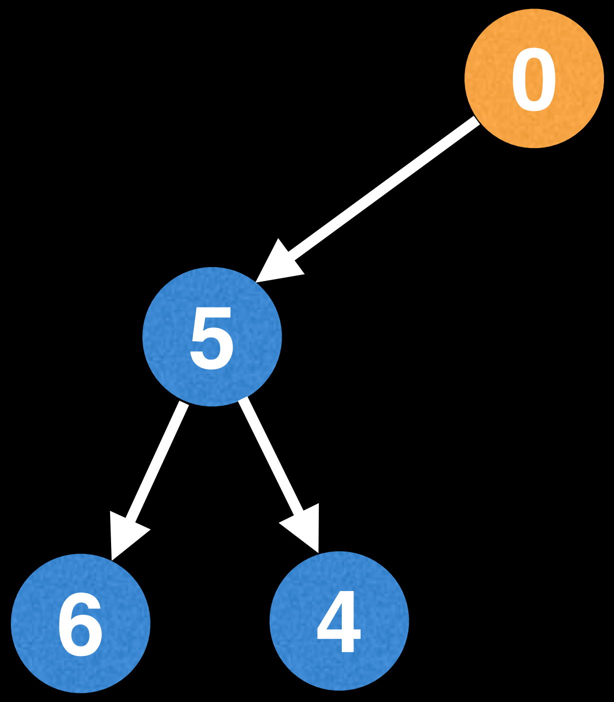
Rooting a tree is easily done depth first.



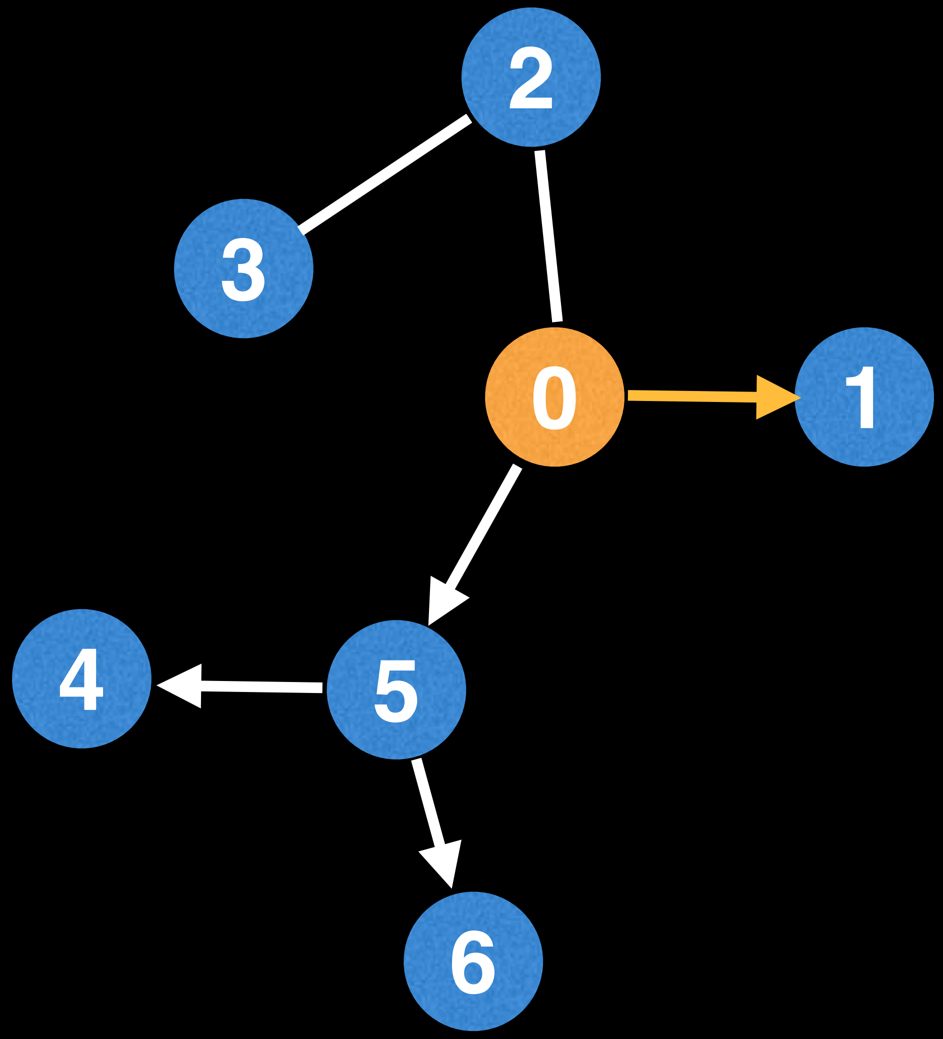


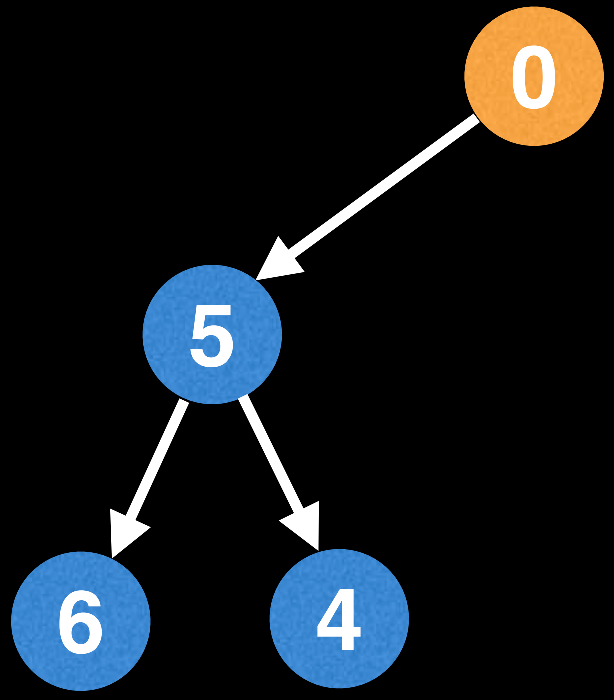
Rooting a tree is easily done depth first.



