# Min max

**PROGRAM:**

class TreeNode:

def \_\_init\_\_(self, label=None):

self.label = label

self.children = []

def build\_tree\_from\_input():

print("Enter the tree nodes in a parent-children format (use '.' for no children):")

root\_label = input("Enter label for root node: ")

root = TreeNode(root\_label)

queue = [root]

while queue:

current\_node = queue.pop(0)

children\_input = input(f"Enter children for {current\_node.label} (comma-separated labels or '.' if no children): ").strip()

if children\_input == '.':

continue

child\_labels = list(map(str.strip, children\_input.split(',')))

for label in child\_labels:

if label == '.':

continue

new\_child = TreeNode(label)

current\_node.children.append(new\_child)

queue.append(new\_child)

return root

def minimax(node, maximizingPlayer):

if not node.children:

return int(node.label) # Convert the label to an integer if necessary

if maximizingPlayer:

best = float('-inf')

for child in node.children:

val = minimax(child, False)

best = max(best, val)

return best

else:

best = float('inf')

for child in node.children:

val = minimax(child, True)

best = min(best, val)

return best

if \_\_name\_\_ == "\_\_main\_\_":

root = build\_tree\_from\_input()

if root is None:

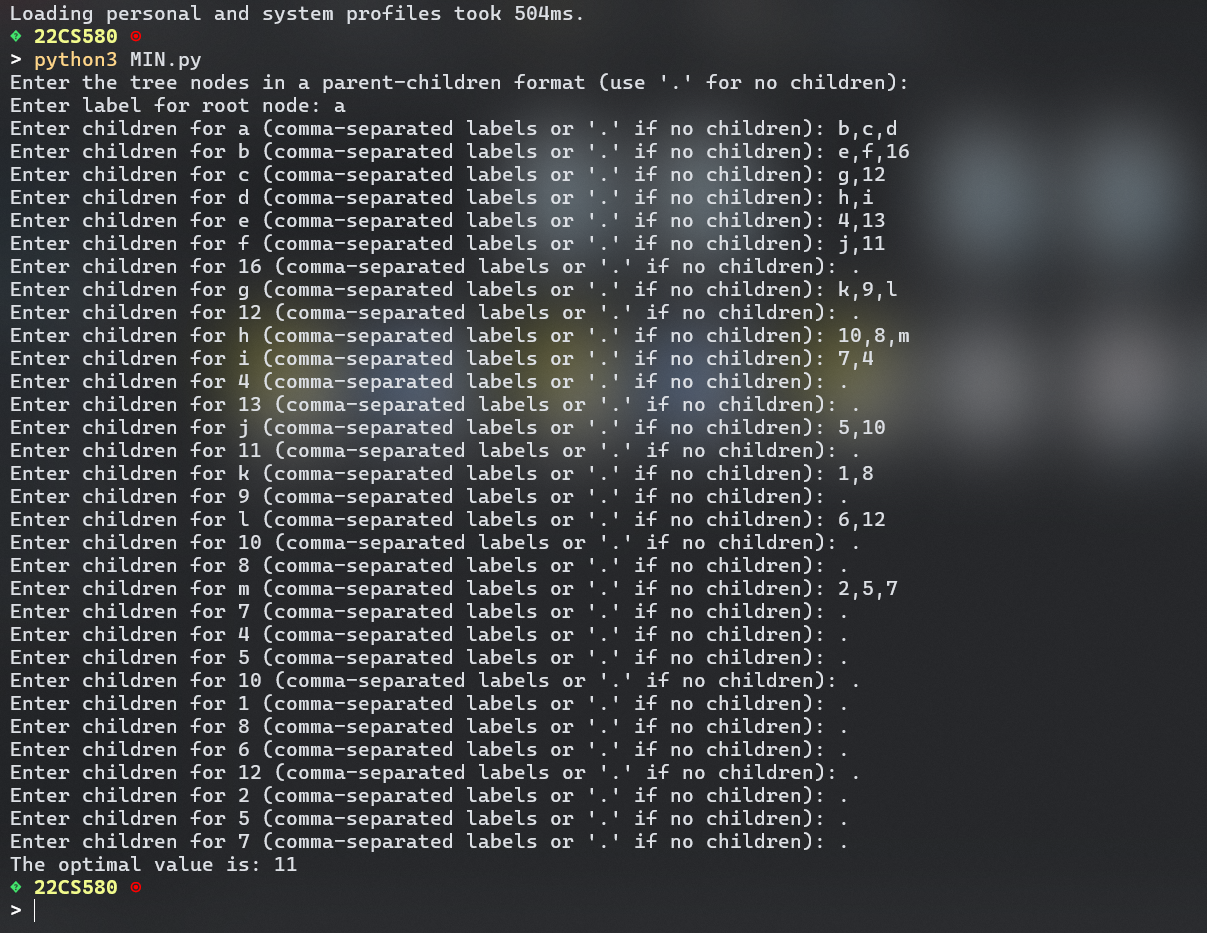
print("Empty tree!")

else:

optimalValue = minimax(root, True)

print("The optimal value is:", optimalValue)

**OUTPUT :**



**ALPHA BETA PRUNING**

**PROGRAM:**

class TreeNode:

def \_\_init\_\_(self, label=None):

self.label = label

self.children = []

def build\_tree\_from\_input():

print("Enter the tree nodes in a parent-children format (use '.' for no children):")

root\_label = input("Enter label for root node: ")

root = TreeNode(root\_label)

queue = [root]

while queue:

current\_node = queue.pop(0)

children\_input = input(f"Enter children for {current\_node.label} (comma-separated labels or '.' if no children): ").strip()

if children\_input == '.':

continue

child\_labels = list(map(str.strip, children\_input.split(',')))

for label in child\_labels:

if label == '.':

continue

new\_child = TreeNode(label)

current\_node.children.append(new\_child)

queue.append(new\_child)

return root

def minimax\_ab(node, maximizingPlayer, alpha, beta):

if not node.children:

return int(node.label) # Convert the label to an integer if necessary

if maximizingPlayer:

best = float('-inf')

for child in node.children:

val = minimax\_ab(child, False, alpha, beta)

best = max(best, val)

alpha = max(alpha, best)

if beta <= alpha:

break

return best

else:

best = float('inf')

for child in node.children:

val = minimax\_ab(child, True, alpha, beta)

best = min(best, val)

beta = min(beta, best)

if beta <= alpha:

break

return best

if \_\_name\_\_ == "\_\_main\_\_":

root = build\_tree\_from\_input()

if root is None:

print("Empty tree!")

else:

optimalValue = minimax\_ab(root, True, float('-inf'), float('inf'))

print("The optimal value is:", optimalValue)

**OUTPUT :**

