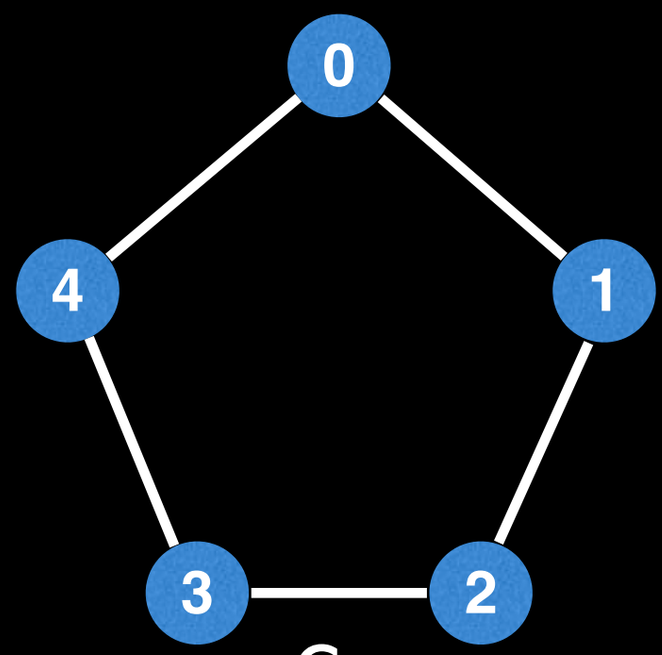
Isomorphisms in trees

A question of equality

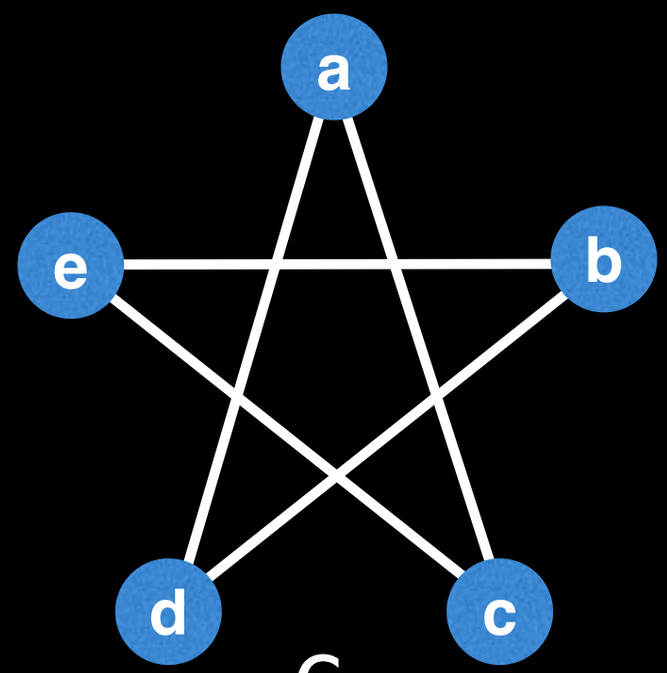
William Fiset

Graph Isomorphism

The question of asking whether two graphs G_1 and G_2 are **isomorphic** is asking whether they are *structurally* the same.



G_1



G_2

Even though G_1 and G_2 are labelled differently and may appear different they are structurally the same graph.

Graph Isomorphism

We can also define the notion of a graph isomorphism more rigorously:

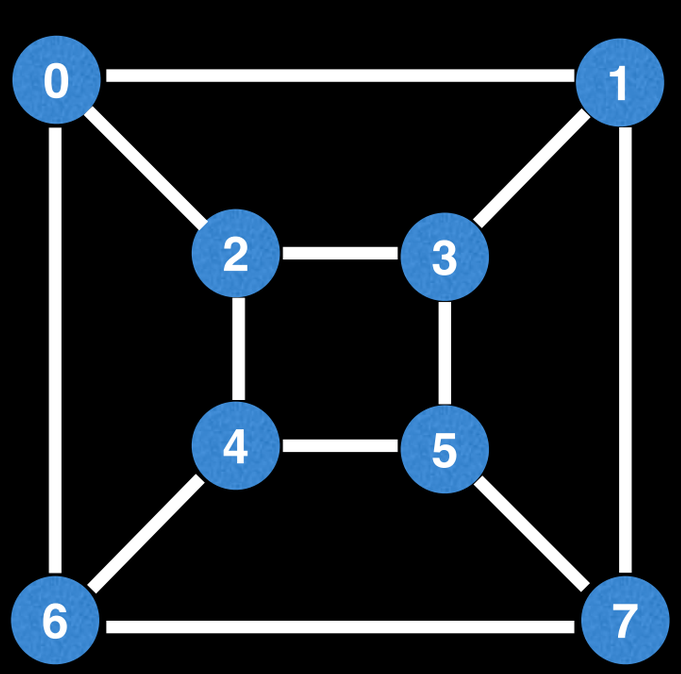
$G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are isomorphic if there exists a **bijection** φ between the sets V_1 -> V_2 such that:

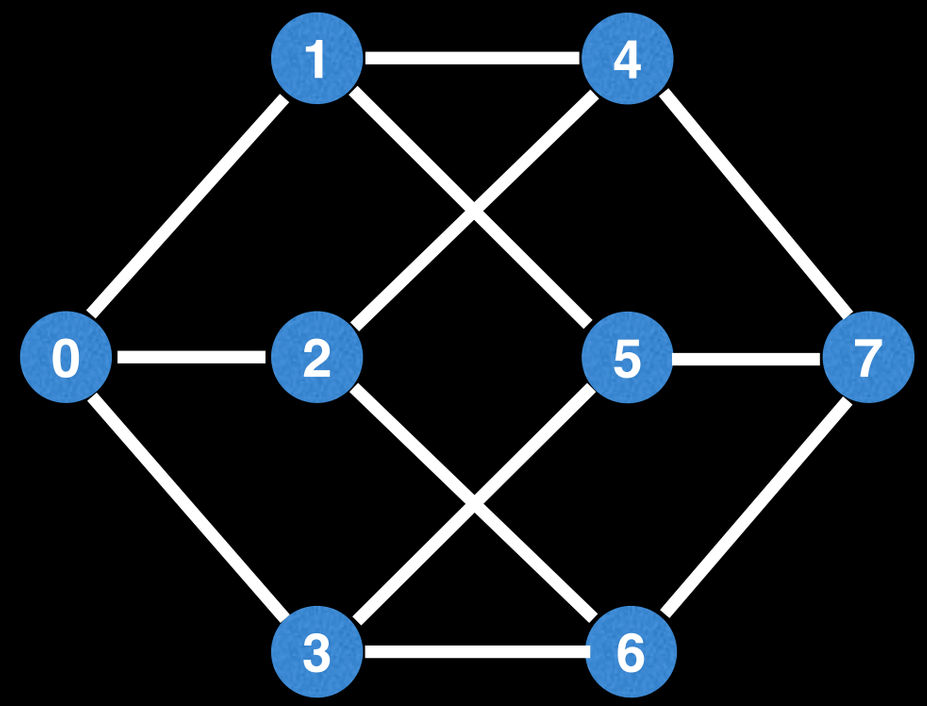
$$\forall\ u,v \in V_1,\ (u,v) \in E_1 \iff (\varphi(u),\varphi(v)) \in E_2$$

In simple terms, for an isomorphism to exist there needs to be a function φ which can map all the nodes/edges in G_1 to G_2 and vice-versa.

Graph Isomorphism

Determining if two graphs are isomorphic is not only not obvious to the human eye, but also a difficult problem for computers.



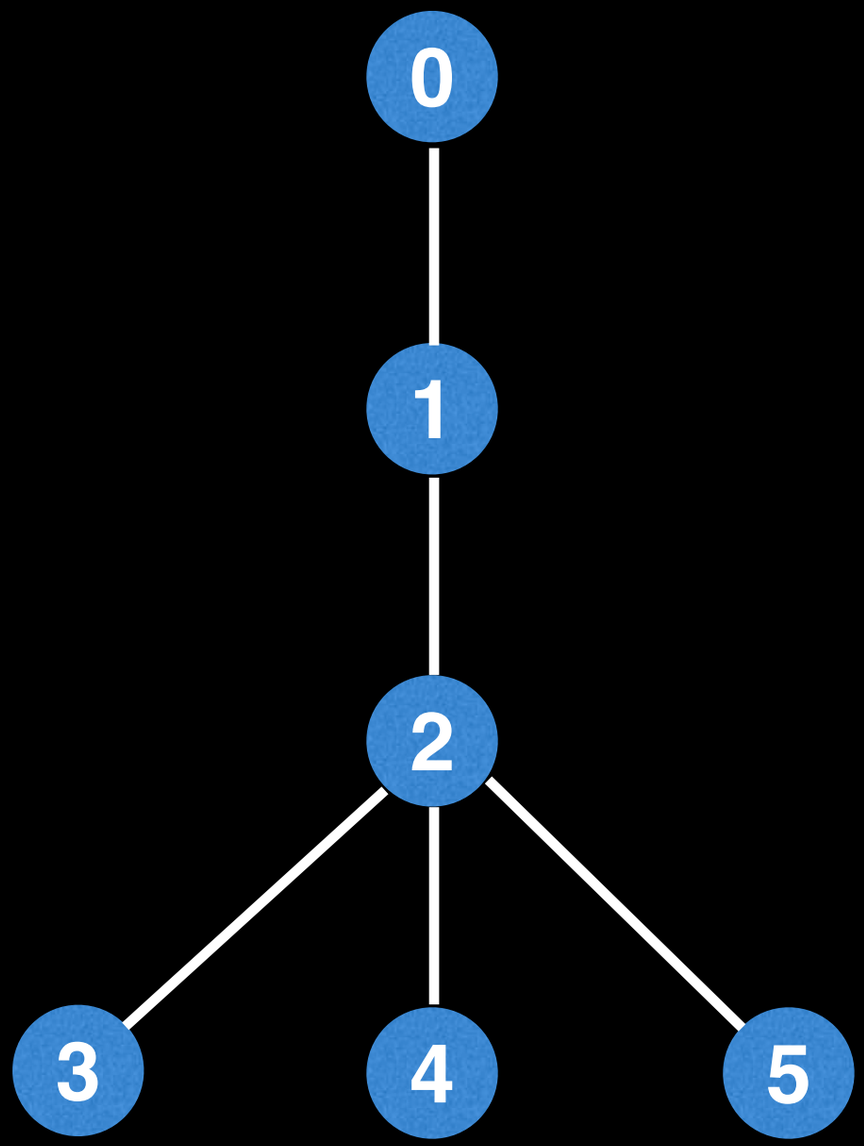


It is still an open question as to whether the graph isomorphism problem is NP complete. However, many polynomial time isomorphism algorithms exist for graph subclasses such as trees.