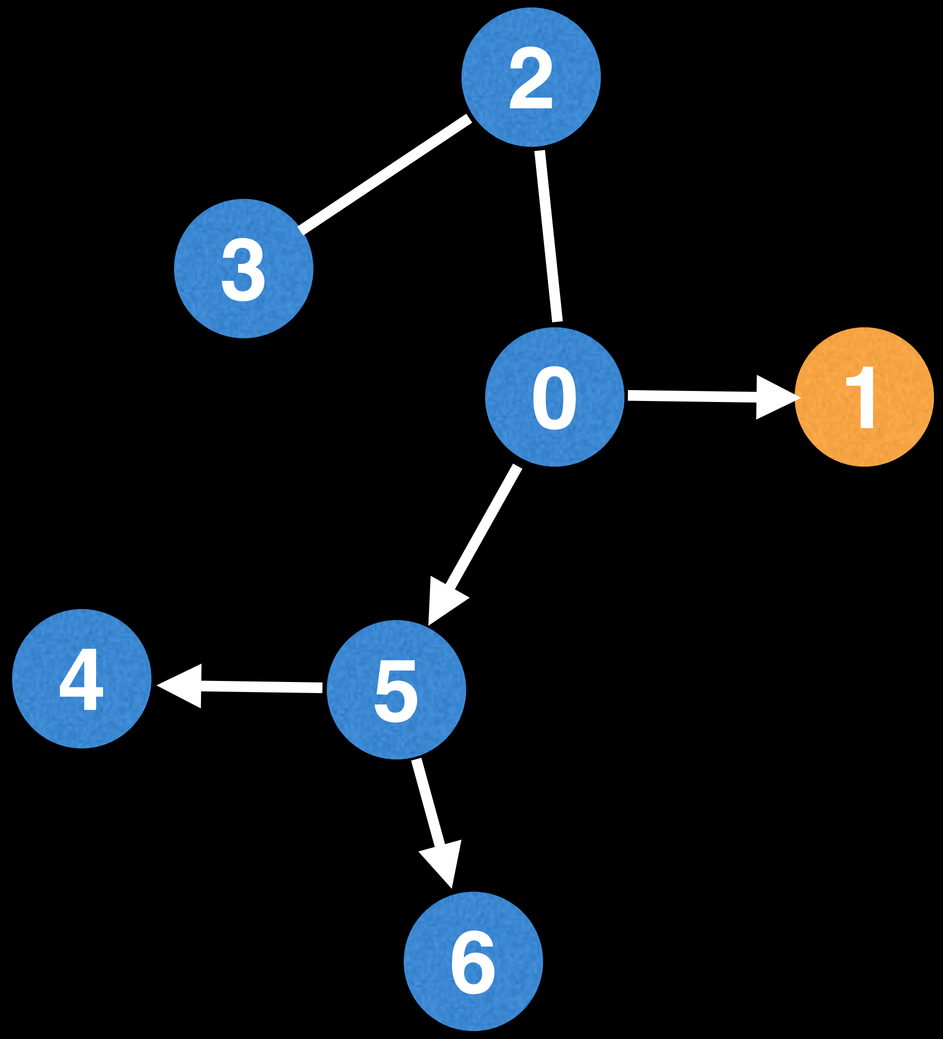


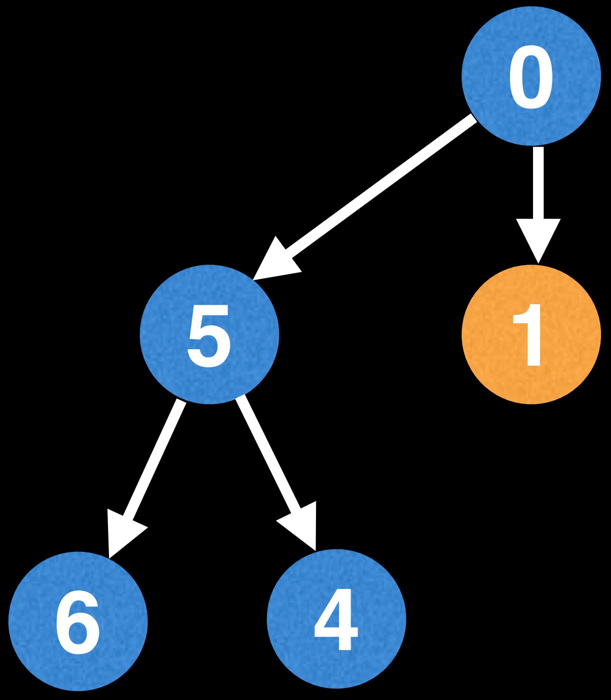
Rooting a tree is easily done depth first.



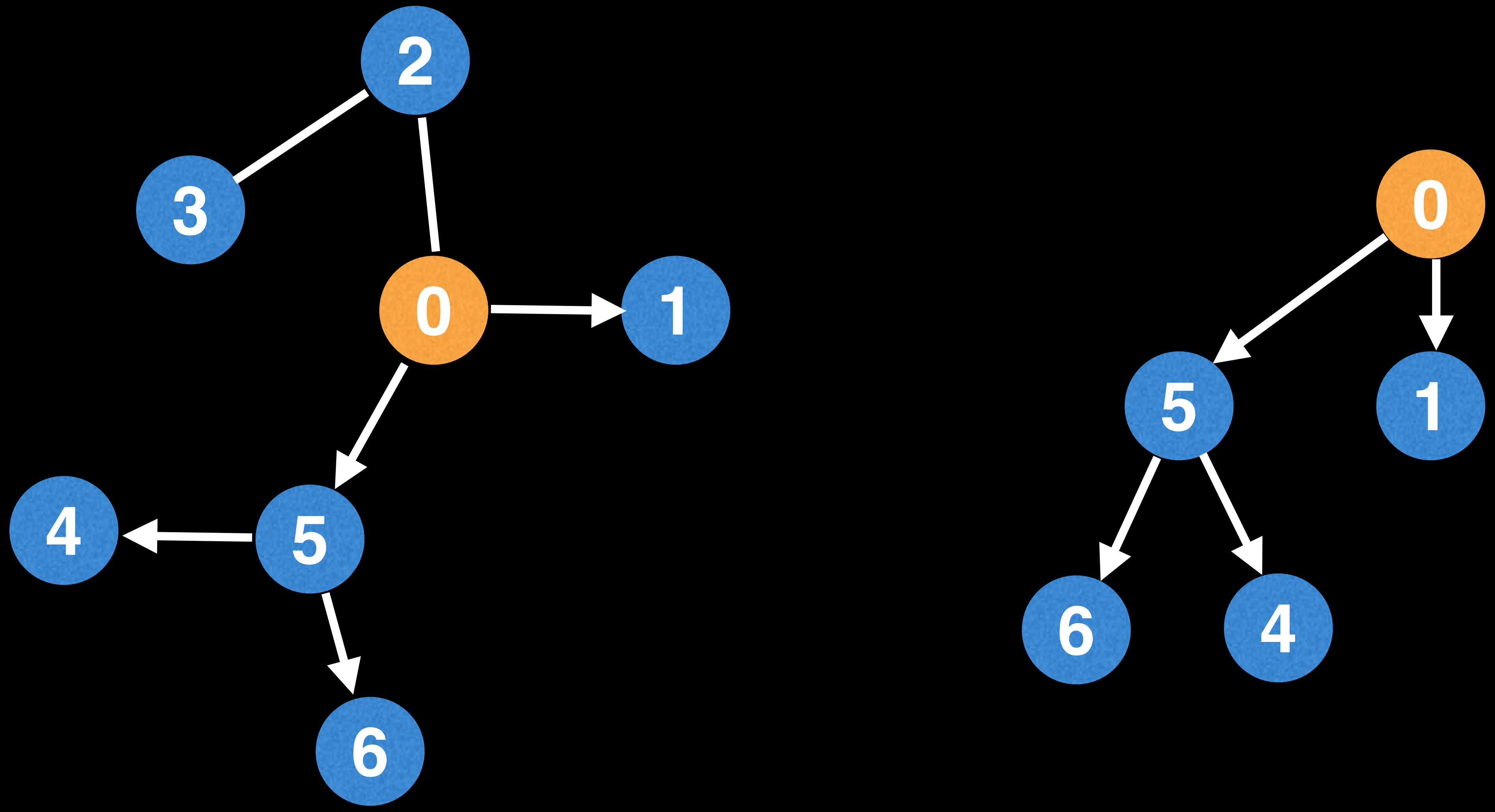


Rooting a tree is easily done depth first.

