```
class TreeNode:
  def __init__(self, val, name=None):
    self.name = name
    self.val = val
    self.children = []

  def __str__(self):
    return self.name if self.name else str(self.val)

def minimax(node, depth, maximizing_player=True):
  if depth == 0 or not node.children:
    return node.val

  if maximizing_player:
    value = float('-inf')
    for child in node.children:
      value = max(value, minimax(child, depth - 1, False))
      node.val = value
    return value
  else:
    value = float('inf')
    for child in node.children:
      value = min(value, minimax(child, depth - 1, True))
      node.val = value
    return value

def printNodeValues(node):
  print(node.name + ": ", node.val);
  for child in node.children:
    printNodeValues(child);


root = TreeNode(0, 'A')
b = TreeNode(10, 'B')
c = TreeNode(0, 'C')
d = TreeNode(0, 'D')
e = TreeNode(-10, 'E')
f = TreeNode(0, 'F')
g = TreeNode(-10, 'G')
h = TreeNode(0, 'H')
i = TreeNode(10, 'I')
j = TreeNode(10, 'J')

root.children = [b, c, d]
c.children = [e, f]
f.children = [i]
d.children = [g, h]
h.children = [j]

optimal_path = [root.name]
optimal_val = [root.val]
```

```
for child in root.children:
  if minimax(child, depth=3, maximizing_player=True) == minimax(root, depth=3,
maximizing_player=True):
    optimal_path.append(child)
    optimal_val.append(child.val)
    break
print("Node values ")
printNodeValues(root)

print("Optimal path:", [str(node) for node in optimal_path])
print("Optimal value:", root.val  + minimax(root, depth=3, maximizing_player=True))
```