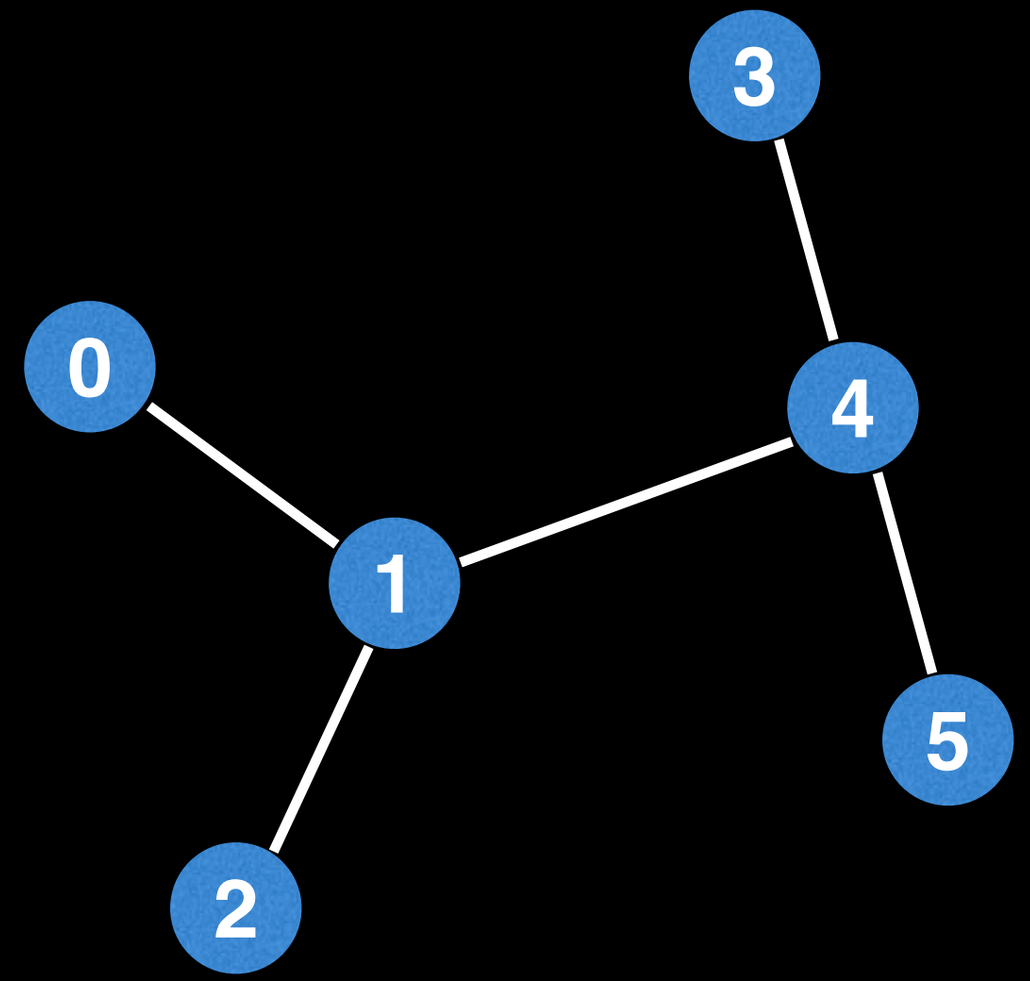
Isomorphic Trees

Isomorphic Trees

tree 1



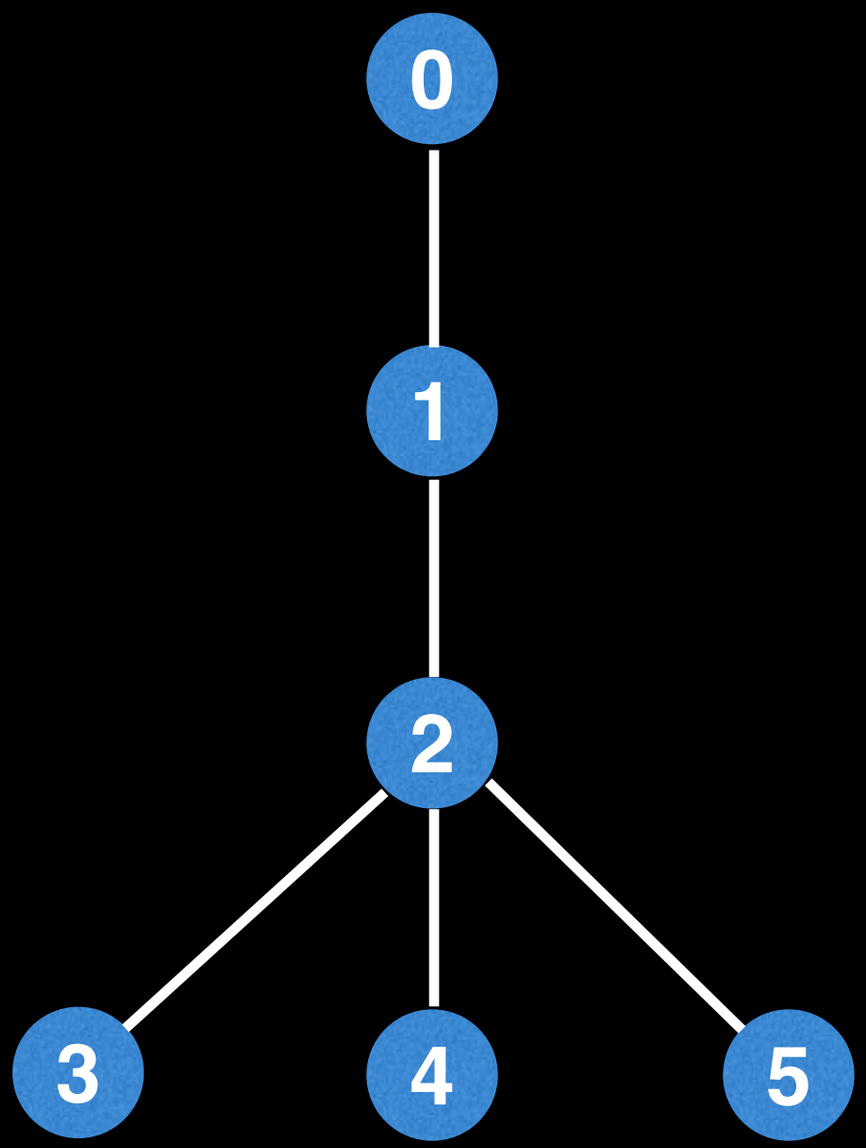
tree 2



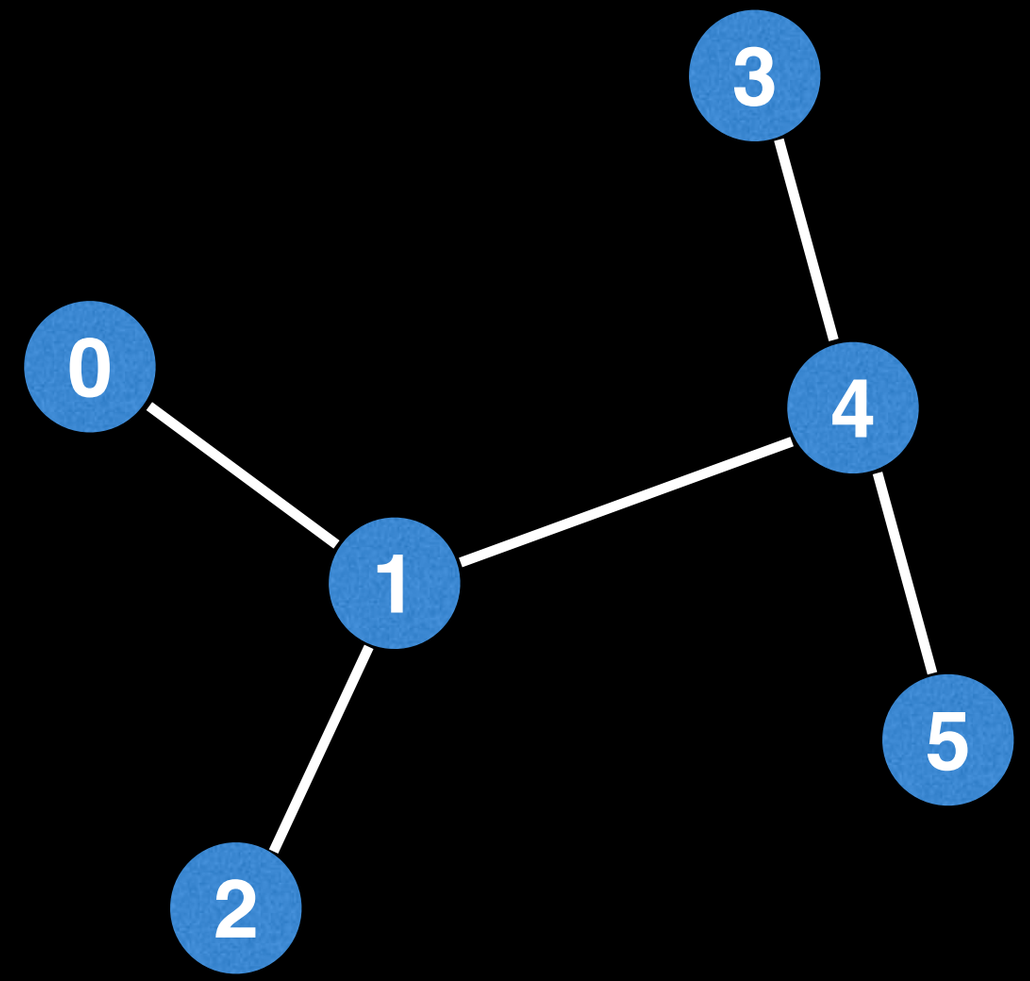
Q: Are these trees isomorphic?

Isomorphic Trees

tree 1



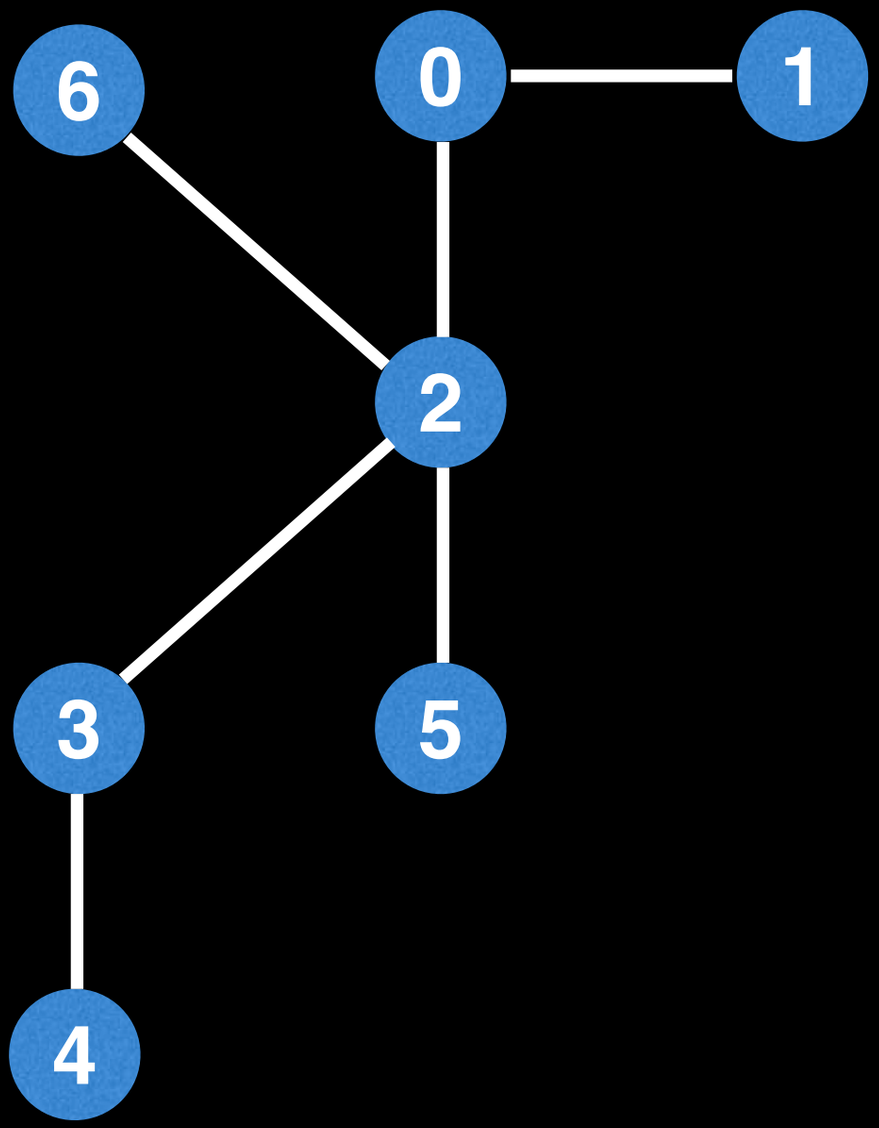
tree 2



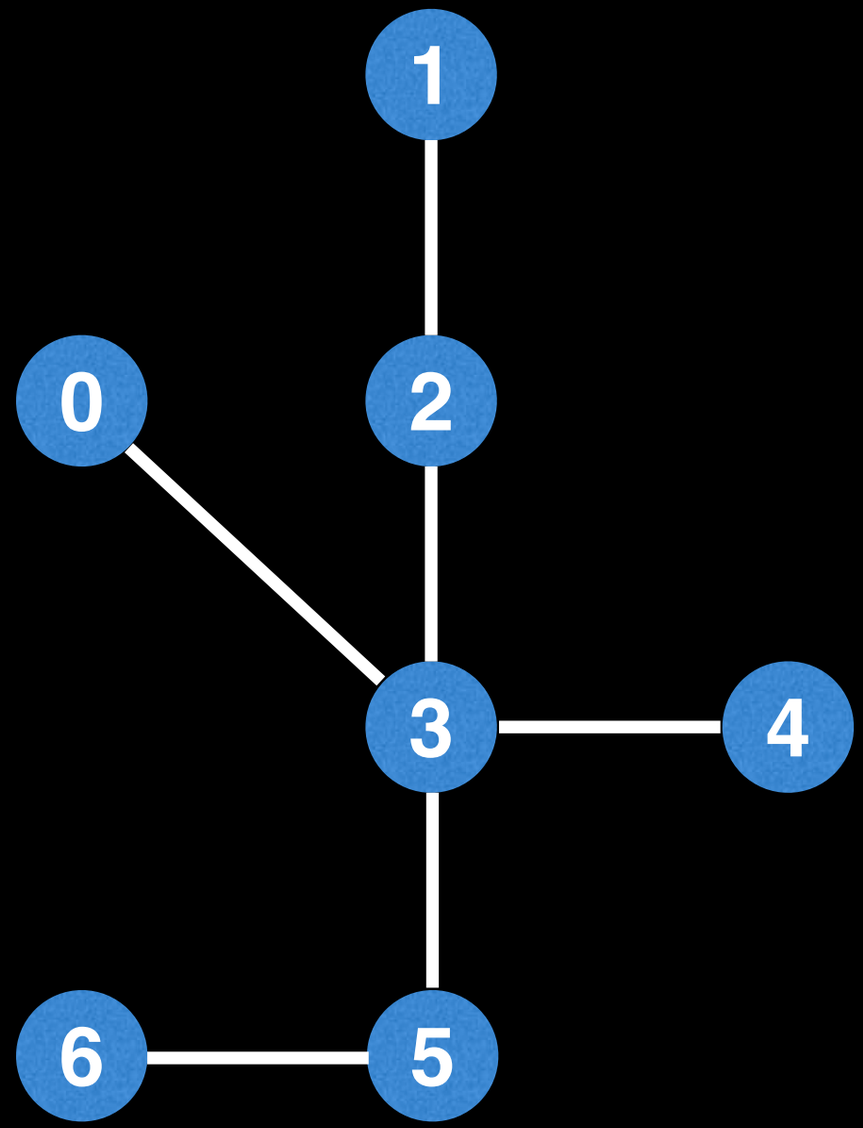
A: no, these trees are structurally different.

Isomorphic Trees

tree 3



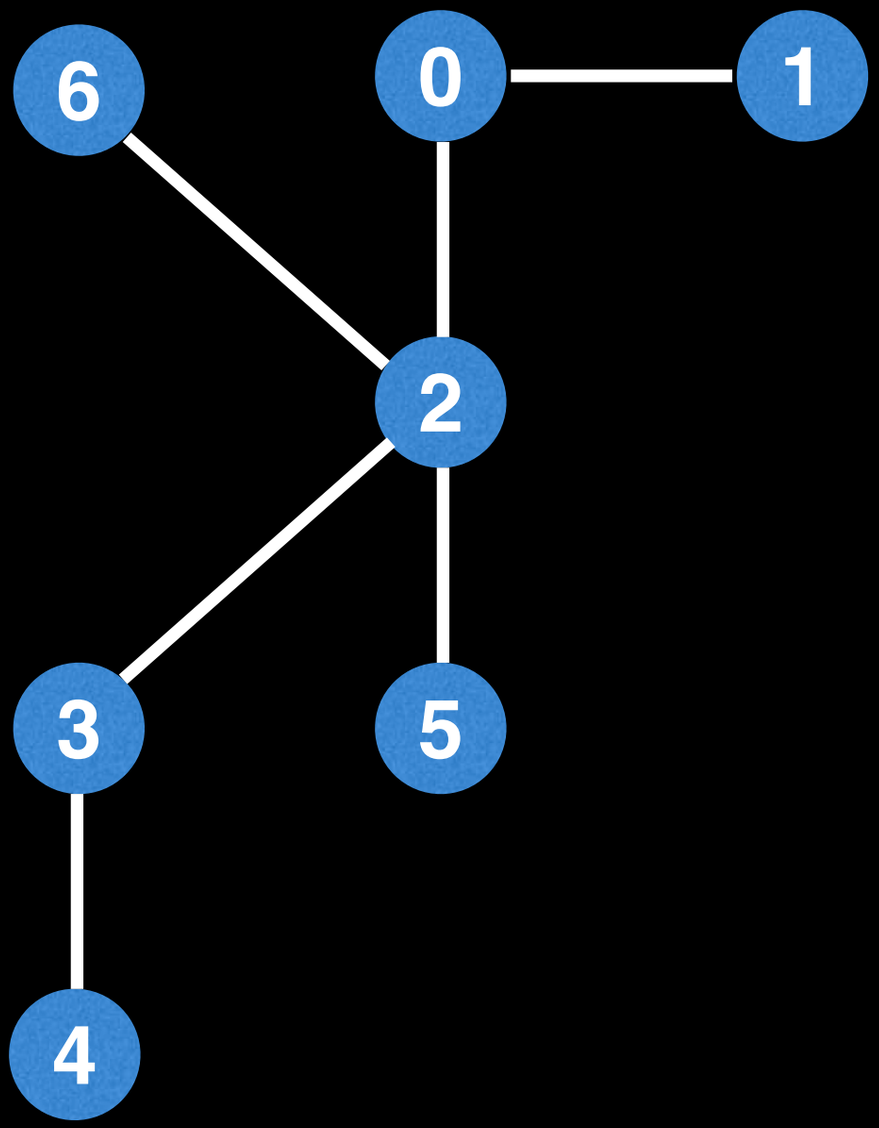
tree 4



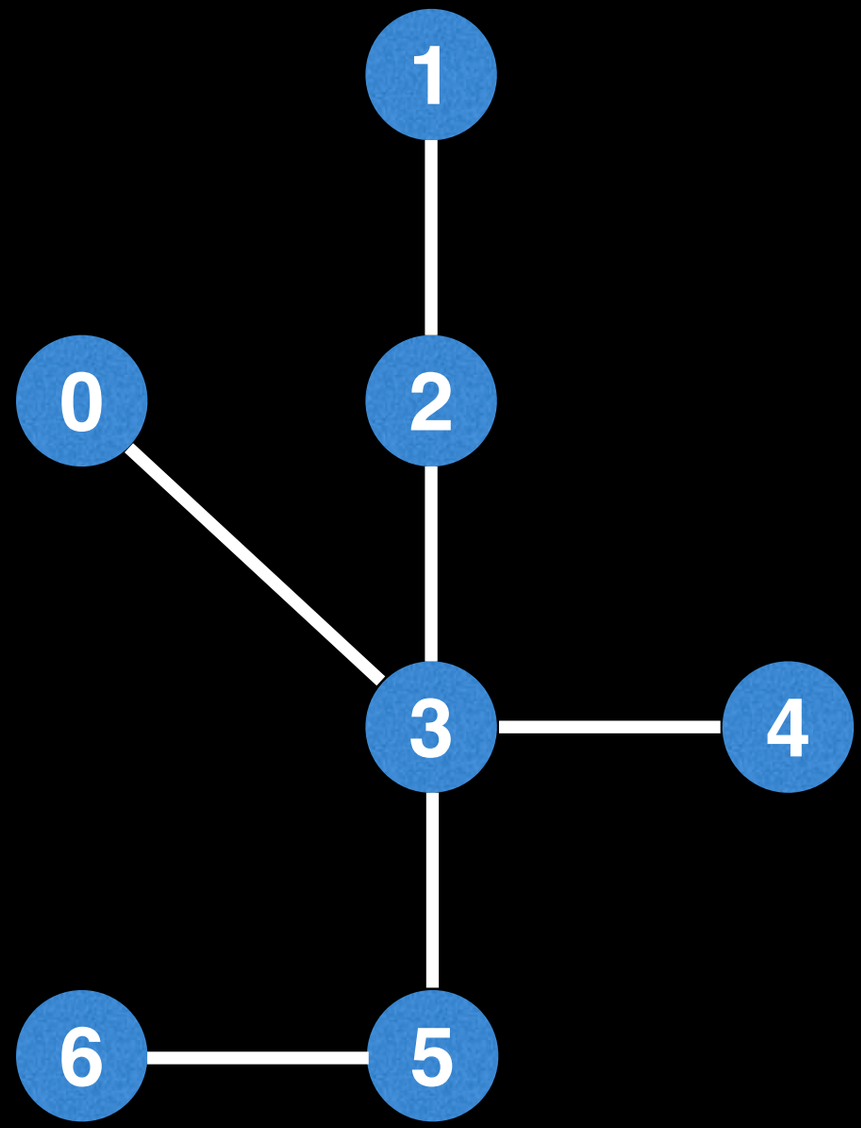
Q: Are these trees isomorphic?