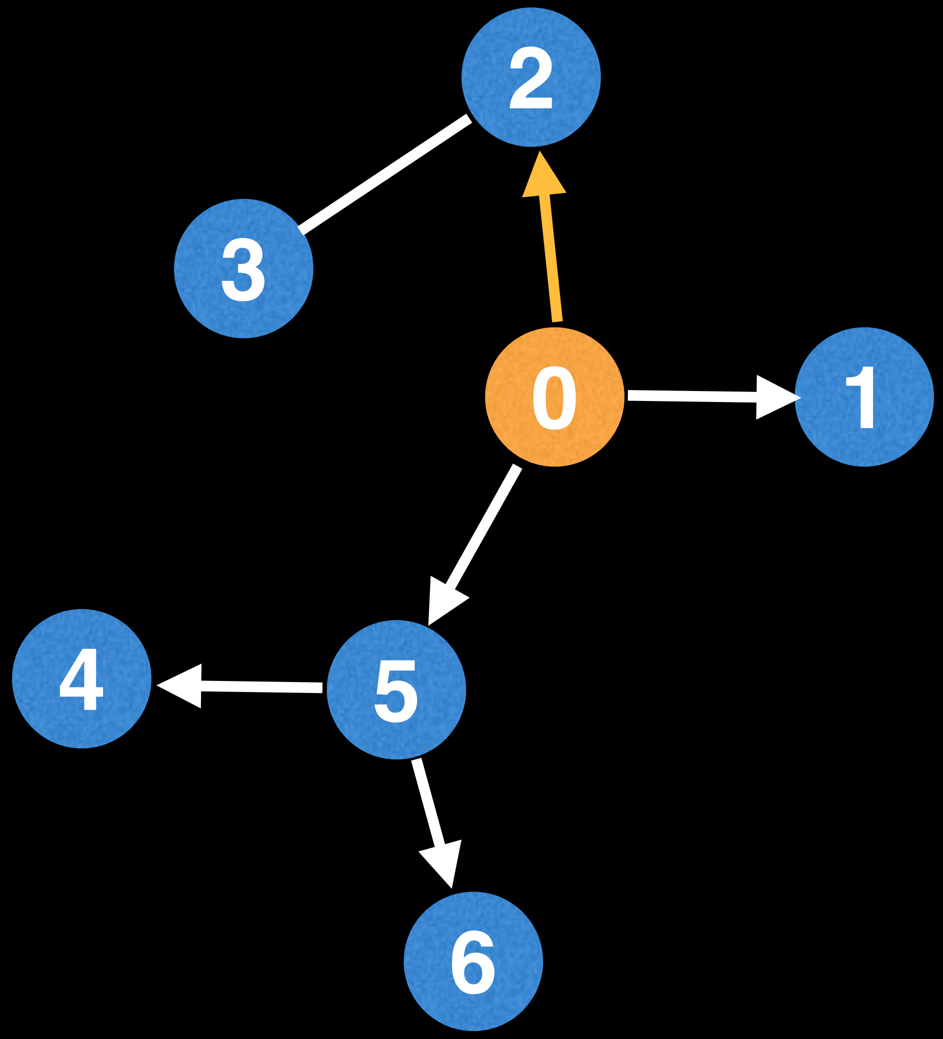


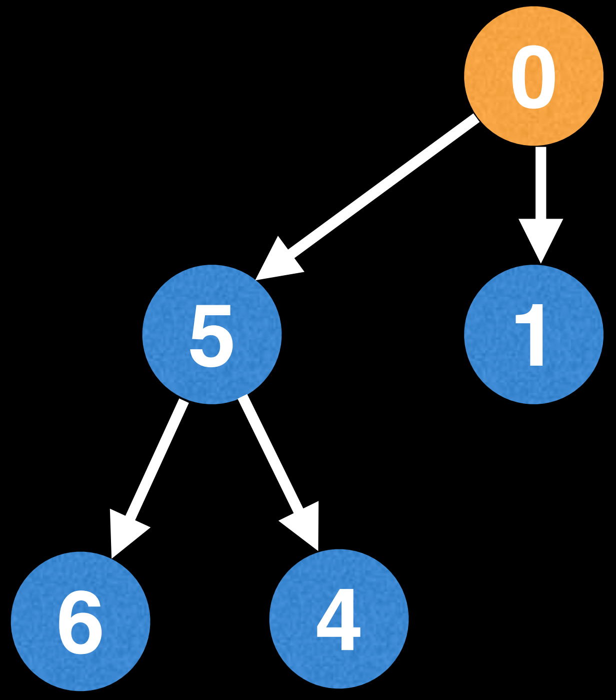


Rooting a tree is easily done depth first.

