// Java program to demonstrate working of Alpha-Beta Pruning

import java.io.\*;

class GFG {

// Initial values of

// Alpha and Beta

static int MAX = 1000;

static int MIN = -1000;

// Returns optimal value for

// current player (Initially called

// for root and maximizer)

static int minimax(int depth, int nodeIndex,

                   Boolean maximizingPlayer,

                   int values[], int alpha,

                   int beta)

{

    // Terminating condition. i.e

    // leaf node is reached

    if (depth == 3)

        return values[nodeIndex];

    if (maximizingPlayer)

    {

        int best = MIN;

        // Recur for left and

        // right children

        for (int i = 0; i < 2; i++)

        {

            int val = minimax(depth + 1, nodeIndex \* 2 + i,

                              false, values, alpha, beta);

            best = Math.max(best, val);

            alpha = Math.max(alpha, best);

            // Alpha Beta Pruning

            if (beta <= alpha)

                break;

        }

        return best;

    }

    else

    {

        int best = MAX;

        // Recur for left and

        // right children

        for (int i = 0; i < 2; i++)

        {

            int val = minimax(depth + 1, nodeIndex \* 2 + i,

                              true, values, alpha, beta);

            best = Math.min(best, val);

            beta = Math.min(beta, best);

            // Alpha Beta Pruning

            if (beta <= alpha)

                break;

        }

        return best;

    }

}

    // Driver Code

    public static void main (String[] args)

    {

        int values[] = {3, 5, 6, 9, 1, 2, 0, -1};

        System.out.println("The optimal value is : " +

                            minimax(0, 0, true, values, MIN, MAX));

    }

}

**Implementation:** Python3

# Python3 program to demonstrate

# working of Alpha-Beta Pruning

# Initial values of Alpha and Beta

MAX, MIN = 1000, -1000

# Returns optimal value for current player

#(Initially called for root and maximizer)

def minimax(depth, nodeIndex, maximizingPlayer,

            values, alpha, beta):

    # Terminating condition. i.e

    # leaf node is reached

    if depth == 3:

        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right children

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                          False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

    else:

        best = MAX

        # Recur for left and

        # right children

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                            True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

# Driver Code

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

    values = [3, 5, 6, 9, 1, 2, 0, -1]

    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))