Isomorphic Trees

tree 3



tree 4



Yes, one possible label mapping is:
6->0, 1->1, 0->2, 2->3, 5->4, 3->5, 4->6

Identifying Isomorphic Trees

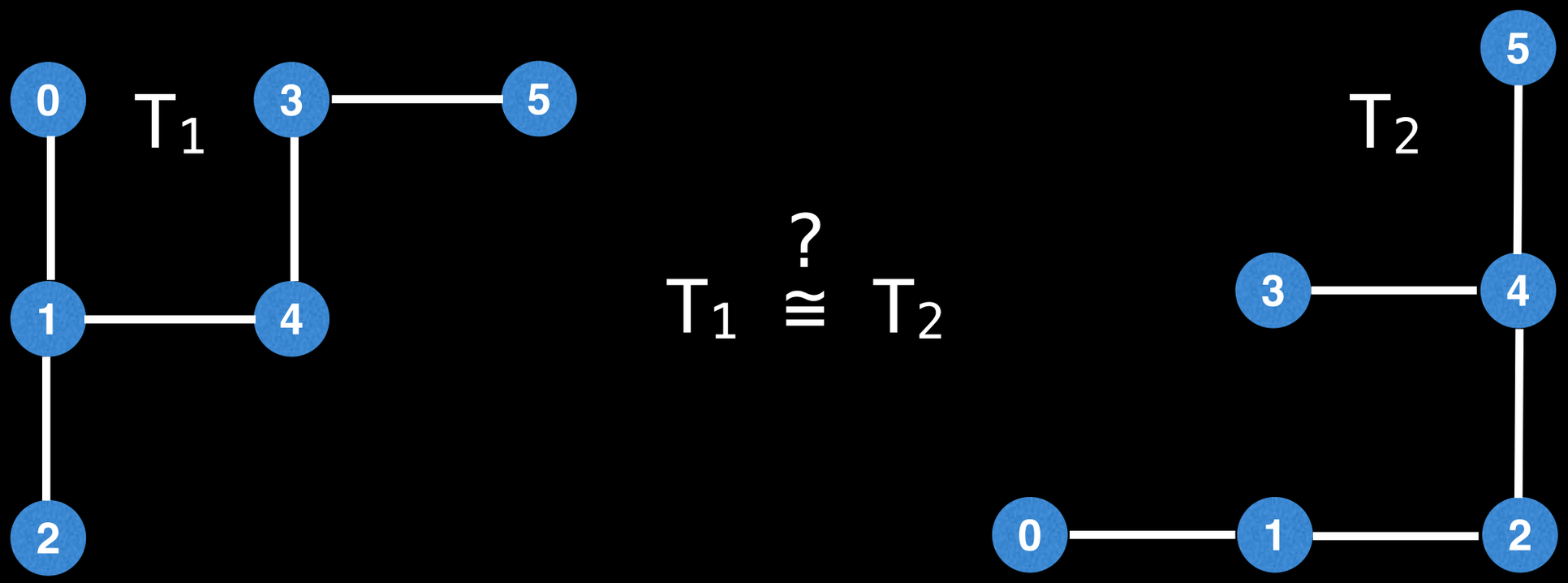
There are several very quick **probabilistic** (usually hash or heuristic based) algorithms for identifying isomorphic trees. These tend to be fast, but also error prone due to hash collisions in a limited integer space.

The method we'll be looking at today involves **serializing** a tree into a **unique encoding**. This unique encoding is simply a unique string that represents a tree, if another tree has the same encoding then they are isomorphic.

Identifying Isomorphic Trees

We can directly serialize an unrooted tree, but in practice serializing a rooted tree is typically easier code wise.

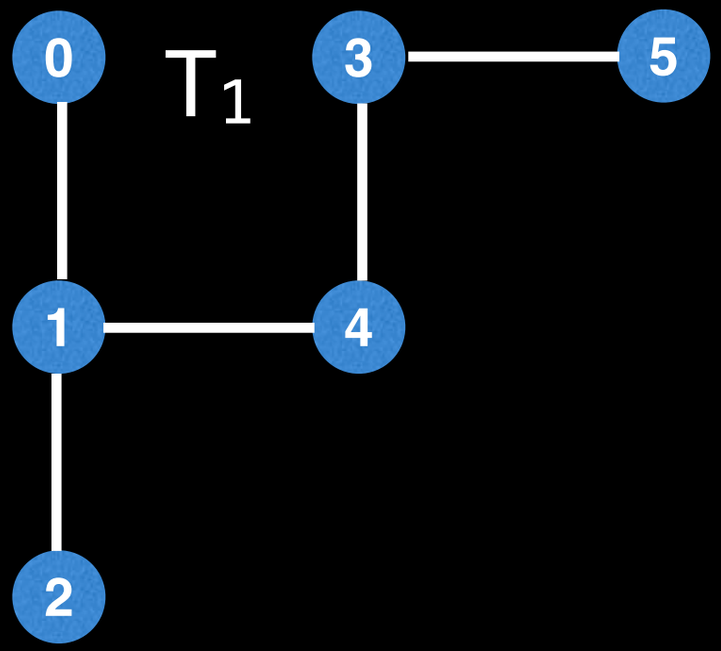
However, one caveat to watch out for if we're going to root our two trees T_1 and T_2 to check if they're isomorphic is to ensure that the same root node is selected in both trees before serializing/encoding the trees.

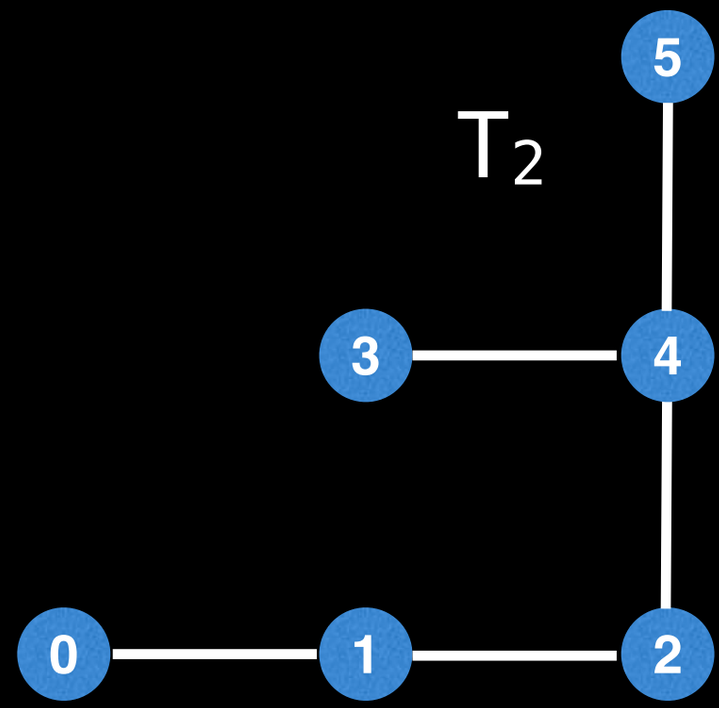


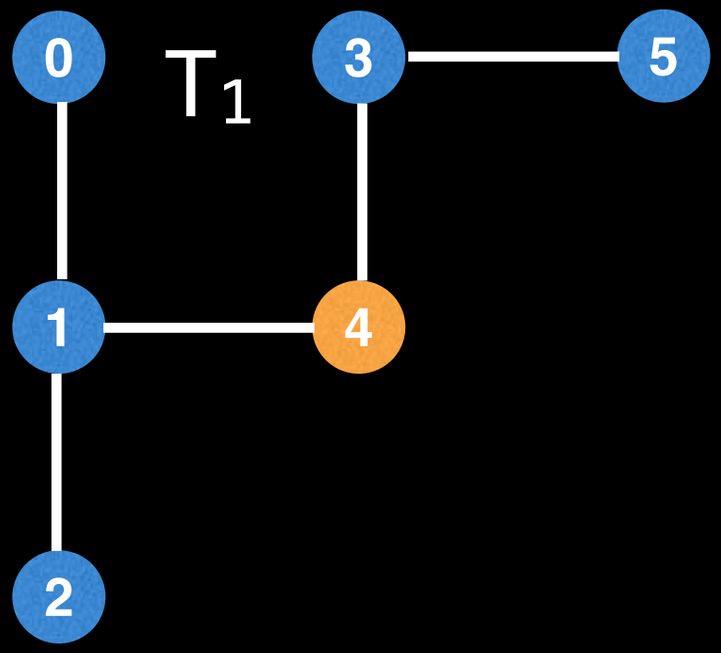
Identifying Isomorphic Trees

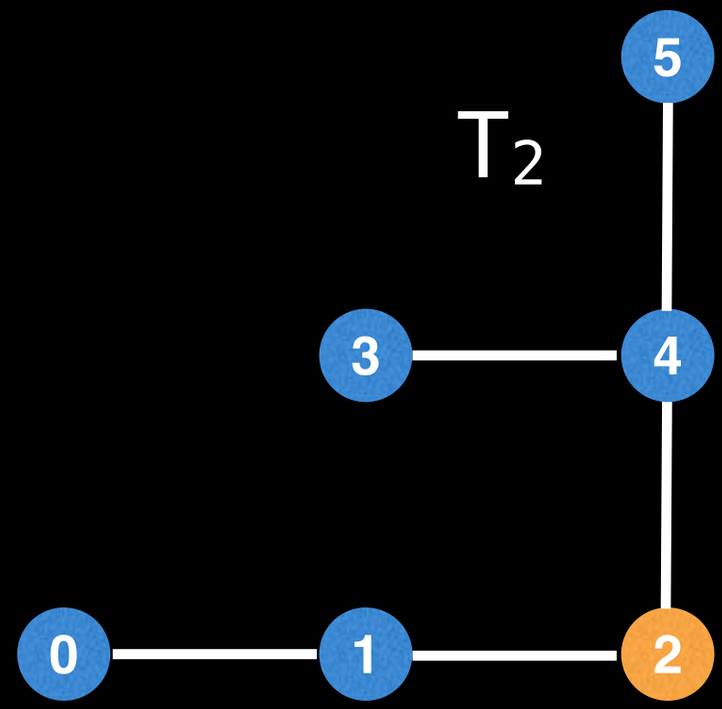
To select a common node between both trees we can use what we learned from finding the center(s) of a tree to help ourselves.

<insert video frame>

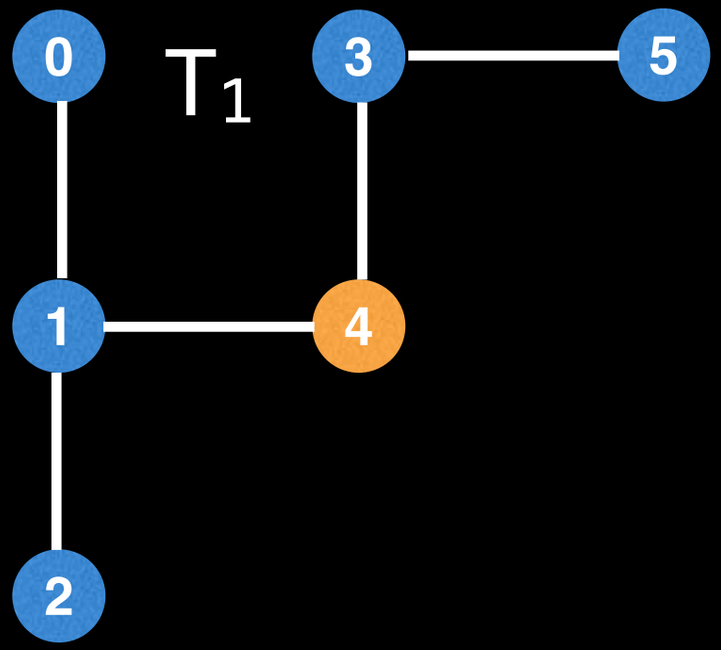

0
T_1
3
5
1
4
2

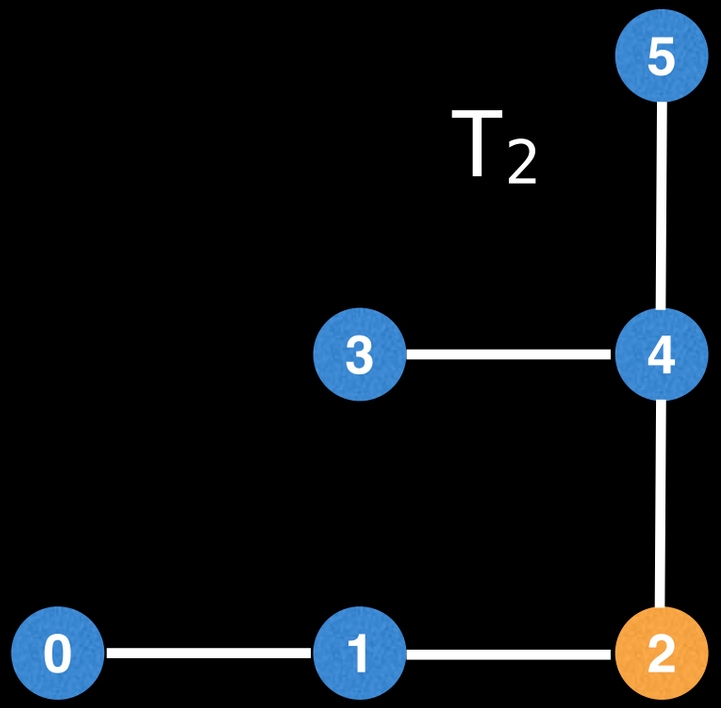

5
T_2
3
4
0
1
2

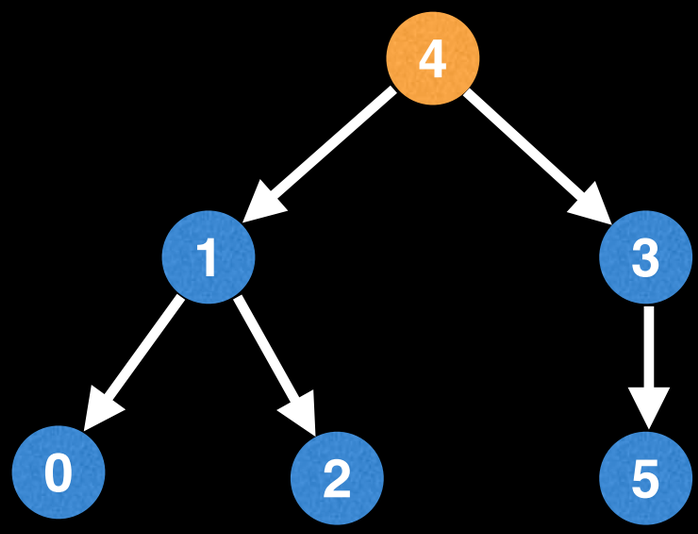


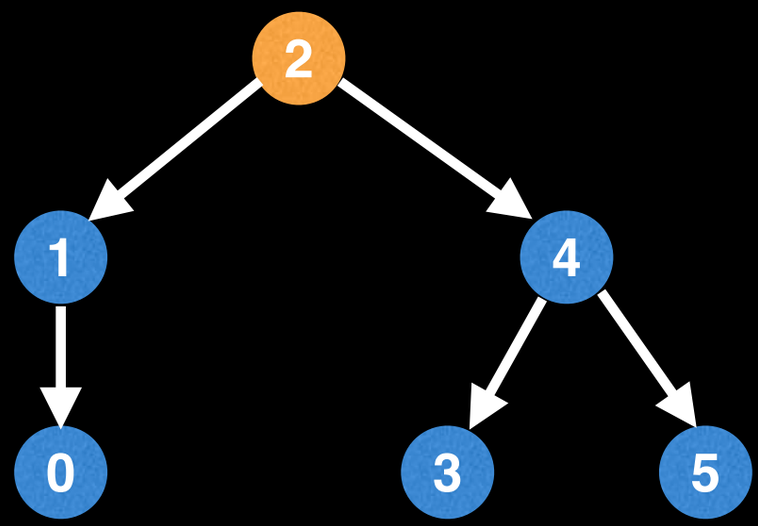


Find the center(s) of the original tree. We'll see how to handle the case where either tree can have more than 1 center shortly.