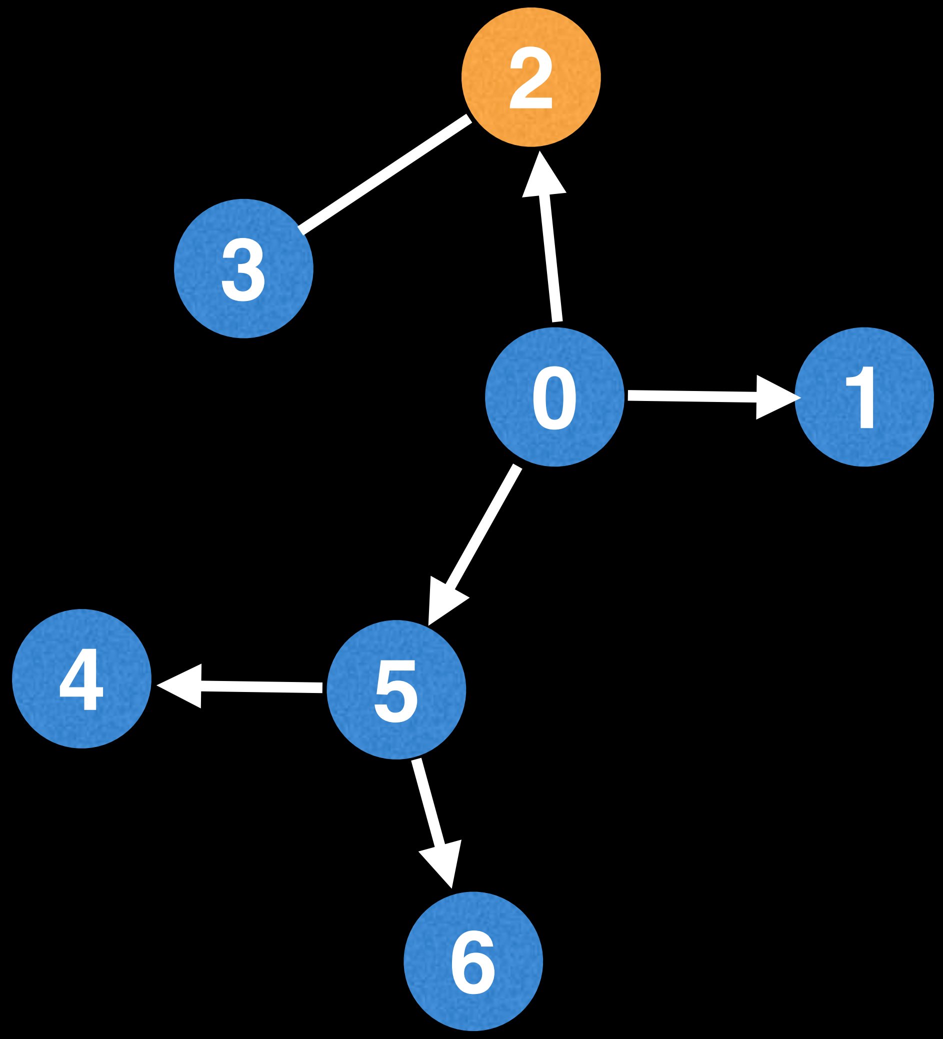


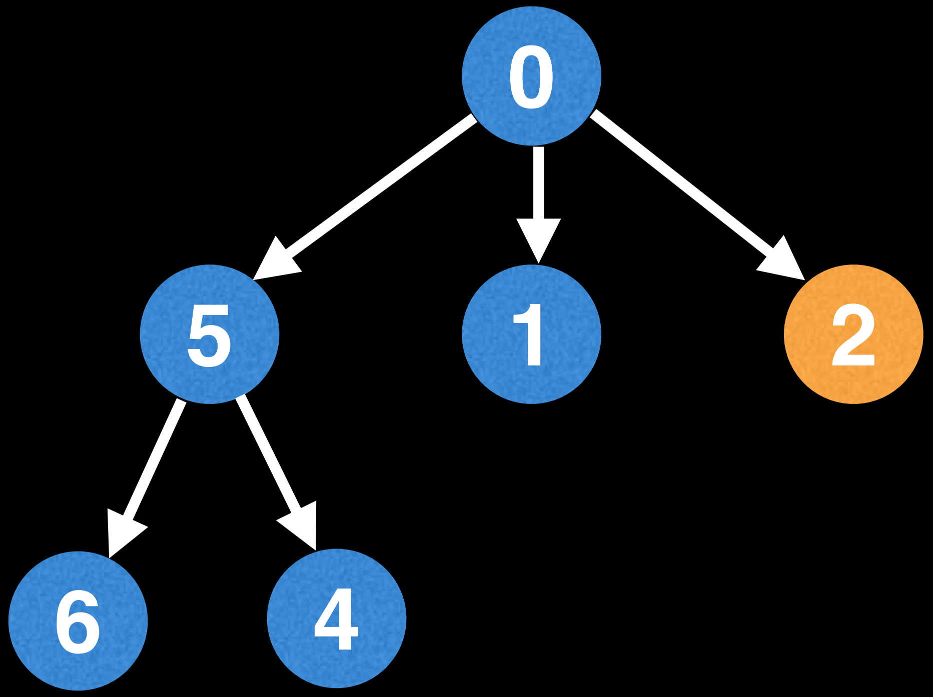
Rooting a tree is easily done depth first.



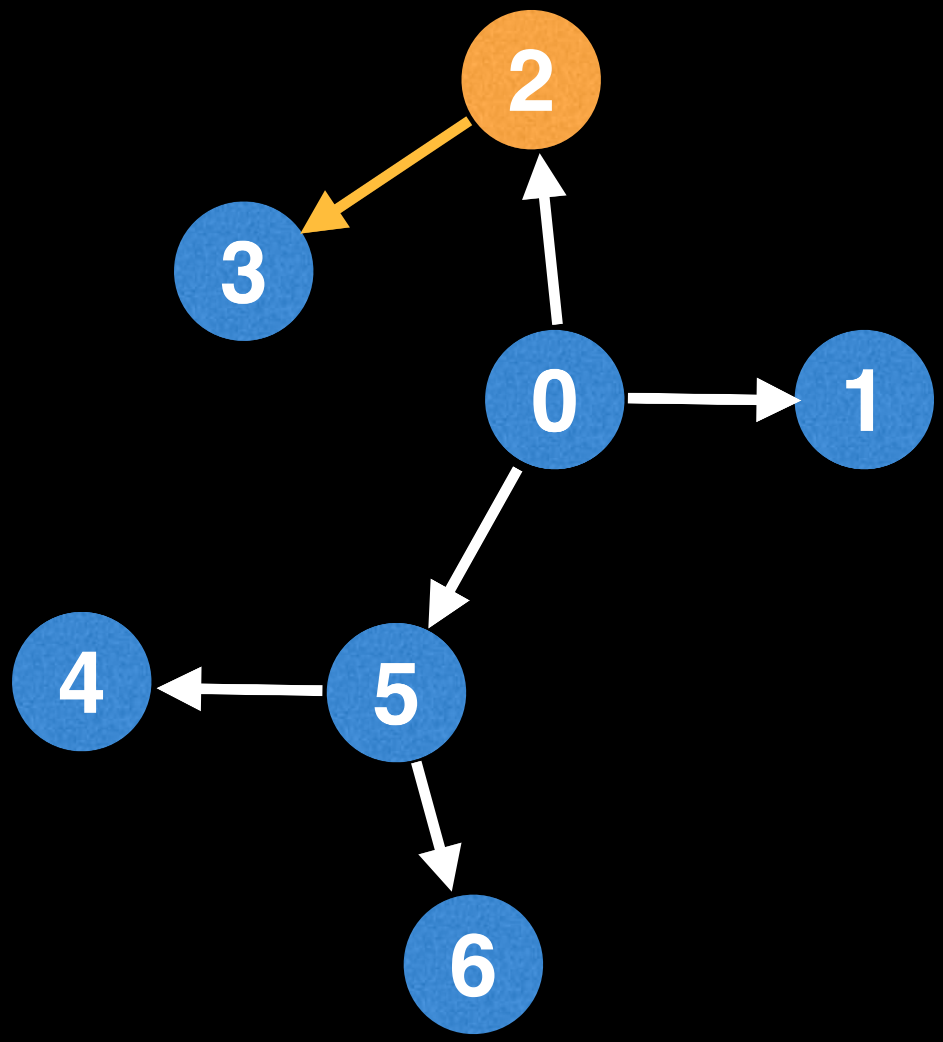


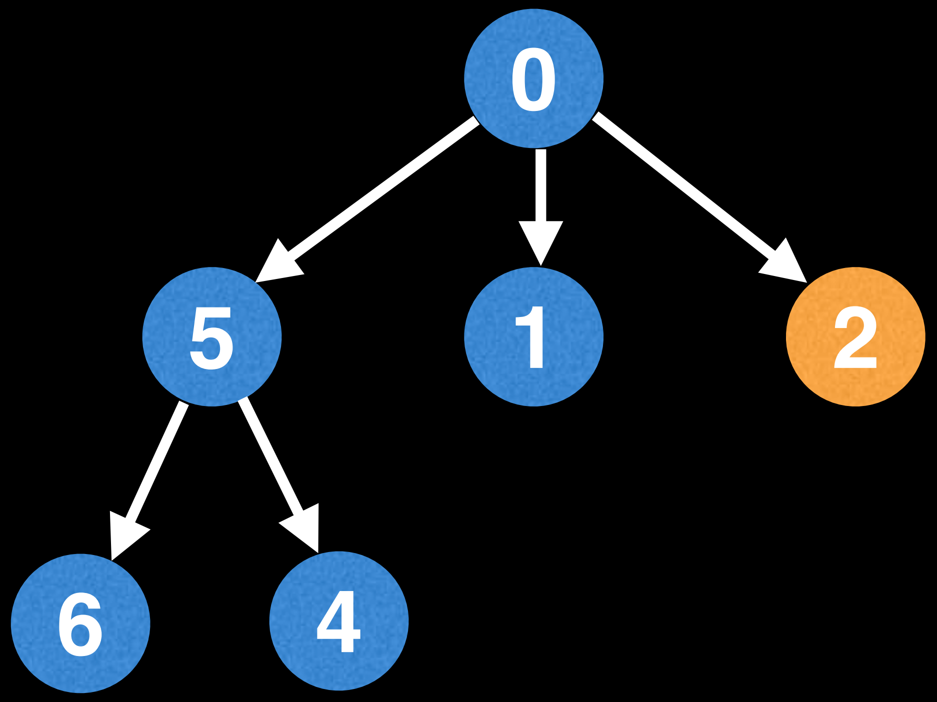
Rooting a tree is easily done depth first.



