# Department of Computing

# CS370: Artificial Intelligence

# Submitted By: M. Hasnain Naeem Class: BSCS-7B Reg #: 212728

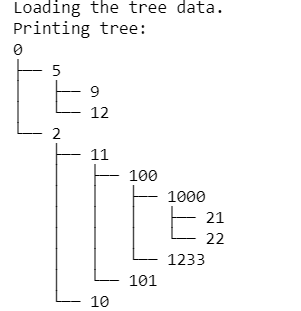
# Lab 07: Minimax with Alpha Beta Pruning

# Date: 11-3-2020

# Time: 10:00-1:00 & 2:00-5:00

# Instructor: Dr. Imran Malik

## Screenshots

****

****

****

****

## Code (Converted from Jupyter Notebook)

**Library used to print the tree:** anytree

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # # Lab 7  # \_\_Submitted By: M. Hasnain Naeem (212728) from BSCS-7B\_\_  # In[36]:  # imports  import math  from anytree import Node, RenderTree, find\_by\_attr, LevelOrderIter  # ## Loading the Tree  # In[135]:  print("Loading the tree data.")  print("Printing tree:")  # load tree from the file  with open('tree\_data.txt', 'r') as f:  # split the parent and child from the line  lines = f.readlines()[1:]  root = Node(lines[0].split(" ")[0])  nodes = {}  nodes[root.name] = root    # iterate the lines and get the tree  for line in lines:  line = line.split(" ")  name = "".join(line[1:]).strip()  nodes[name] = Node(name, parent=nodes[line[0]])  for pre, \_, node in RenderTree(root):  print("%s%s" % (pre, node.name))  # In[124]:  # get the level order list for the mini-max and alpha beta pruning algorithm  node\_list = [float(node.name) for node in LevelOrderIter(root)]  print("Tree node list using level order:")  print(node\_list)  # ## Task 1 - MiniMax  # In[125]:  # minimax algorithm  def minimax (curr\_depth, node\_index, max\_turn, node\_list, target\_depth):    # base case : target depth is reached  if curr\_depth == target\_depth:  return node\_list[node\_index]  if max\_turn:  return max(  minimax(curr\_depth + 1, node\_index \* 2, False, node\_list, target\_depth),  minimax(curr\_depth + 1, node\_index \* 2 + 1, False, node\_list, target\_depth)  )  else:  return min(  minimax(curr\_depth + 1, node\_index \* 2, True, node\_list, target\_depth),  minimax(curr\_depth + 1, node\_index \* 2 + 1, True, node\_list, target\_depth)  )  # In[136]:  # get depth  tree\_depth = math.log(len(node\_list), 2)  # run the algorithm  optimal\_val = minimax(0, 0, True, node\_list, tree\_depth)  print("Optimal Value using Minimax: " + str(optimal\_val))  # ## Task 2 - Alpha-Beta Pruning  # In[133]:  # Alpha Beta Prunning Algorithm  MAX, MIN = math.inf, -math.inf  def alpha\_beta\_prunning(depth, node\_index, max\_turn, node\_list, alpha, beta):    """  Returns optimal value using alpha beta prunning  """    # base case  if depth == 3:  return node\_list[node\_index]    # recursion case  if max\_turn:    max\_value = MIN  # check left and right children  for i in range(0, 2):    cur\_val = float(alpha\_beta\_prunning(depth + 1, node\_index \* 2 + i, False, node\_list, alpha, beta))  max\_value = max(max\_value, cur\_val)  alpha = max(alpha, max\_value)    # Check pruning condition  if beta <= alpha:  break    return max\_value    else:    min\_value = MAX  # check for left andright children  for i in range(0, 2):  cur\_val = float(alpha\_beta\_prunning(depth + 1, node\_index \* 2 + i, True, node\_list, alpha, beta))  min\_value = min(min\_value, cur\_val)  beta = min(beta, min\_value)    # Check pruning condition  if beta <= alpha:  break    return min\_value  # In[137]:  optimal\_val\_abp = alpha\_beta\_prunning(0, 0, True, node\_list, MIN, MAX)  print("Optimal Value using Alpha Beta Pruning: " + str(optimal\_val\_abp))  # In[ ]: |