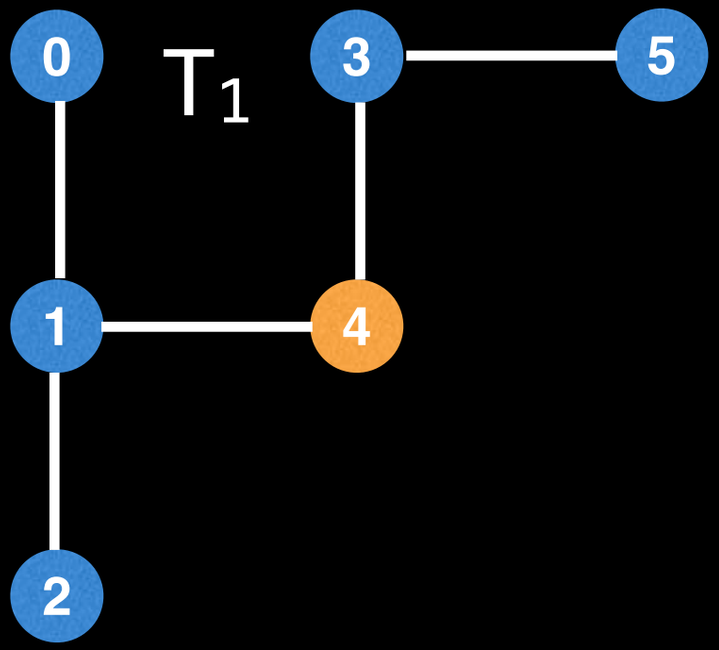


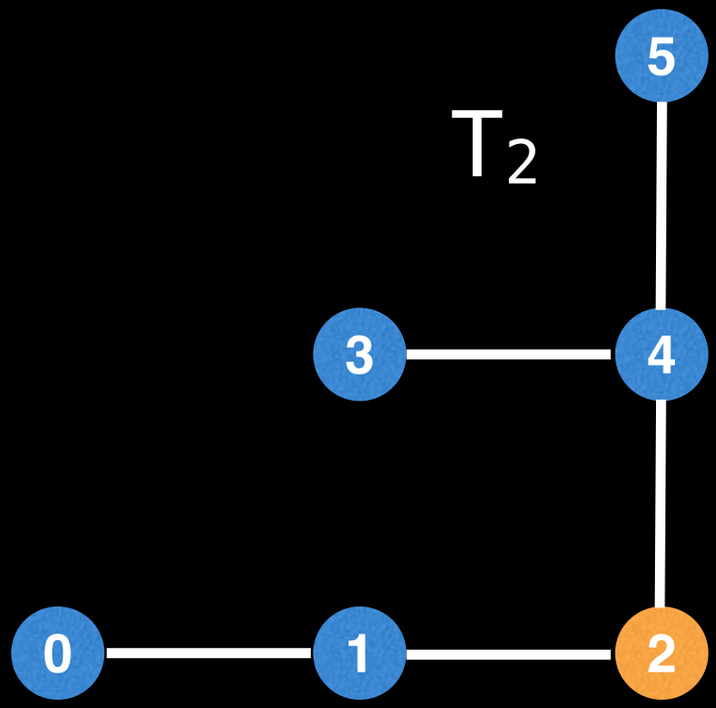


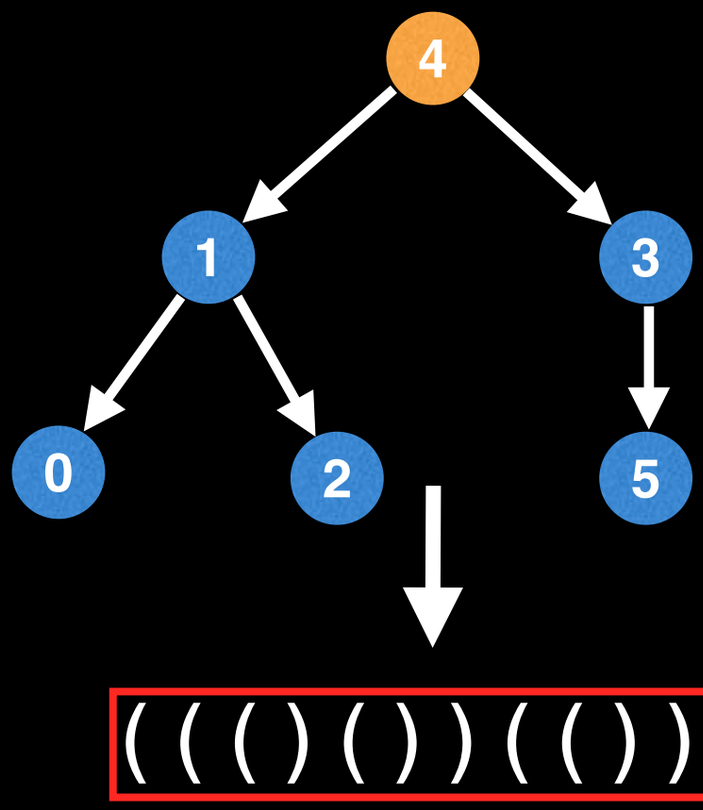


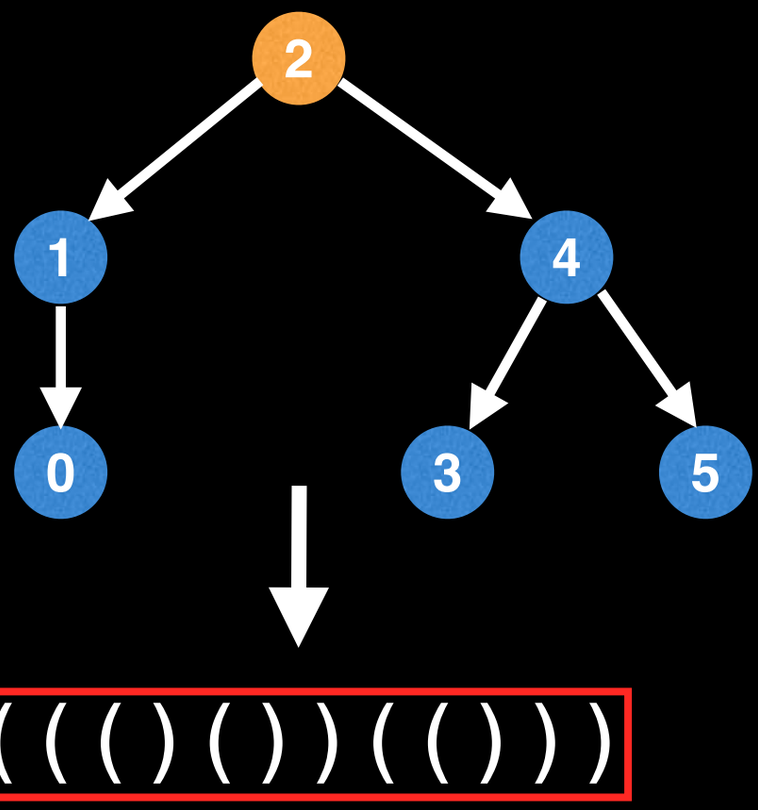


Root the tree at the center node.

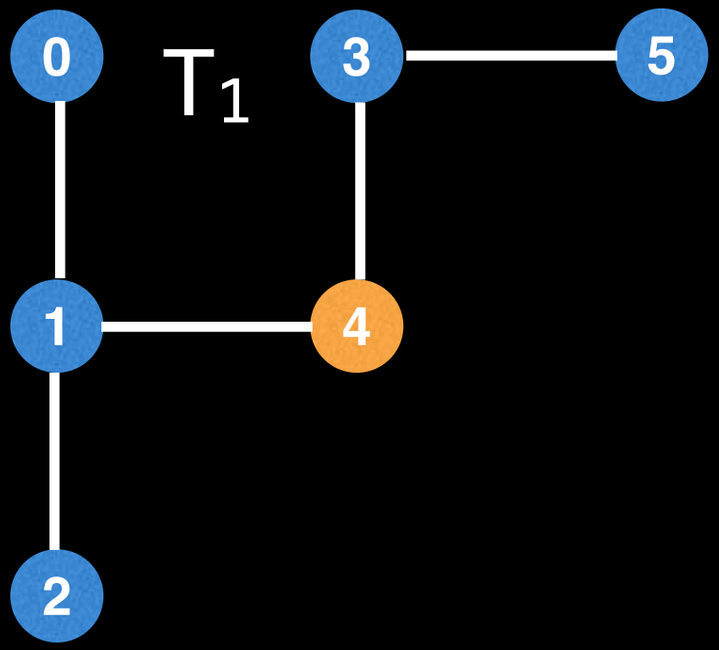


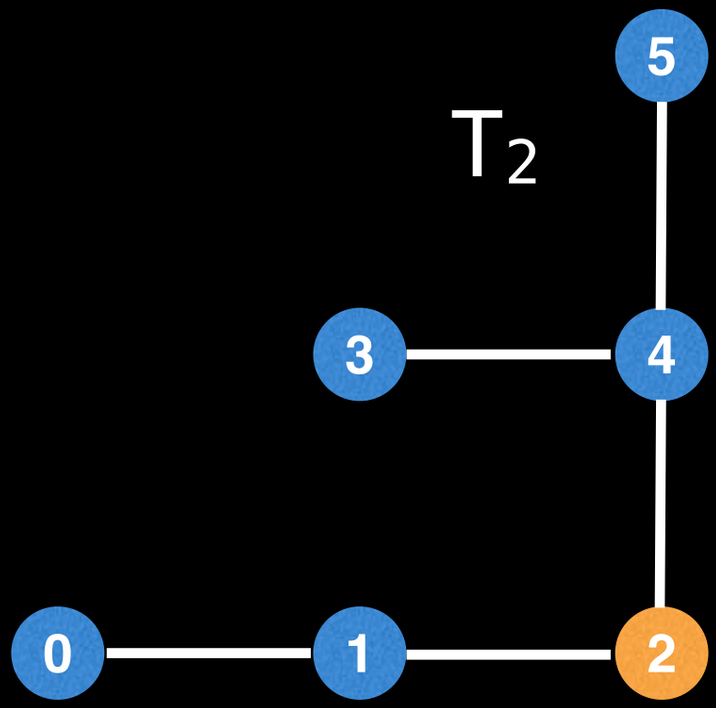


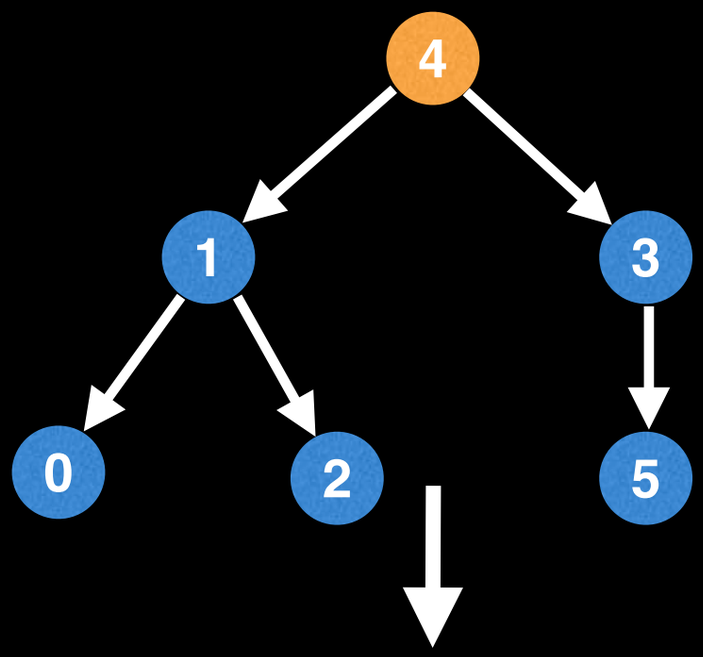


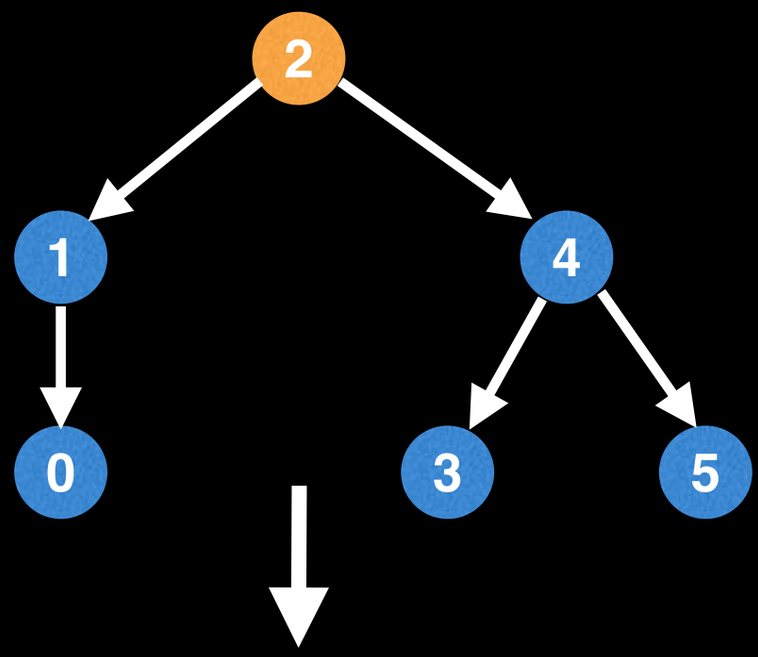


Generate the encoding for each tree and compare the serialized trees for equality.