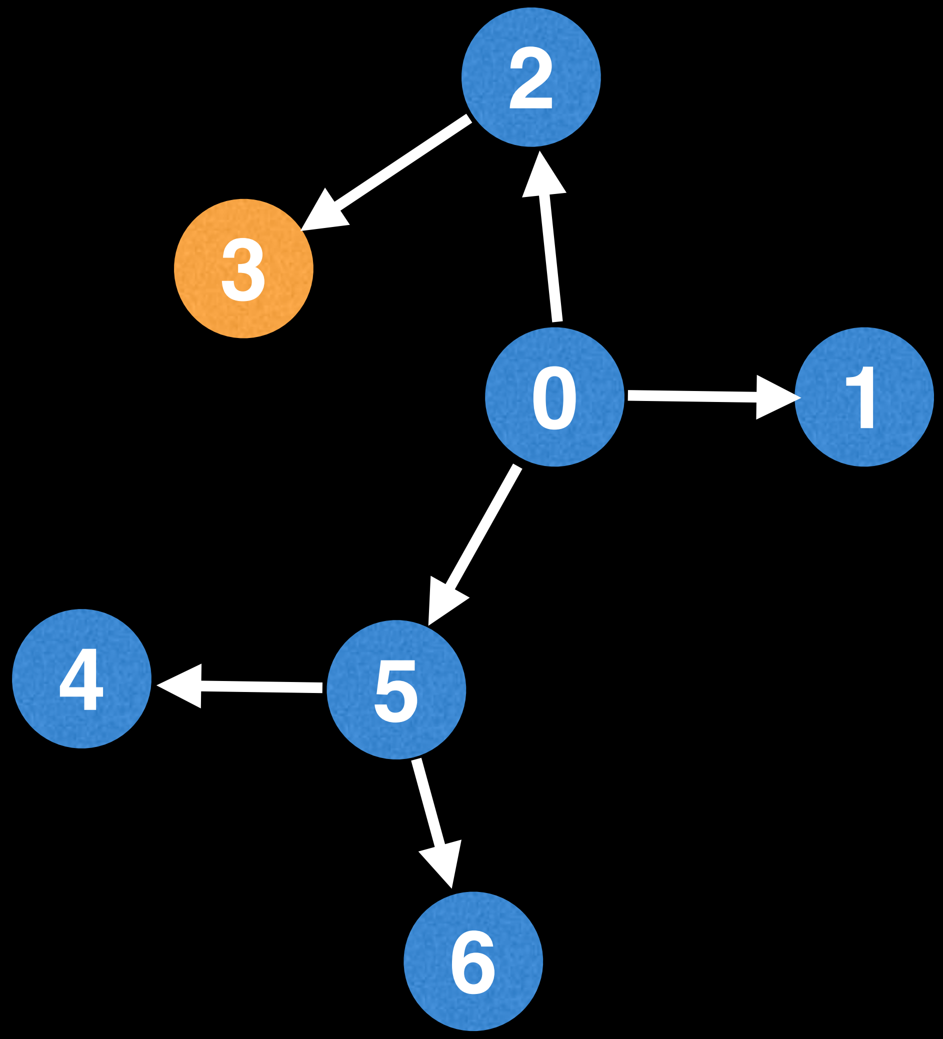


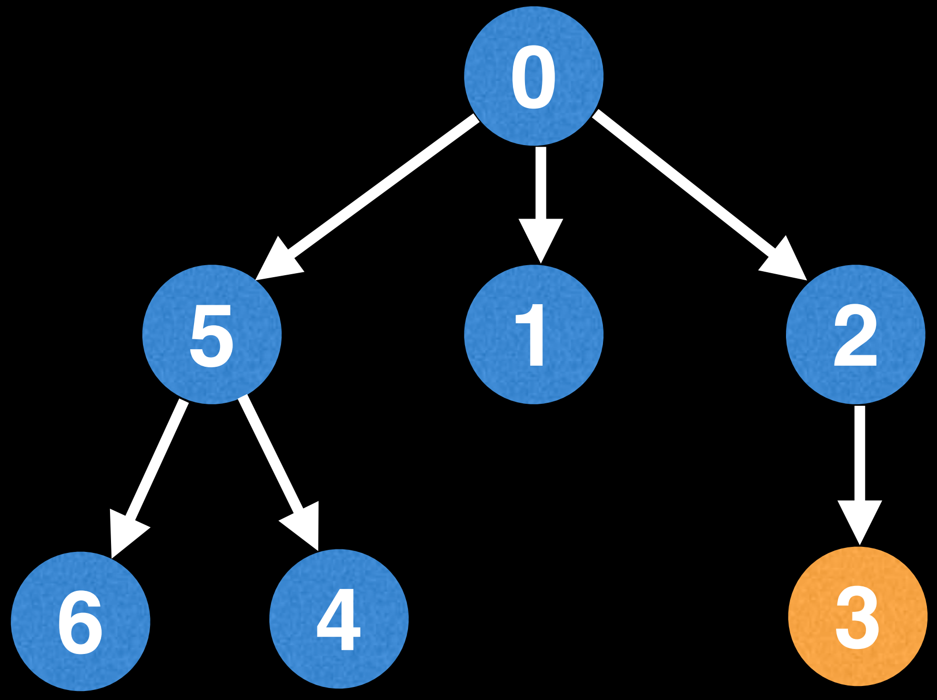
Rooting a tree is easily done depth first.