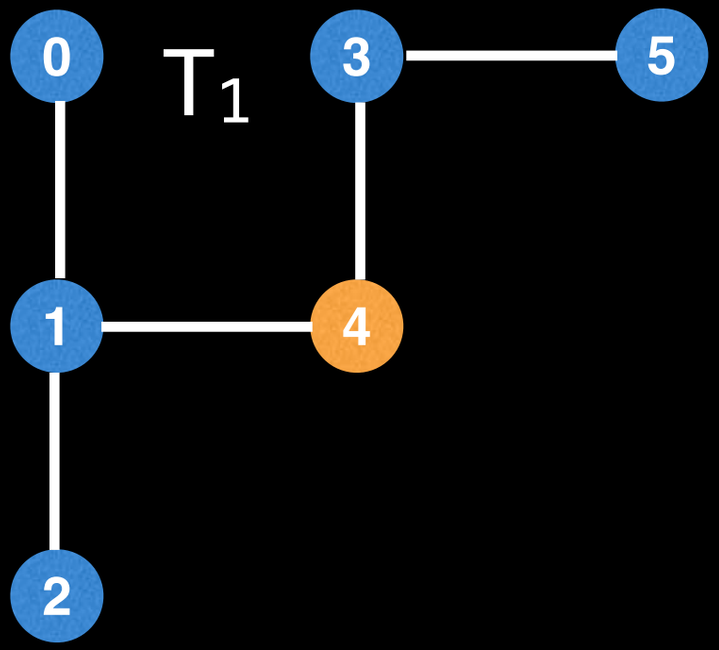


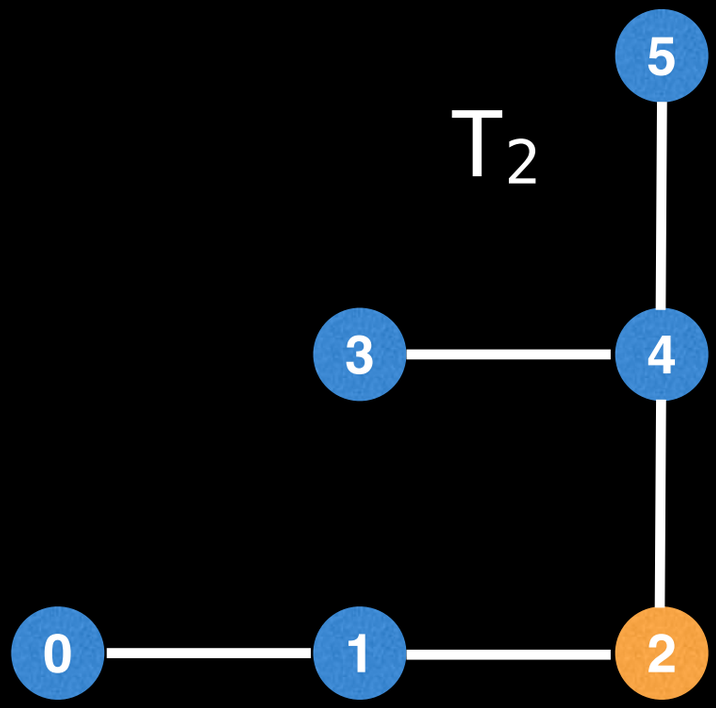


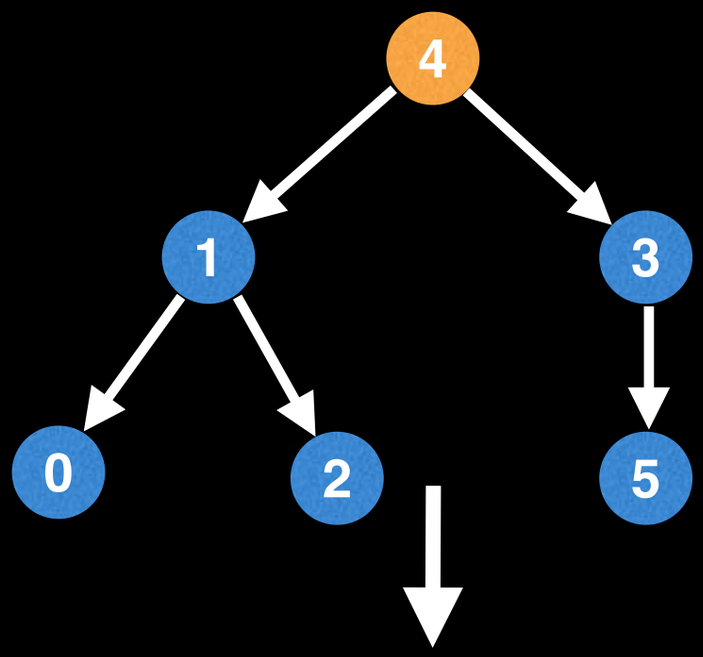
000101100111

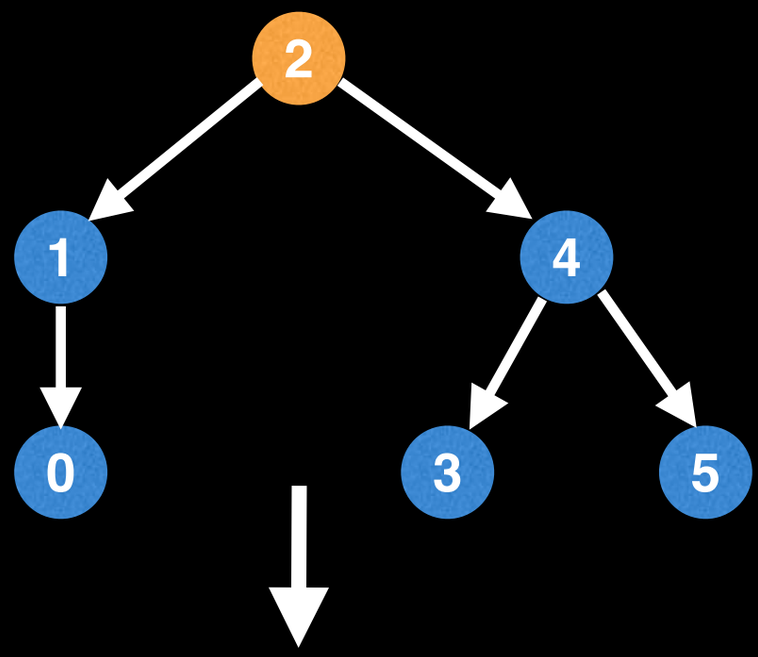
000101100111

The tree encoding is simply a sequence of left '(' and right ')' brackets. However, you can also think of them as 1’s and 0’s (i.e a large number) if you prefer.









((()())(()))

((()())(()))

It should also be possible to reconstruct the tree solely from the encoding. This is left as an exercise to the reader… 😛

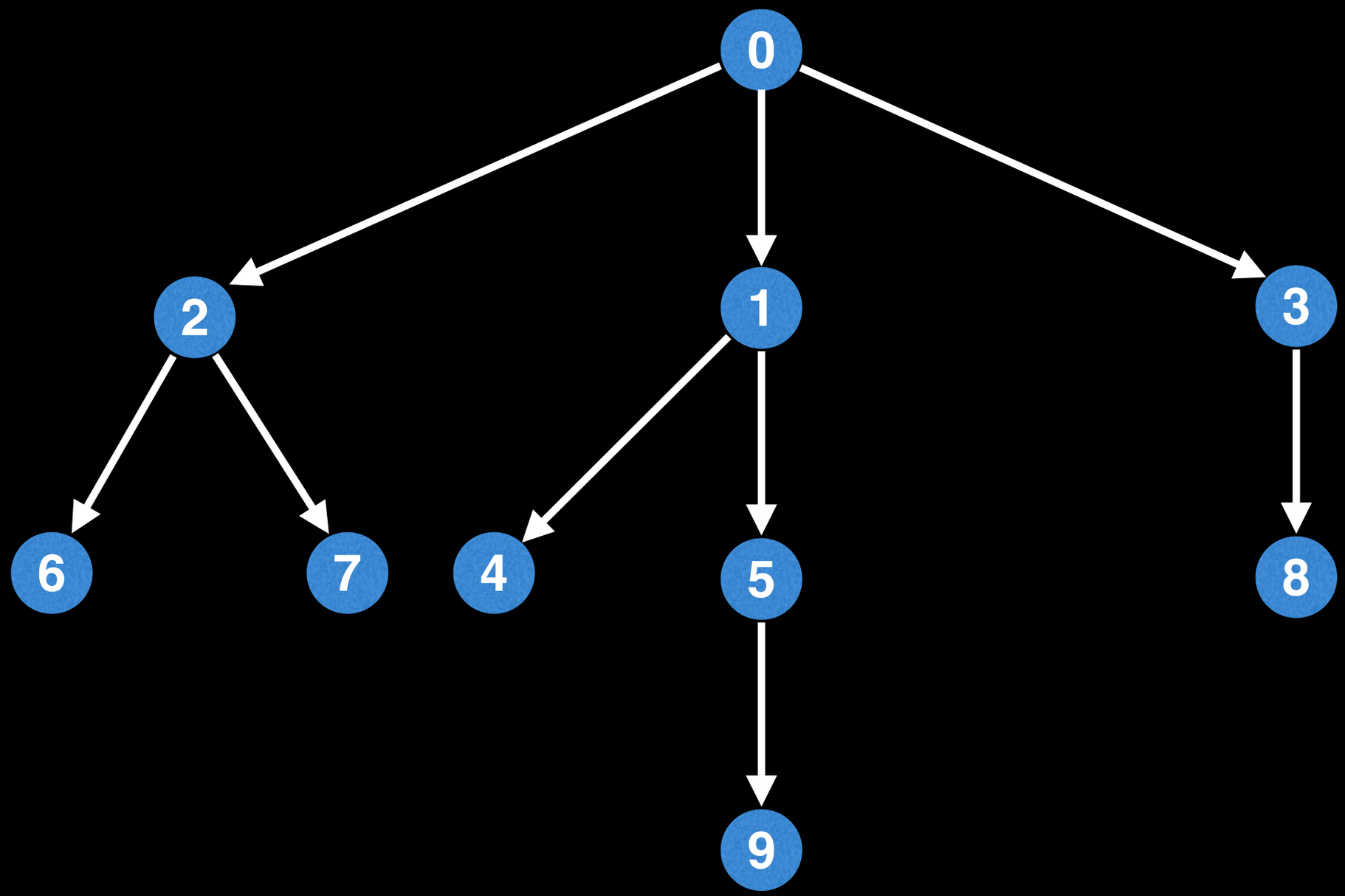
Generating the tree encoding

The AHU (Aho, Hopcroft, Ullman) algorithm is a clever serialization technique for representing a tree as a unique string.

Unlike many tree isomorphism invariants and heuristics, AHU is able to capture a **complete history** of a tree's **degree spectrum** and structure ensuring a deterministic method of checking for tree isomorphisms.

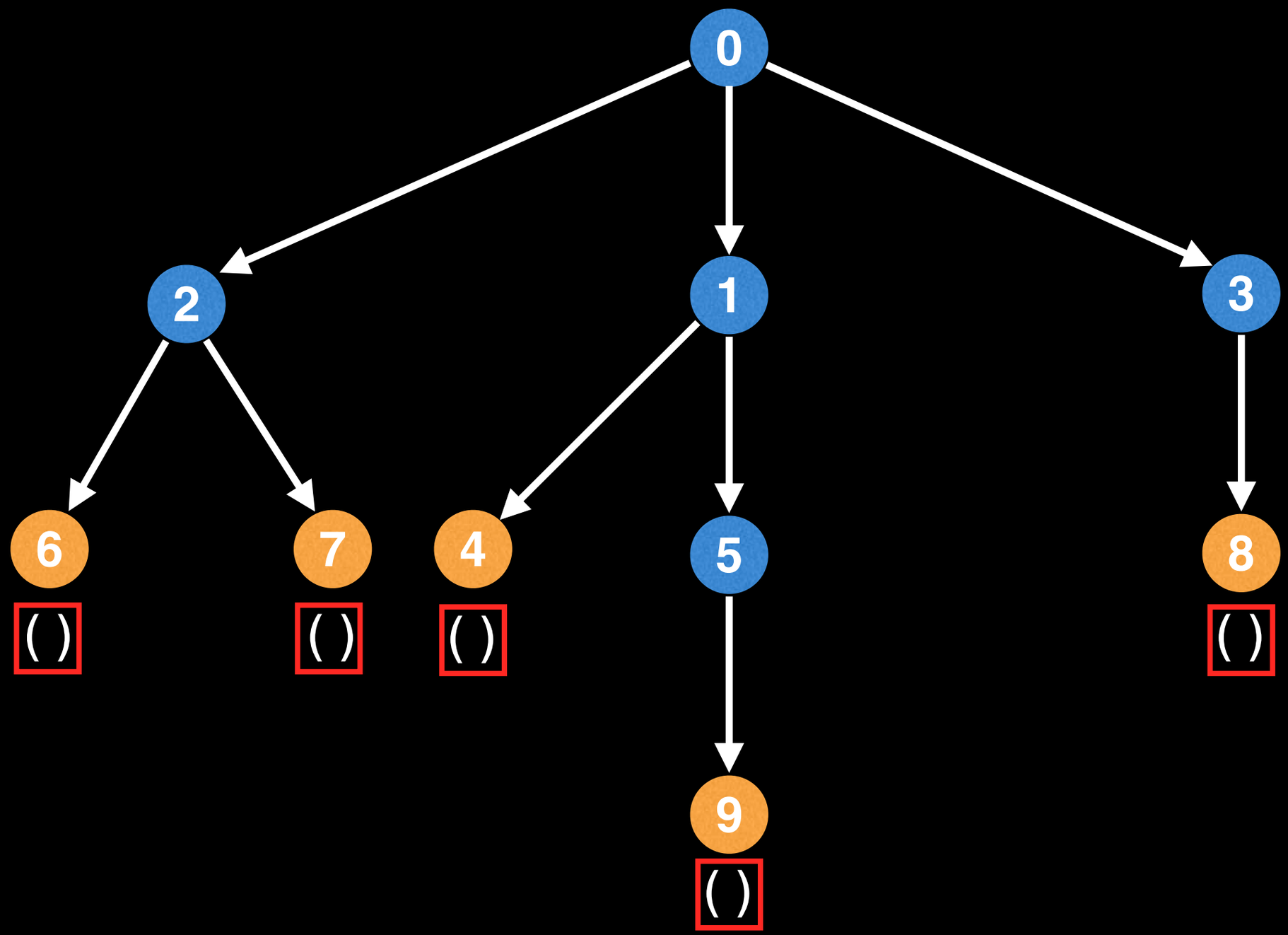
Let's have a closer look…

Tree Encoding



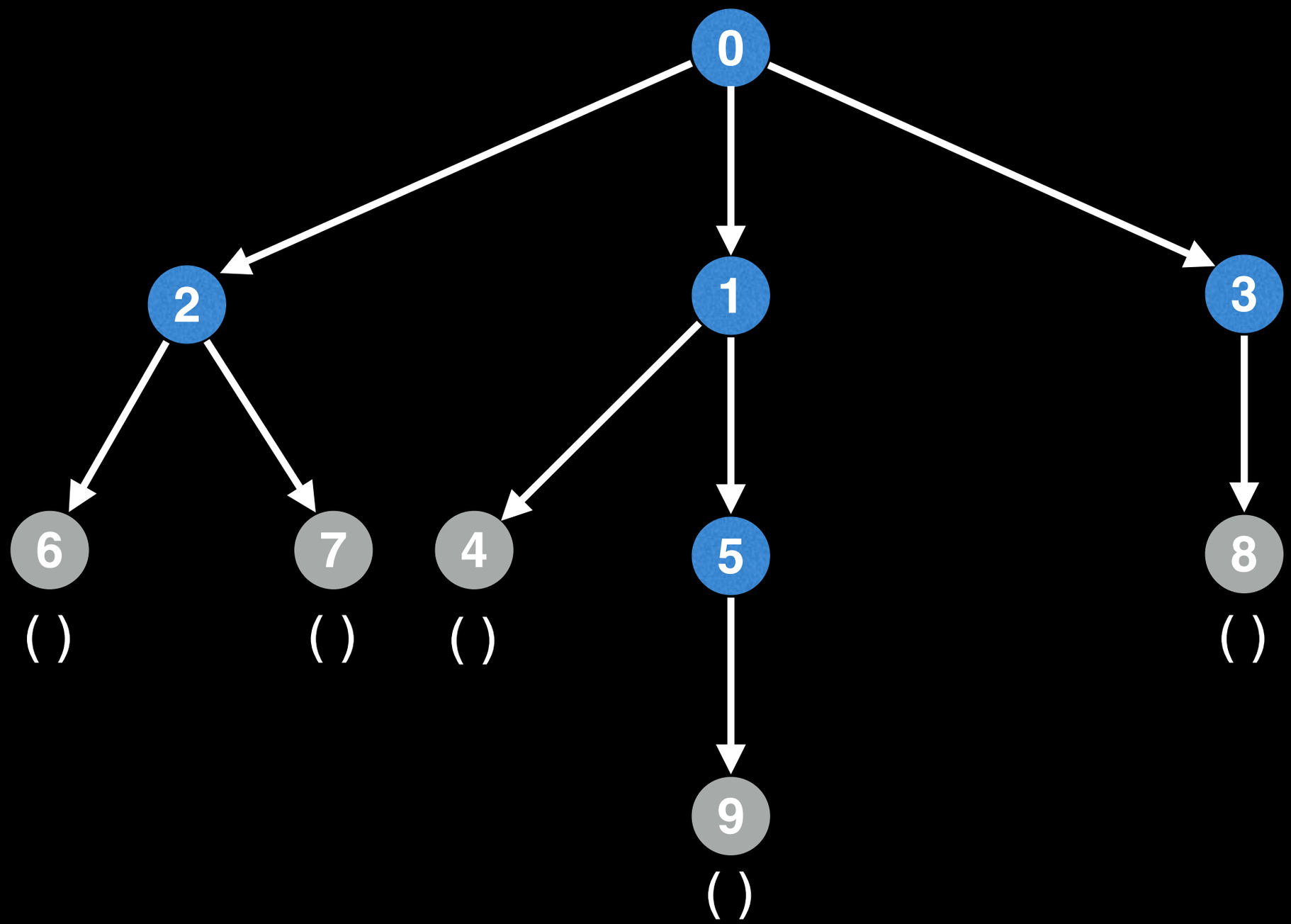
Tree Encoding

Start by assigning all leaf nodes
Knuth tuples: '()'



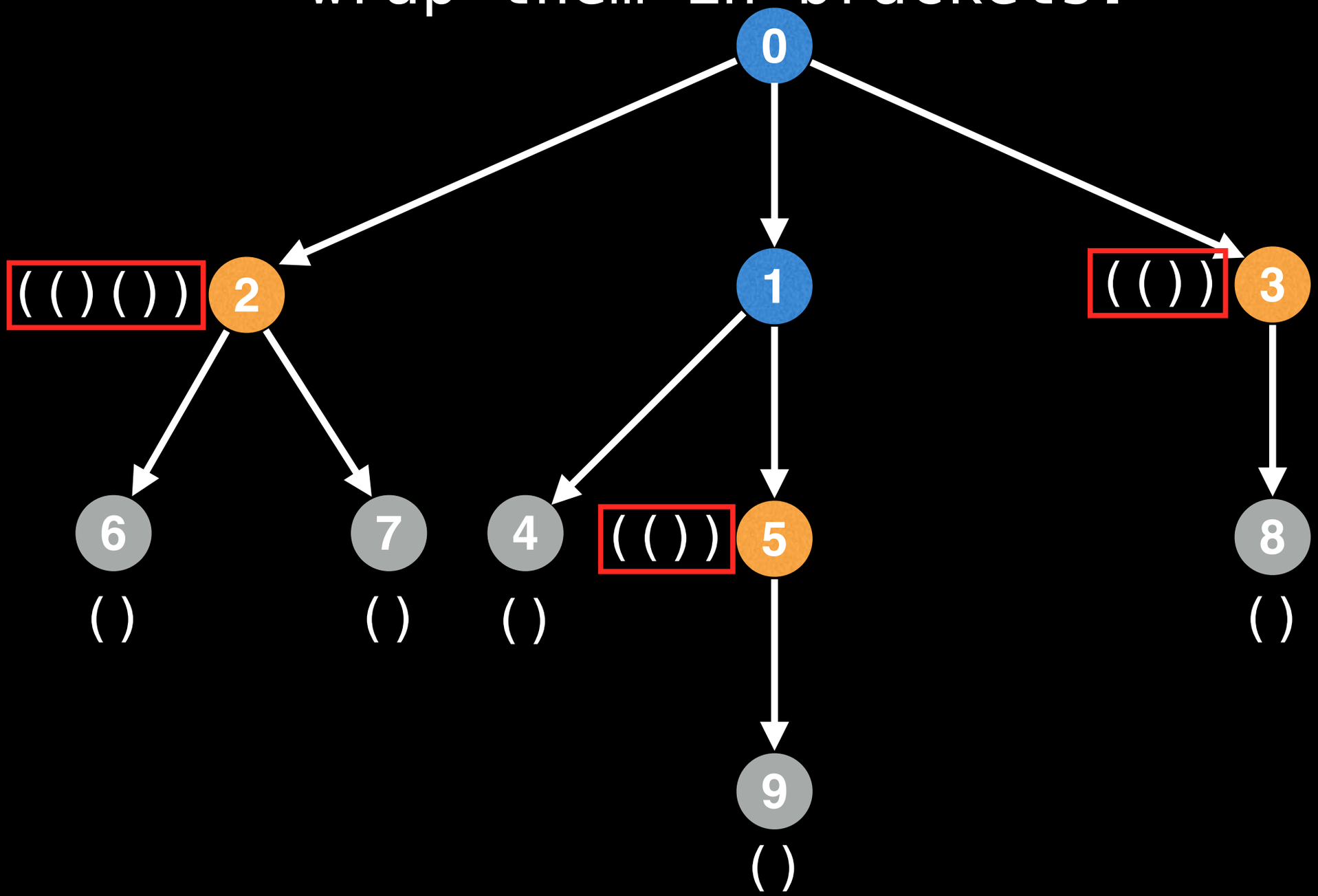
Tree Encoding

Start by assigning all leaf nodes
Knuth tuples: '()'



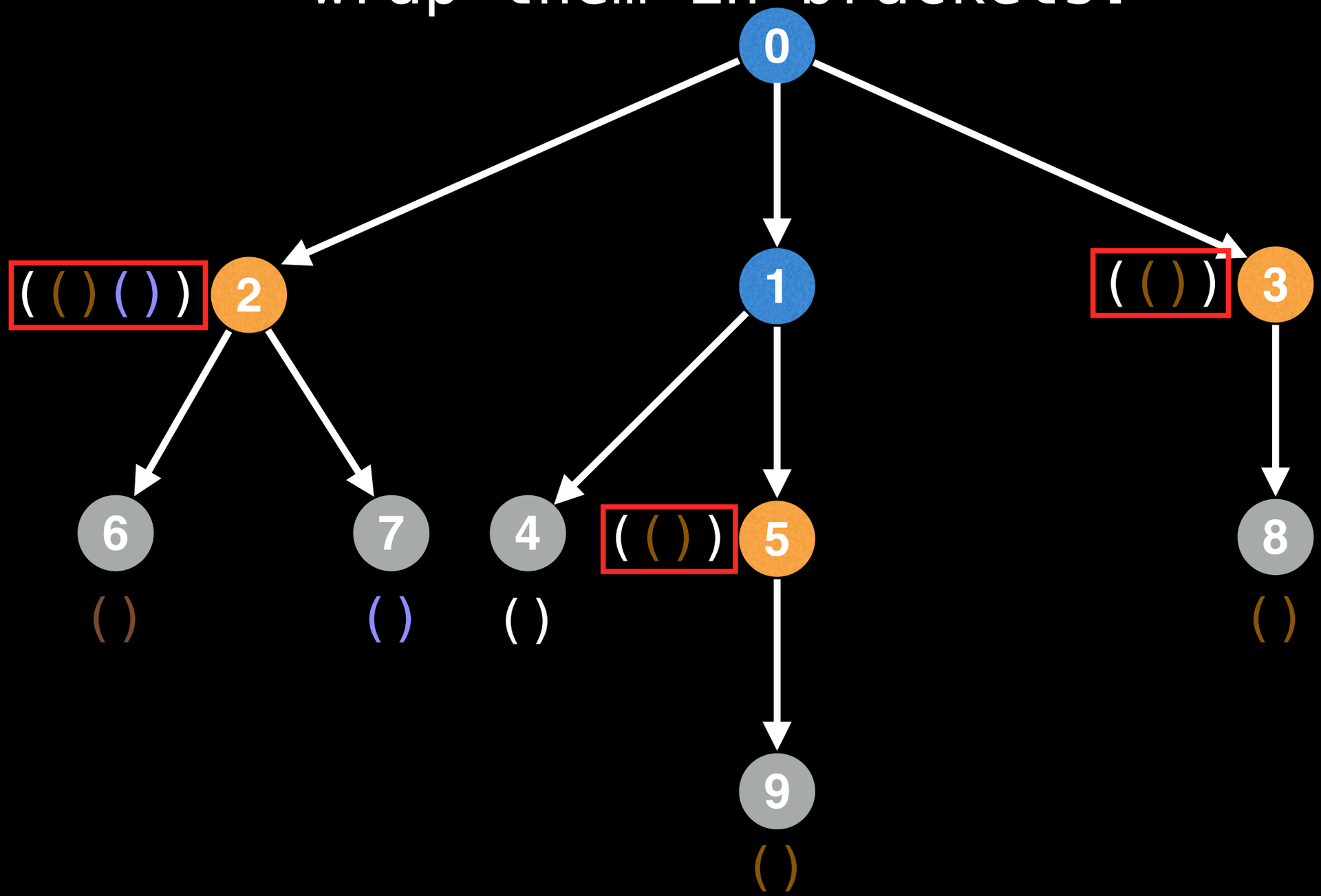
Tree Encoding

Process all nodes with grayed out children and combine the labels of their child nodes and wrap them in brackets.