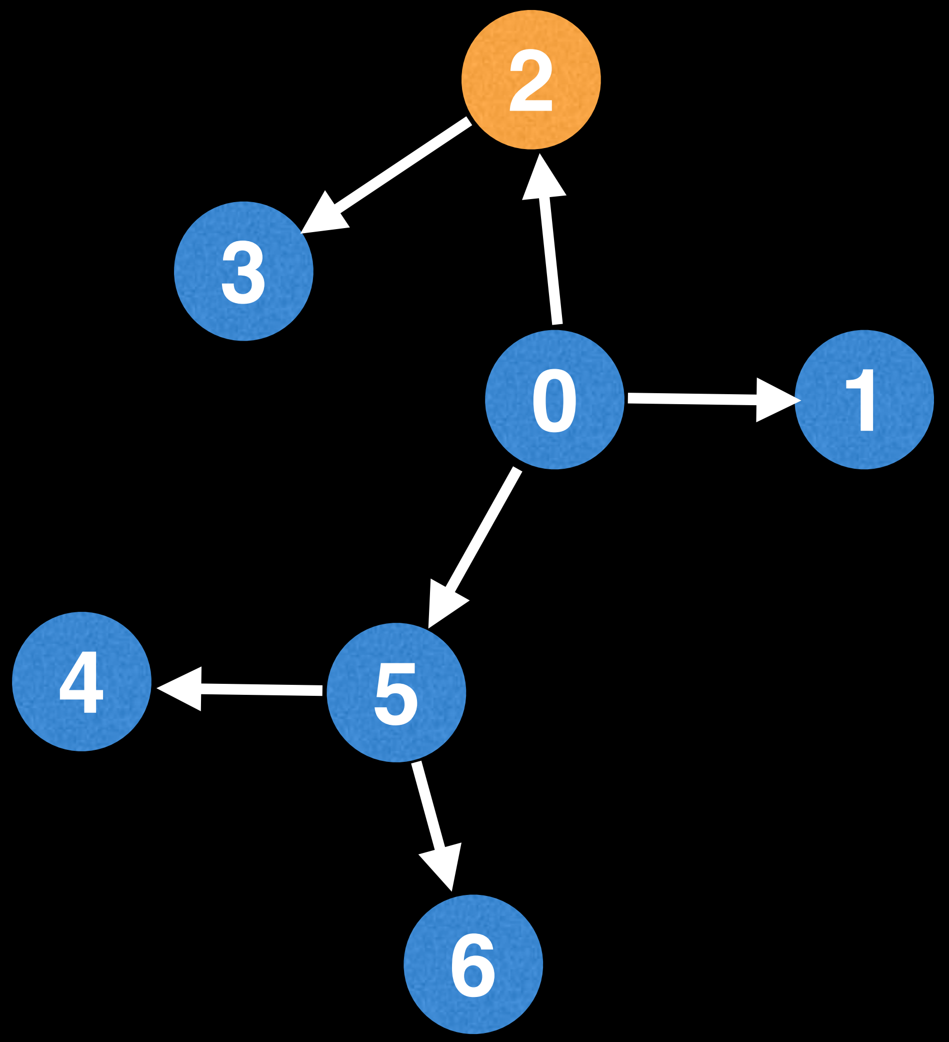


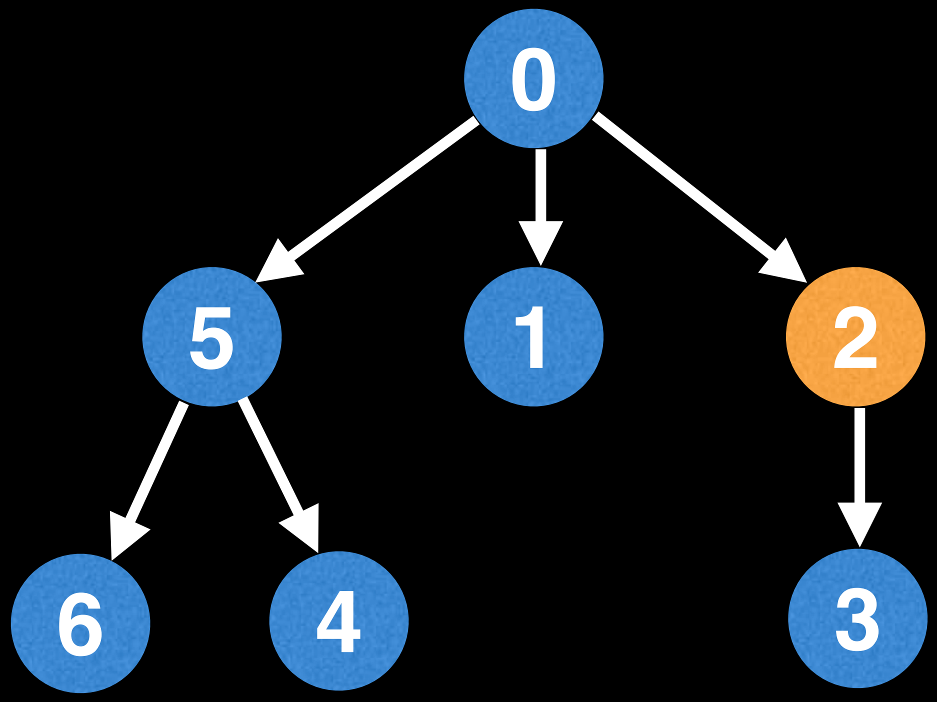
Rooting a tree is easily done depth first.