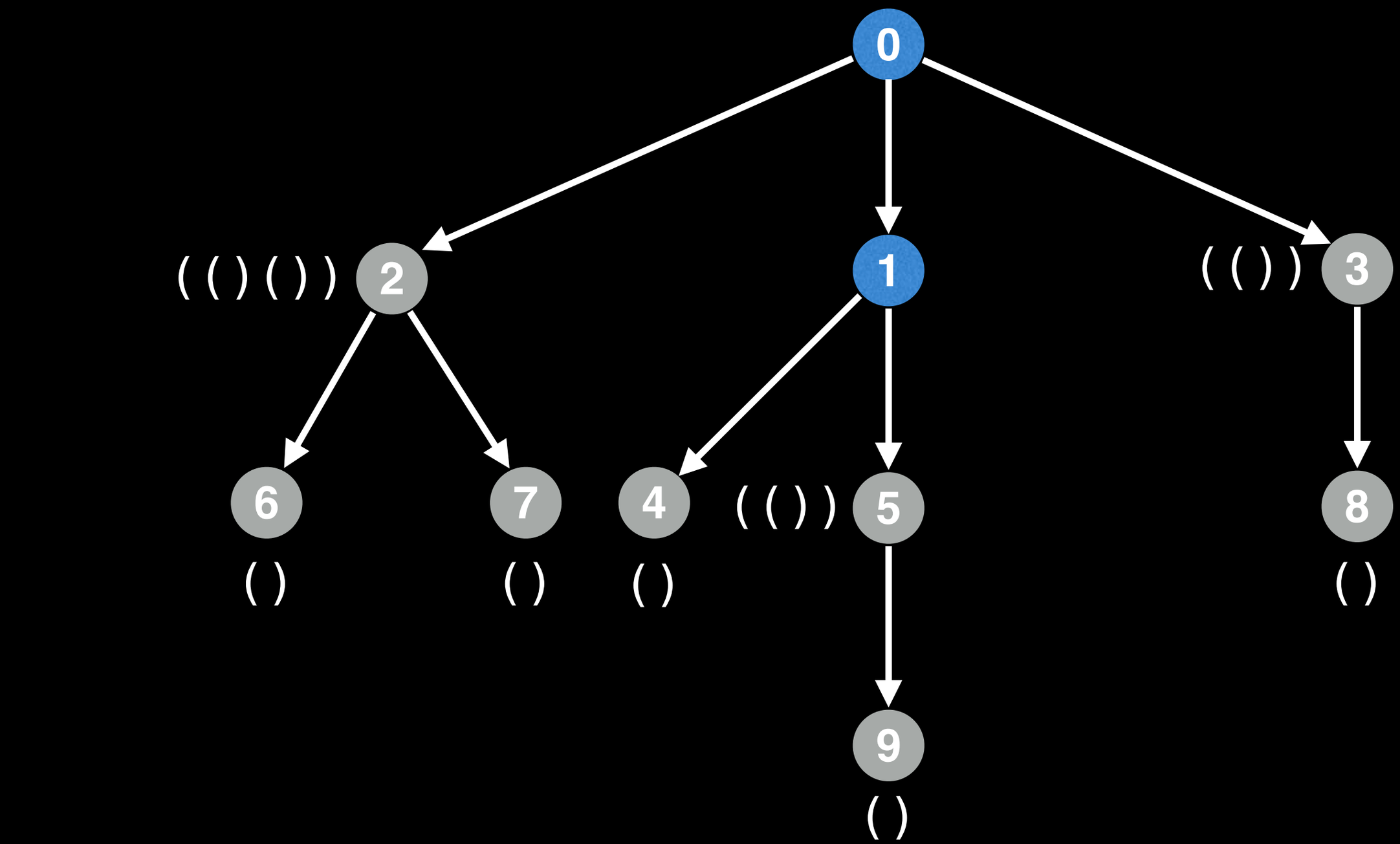


Tree Encoding

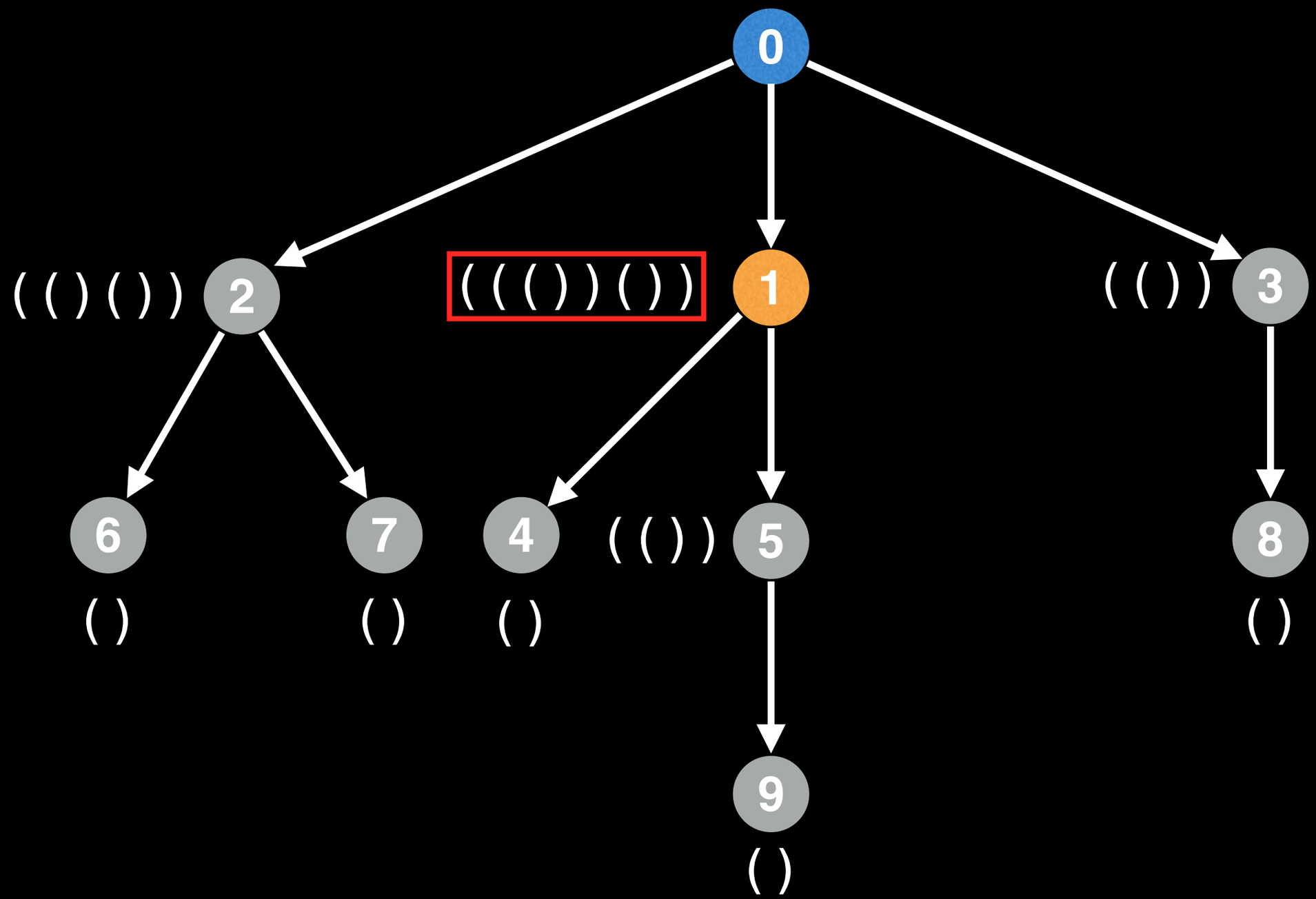
Process all nodes with grayed out children and combine the labels of their child nodes and wrap them in brackets.



Tree Encoding

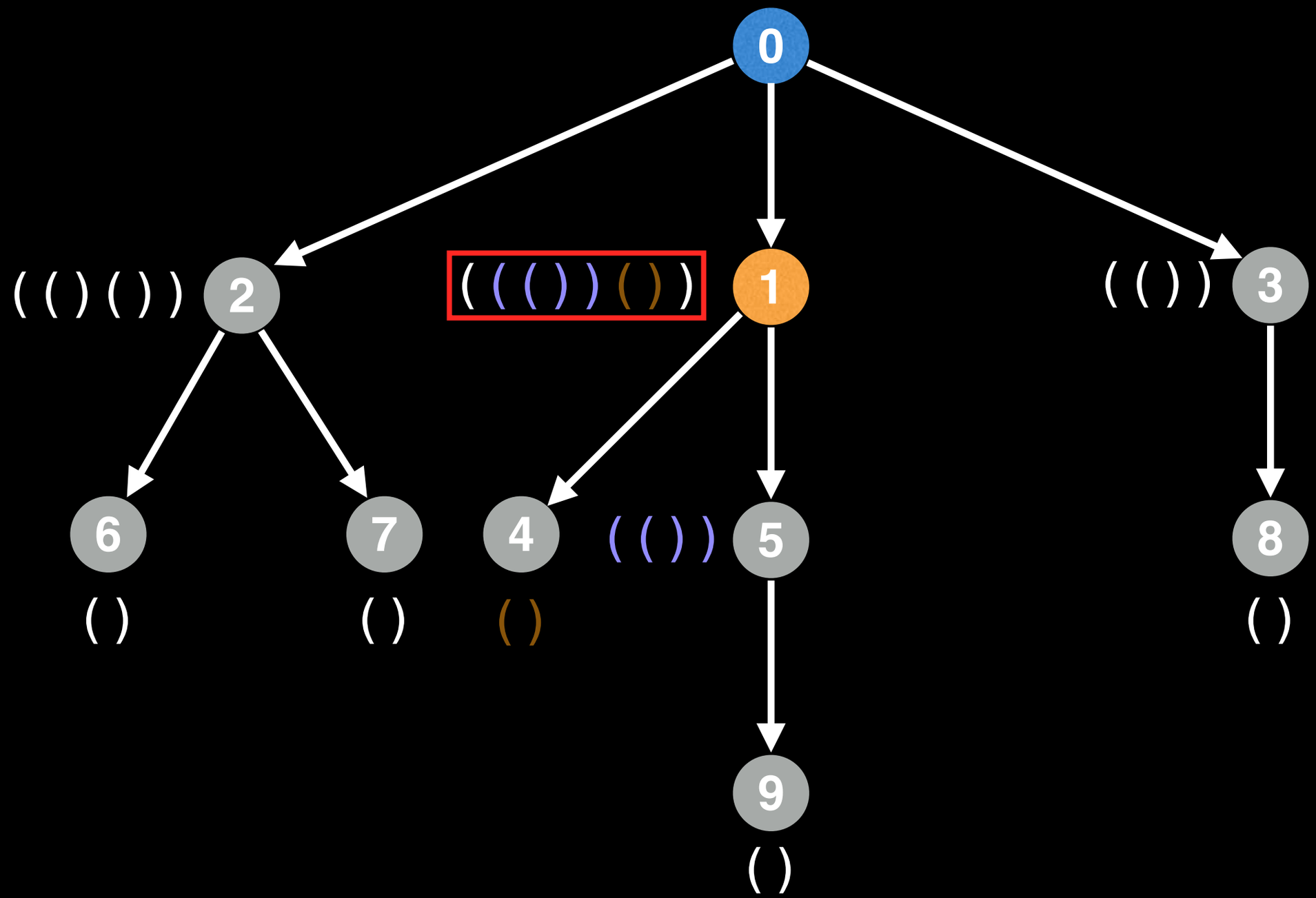


Tree Encoding



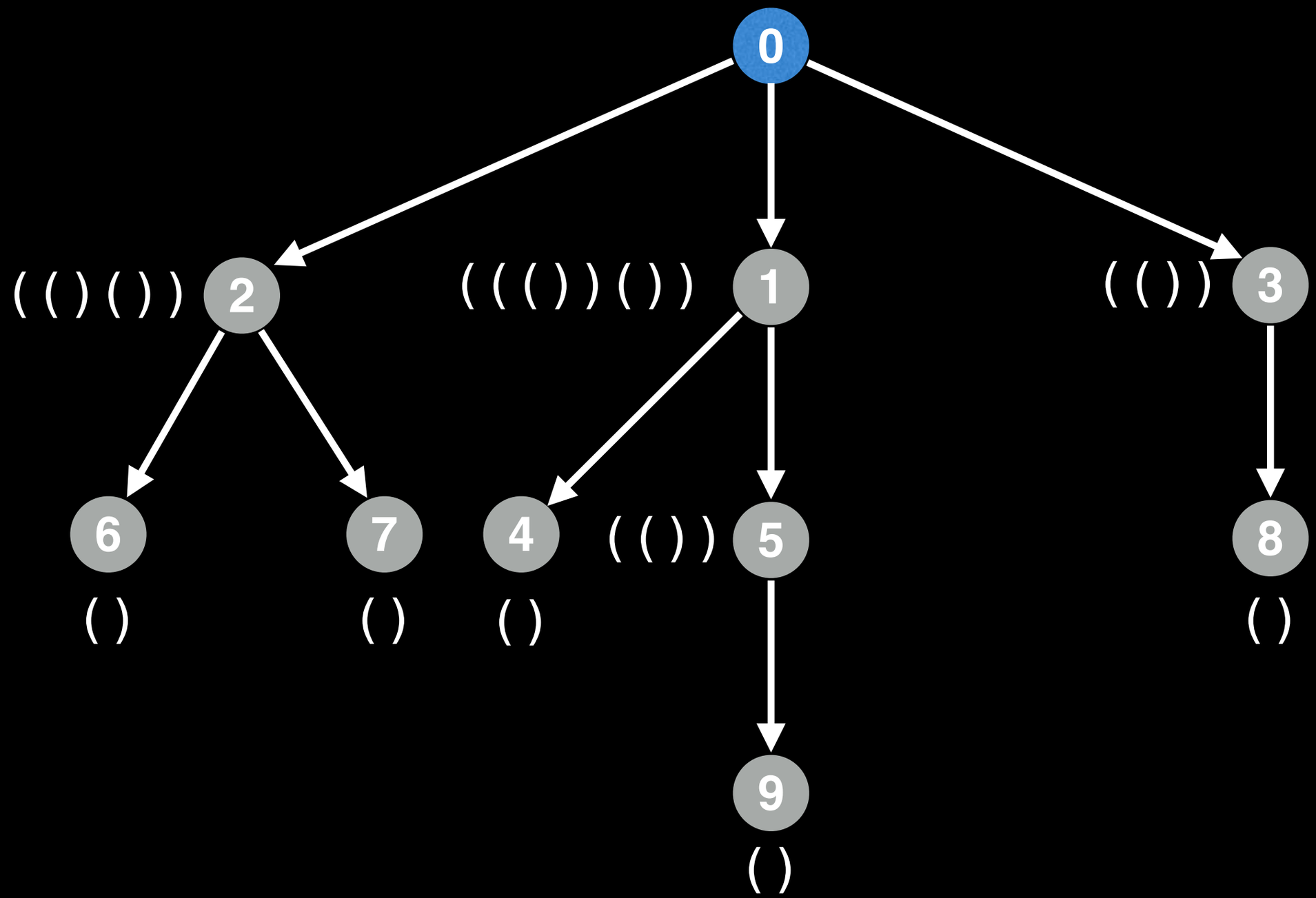
Tree Encoding

Notice that the labels get ***sorted*** when combined, this is important.

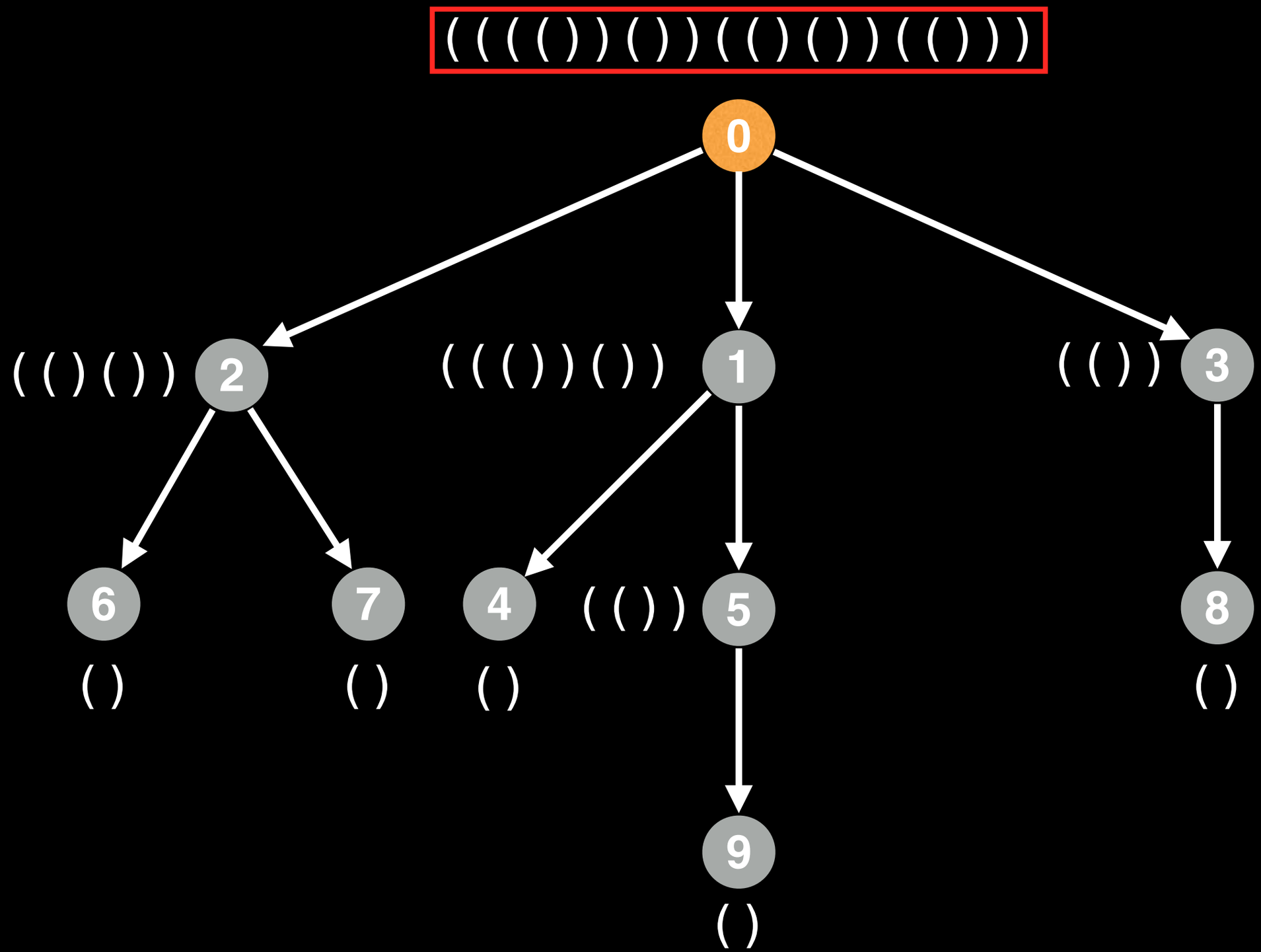


Tree Encoding

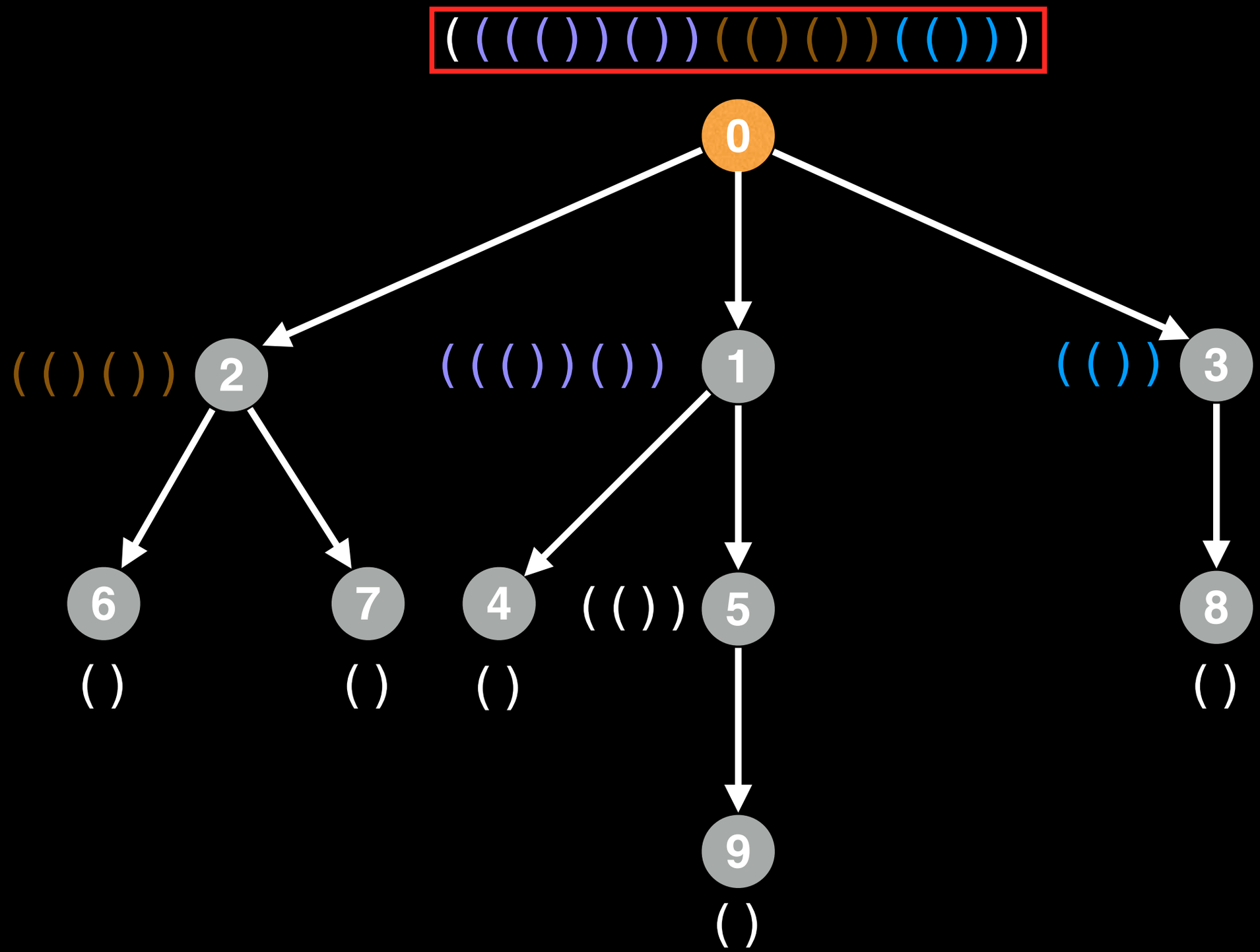
Notice that the labels get ***sorted*** when combined, this is important.



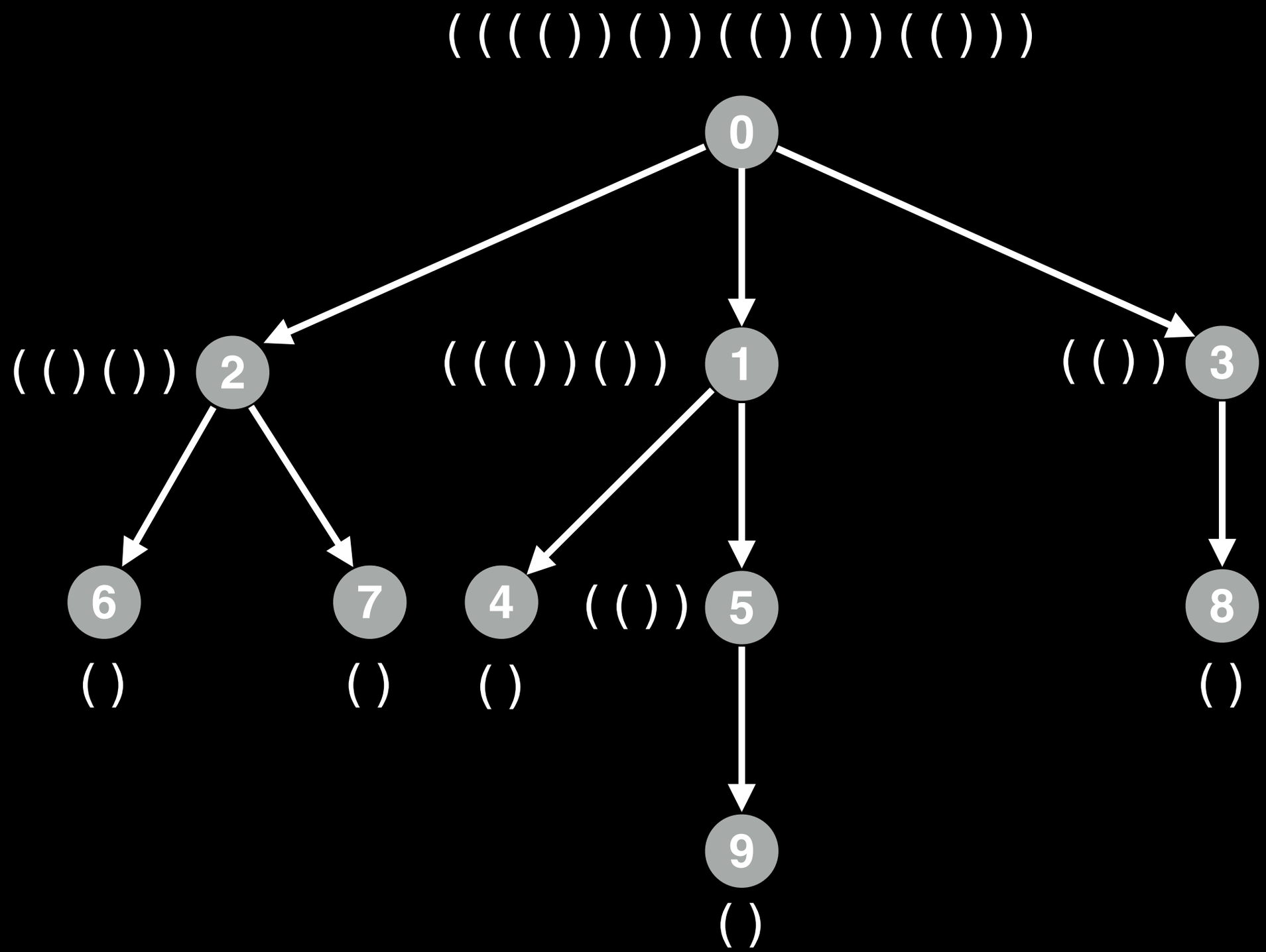
Tree Encoding



Tree Encoding



Tree Encoding



Tree Encoding Summary

In summary of what we did for AHU:

- Leaf nodes are assigned Knuth tuples '()' to begin with.
- Every time you move up a layer the labels of the previous subtrees get sorted lexicographically and wrapped in brackets.
- You cannot process a node until you have processed all its children.

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Unrooted tree encoding pseudocode

Rooted trees are stored recursively in TreeNode objects:

```
# 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;
```

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)
  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Unrooted tree encoding pseudocode

```
function encode(node):
  if node == null:
    return ""

  labels = []
  for child in node.children():
    labels.add(encode(child))

  # Regular lexicographic sort
  sort(labels)

  result = ""
  for label in labels:
    result += label

  return "(" + result + ")"
```

Unrooted tree encoding pseudocode

```
function encode(node):
  if node == null:
    return ""

  labels = []
  for child in node.children():
    labels.add(encode(child))

  # Regular lexicographic sort
  sort(labels)

  result = ""
  for label in labels:
    result += label

  return "(" + result + ")"
```

Unrooted tree encoding pseudocode

```
function encode(node):
  if node == null:
    return ""

  labels = []
  for child in node.children():
    labels.add(encode(child))

  # Regular lexicographic sort
  sort(labels)

  result = ""
  for label in labels:
    result += label

  return "(" + result + ")"
```

Unrooted tree encoding pseudocode

```
function encode(node):
  if node == null:
    return ""

  labels = []
  for child in node.children():
    labels.add(encode(child))

  # Regular lexicographic sort
  sort(labels)

  result = ""
  for label in labels:
    result += label

  return "(" + result + ")"
```

Unrooted tree encoding pseudocode

```
function encode(node):
  if node == null:
    return ""

  labels = []
  for child in node.children():
    labels.add(encode(child))

# Regular lexicographic sort
sort(labels)

result = ""
for label in labels:
  result += label

return "(" + result + ")"
```

Unrooted tree encoding pseudocode

```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

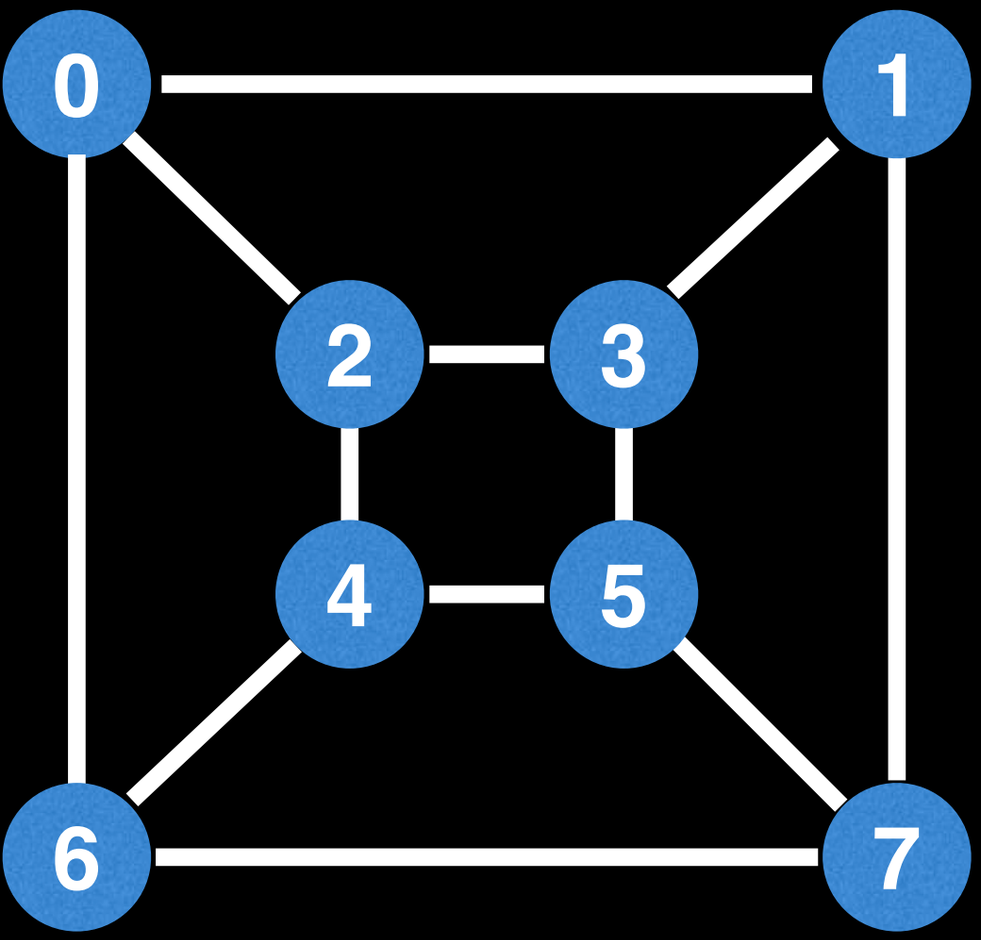
Unrooted tree encoding pseudocode

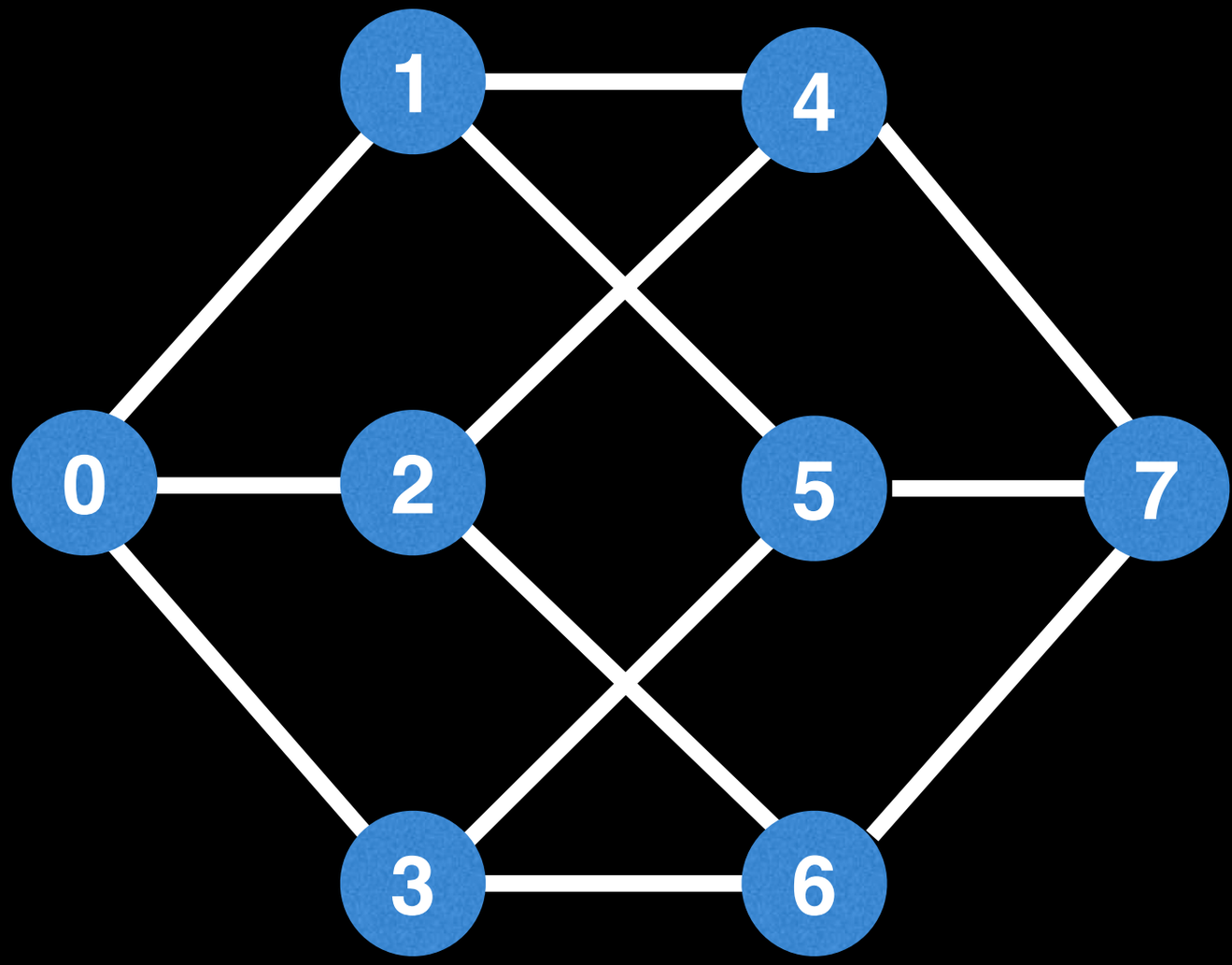
```
# Returns whether two trees are isomorphic.
# Parameters tree1 and tree2 are undirected trees
# stored as adjacency lists.
function treesAreIsomorphic(tree1, tree2):
  tree1_centers = treeCenters(tree1)
  tree2_centers = treeCenters(tree2)

  tree1_rooted = rootTree(tree1, tree1_centers[0])
  tree1_encoded = encode(tree1_rooted)

  for center in tree2_centers:
    tree2_rooted = rootTree(tree2, center)
    tree2_encoded = encode(tree2_rooted)
    # Two trees are isomorphic if their encoded
    # canonical forms are equal.
    if tree1_encoded == tree2_encoded:
      return True
  return False
```

Isomorphic Trees





Determining if two graphs are isomorphic is not only not obvious to the human eye, but also a difficult problem for computers.