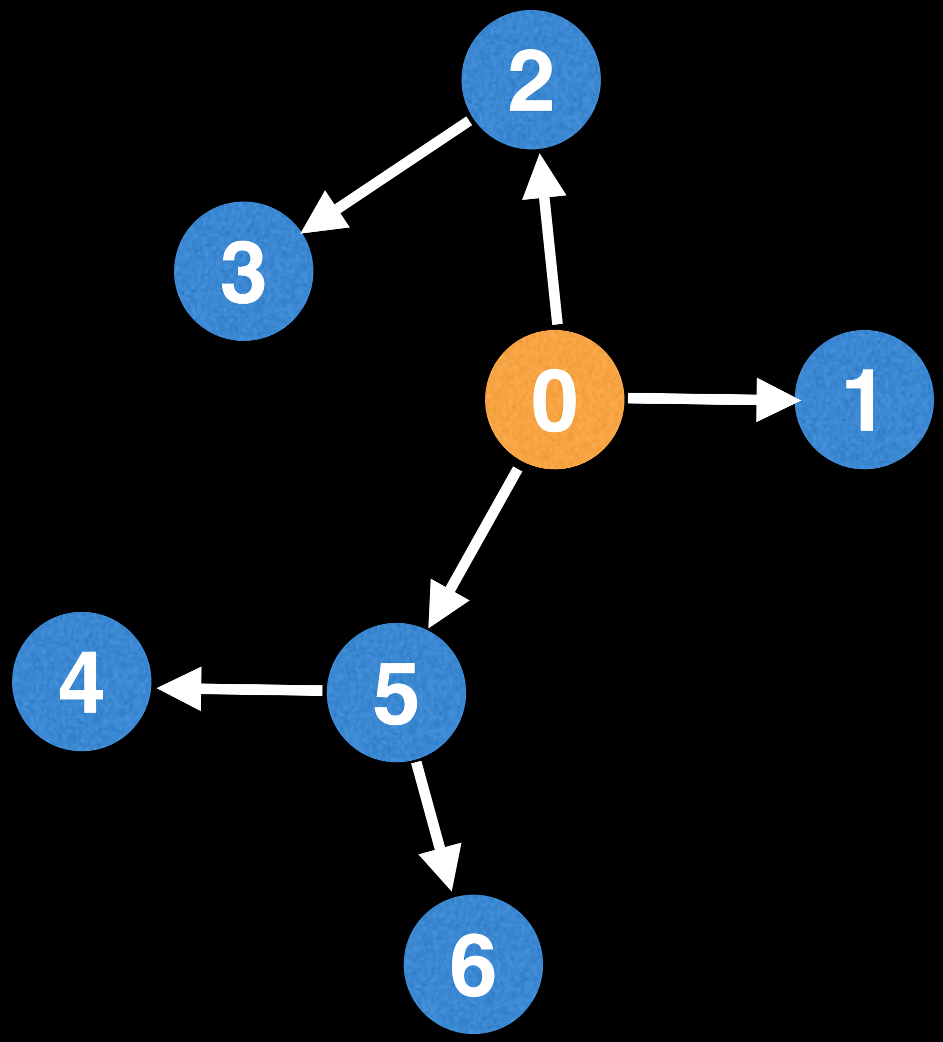


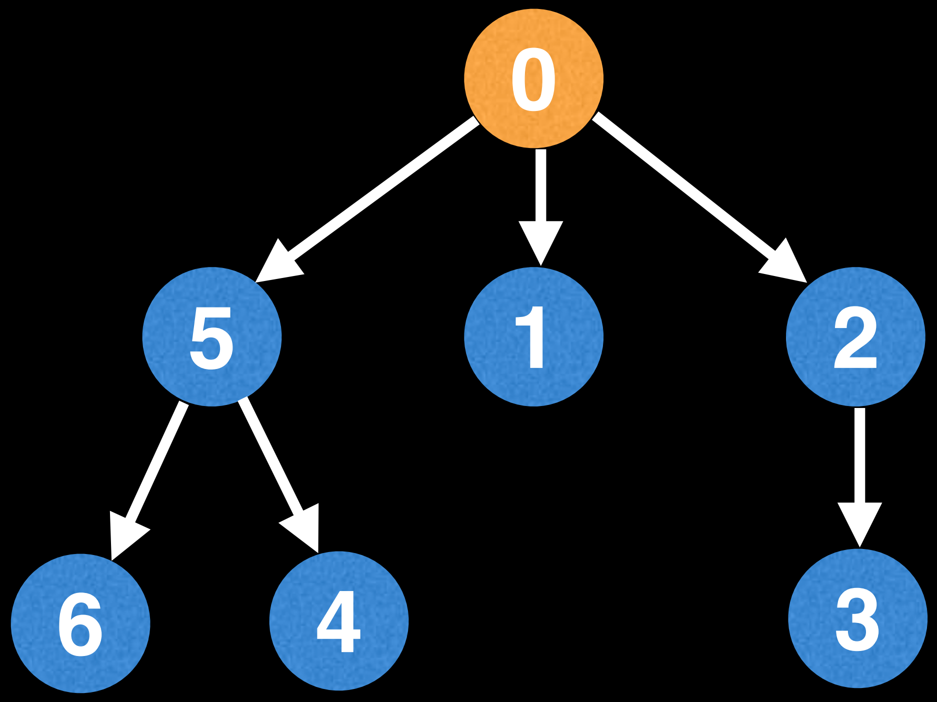
Rooting a tree is easily done depth first.



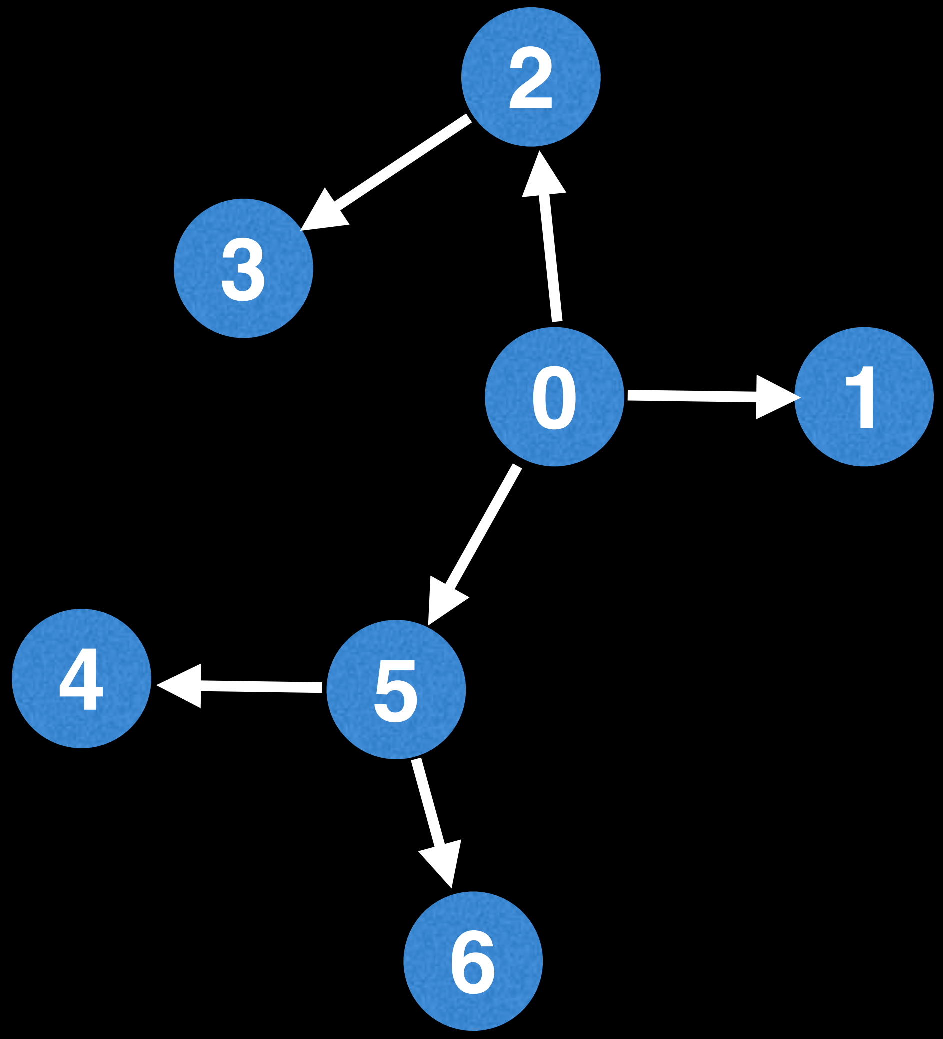


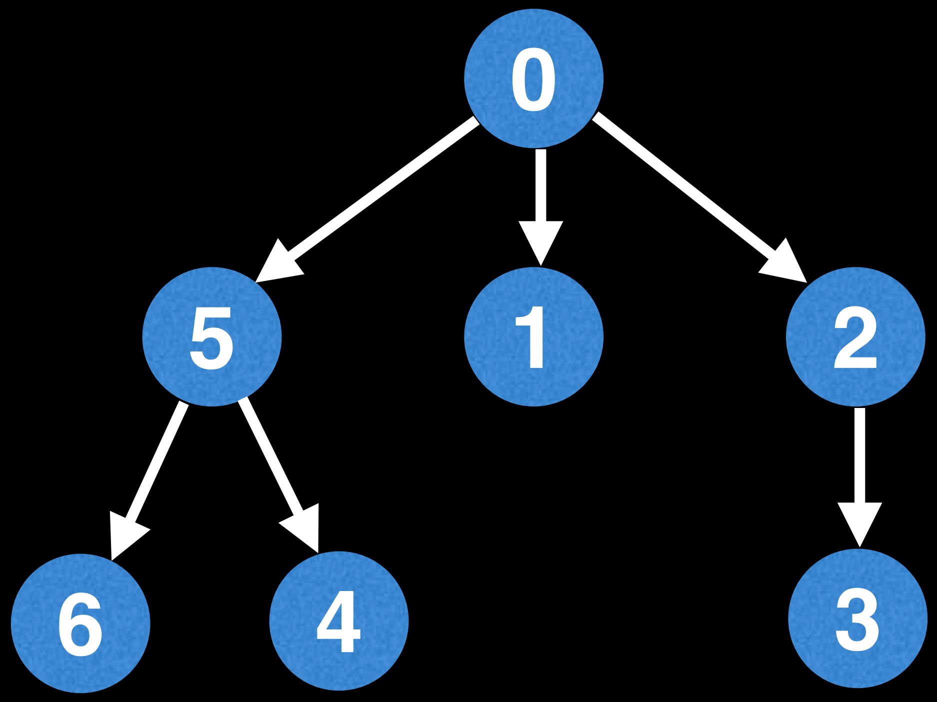
Rooting a tree is easily done depth first.





Rooting a tree is easily done depth first.





Rooting tree pseudocode

```
# TreeNode object structure.
class TreeNode:
  # Unique integer id to identify this node.
  int id;

  # Pointer to parent TreeNode reference. Only the
  # root node has a null parent TreeNode reference.
  TreeNode parent;

  # List of pointers to child TreeNodes.
  TreeNode[] children;
```

Rooting tree pseudocode

```
# TreeNode object structure.
class TreeNode:
  # Unique integer id to identify this node.
  int id;

  # Pointer to parent TreeNode reference. Only the
  # root node has a null parent TreeNode reference.
  TreeNode parent;

  # List of pointers to child TreeNodes.
  TreeNode[] children;
```

Rooting tree pseudocode

```
# TreeNode object structure.
class TreeNode:
  # Unique integer id to identify this node.
  int id;

  # Pointer to parent TreeNode reference. Only the
  # root node has a null parent TreeNode reference.
  TreeNode parent;

  # List of pointers to child TreeNodes.
  TreeNode[] children;
```

Rooting tree pseudocode

```
# TreeNode object structure.
class TreeNode:
  # Unique integer id to identify this node.
  int id;

  # Pointer to parent TreeNode reference. Only the
  # root node has a null parent TreeNode reference.
  TreeNode parent;

  # List of pointers to child TreeNodes.
  TreeNode[] children;
```

Rooting tree pseudocode

```
# TreeNode object structure.
class TreeNode:
  # Unique integer id to identify this node.
  int id;

  # Pointer to parent TreeNode reference. Only the
  # root node has a null parent TreeNode reference.
  TreeNode parent;

  # List of pointers to child TreeNodes.
  TreeNode[] children;
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there's an edge between
# (u, v) there's also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there's an edge between
# (u, v) there's also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there's an edge between
# (u, v) there's also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

root node has no parent!

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there's an edge between
# (u, v) there's also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

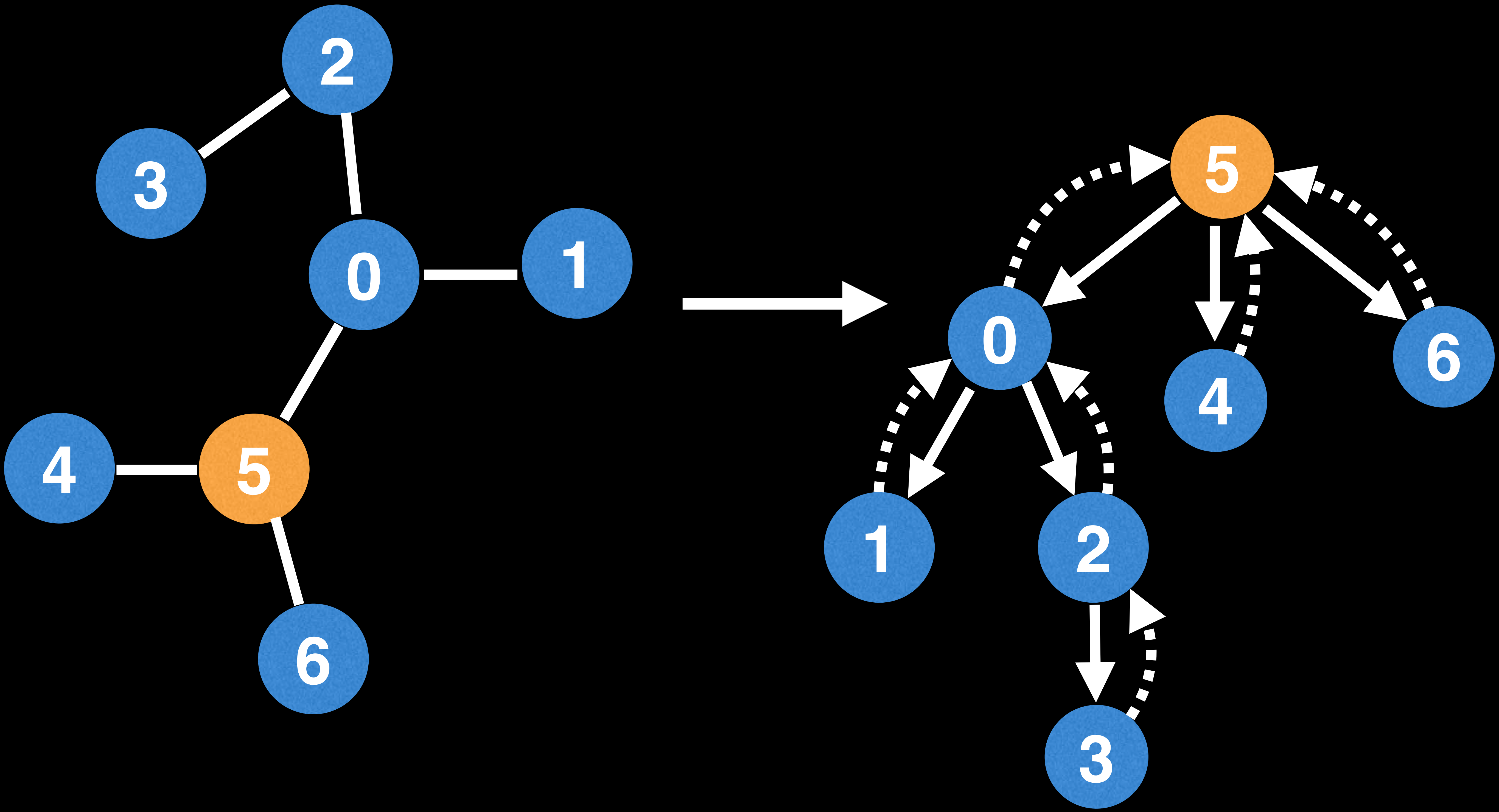
Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there's an edge between
# (u, v) there's also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting tree pseudocode

```
# g is the graph/tree represented as an adjacency
# list with undirected edges. If there’s an edge between
# (u, v) there’s also an edge between (v, u).
# rootId is the id of the node to root the tree from.
function rootTree(g, rootId = 0):
  root = TreeNode(rootId, null, [])
  return buildTree(g, root, null)

# Build tree recursively depth first.
function buildTree(g, node, parent):
  for childId in g[node.id]:
    # Avoid adding an edge pointing back to the parent.
    if parent != null and childId == parent.id:
      continue
    child = TreeNode(childId, node, [])
    node.children.add(child)
    buildTree(g, child, node)
  return node
```

Rooting a Tree


2
3
0
1
4
5
6
5
0
4
6
1
2
3