Experiment 5

Foundations of AI

Alpha beta pruning

By- Shivansh Jain,VIT Chennai

AIM:-

To find the optimal scores from the scores of leaf nodes using alpha beta pruning

Concept:-

Alpha-Beta pruning is an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.
Alpha is the best value that the **maximizer** currently can guarantee at that level or above.
Beta is the best value that the **minimizer** currently can guarantee at that level or above.

Algorithm:-

1. Enter an array of scores find its length and thus its height
2. Check the terminating statement else continue
3. If its max player turn assign the current MIN(beta) as best and find max between max attainable from the children nodes and the current best
4. Assign alpha as the max from best and current alpha
5. Vice versa if max player turn is false
6. Give the inputs values and call the function
7. Display the optimum score

Step by step implementation:-

```
#R version 4.1.0

#RStudio version 1.4.1717


rm(list = ls())

#to ensure a clean environment before executing the code


alphabeta <- function(depth, nodeIndex, maximizingPlayer,values, alpha, beta,targetDepth){

#function alphabeta containing 7 arguments

#depth beng the current depth of the node which changes ecery recursion
```

```
#nodeIndex being the index of node in array of nodes
#maximizingPlayer a boolean value which showcases if its max players turn or not
#values an array storing all the scores
#targetDepth being the maximum depth code will run till
#alpha beta represent the max and min values during the algorithm

 #Terminating condition
 #if we the target depth return the score of current node
if (depth == targetDepth){
  return(values[nodeIndex])}

 #if its maximizing players turn
 #return the max attainable value of two children of current node
if(maximizingPlayer){

  best = MIN

  #recur left and right child
  for(i in 0:1){

    val = alphabeta(depth + 1, (nodeIndex * 2) - i,FALSE, values, alpha, beta,targetDepth)
    best = max(best, val)
    alpha = max(alpha, best)

    #pruning
    if(beta <= alpha){
      break
    }
  }
    return(best)

}
```

```
  #minimizing turn
  #return the min attainable value of two children of current node
  else{
    best = MAX

    #recur left and right child
    for (i in 0:1){

      val = alphabeta(depth + 1, (nodeIndex * 2) - i,TRUE, values, alpha, beta,targetDepth)
      best = min(best, val)
      beta = min(beta, best)

      #pruning
      if(beta <= alpha){
        break
      }
    }
      return(best)

  }
}

#driver code

#initial values of alpha and beta
MIN=-1000
MAX=1000

test_case1 = c(3, 5, 6, 9, 1, 2, 0, -1)
len1=length(test_case1)
treeDepth1 = ceiling(log(len1, base=2))
alphabeta(0, 1, TRUE, test_case1,MIN,MAX, treeDepth1)
```

test_case2 = c(2,3,5,9,0,0,7,4,2,1,5,6)

len2=length(test_case2)

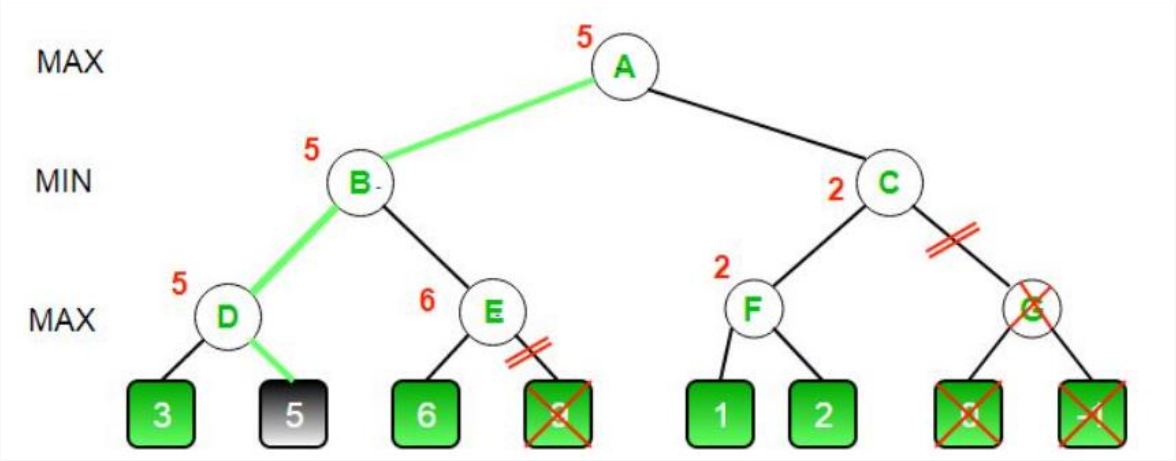
treeDepth2 = ceiling(log(len2, base=2))

alphabeta(0, 1, TRUE, test_case2,MIN,MAX, treeDepth1)

RESULTS AND OUTPUT:-

→

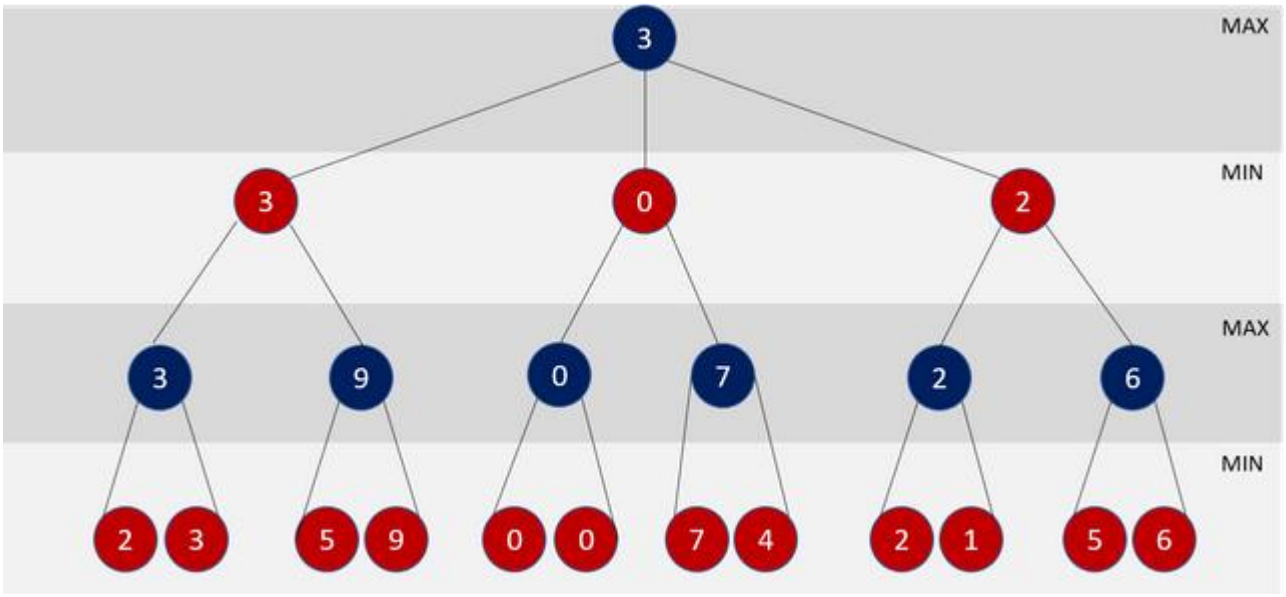
Test case 1:-



Output:-

```
> test_case1 = c(3, 5, 6, 9, 1, 2, 0, -1)
> len1=length(test_case1)
> treeDepth1 = ceiling(log(len1, base=2))
> alphabeta(0, 1, TRUE, test_case1,MIN,MAX, treeDepth1)
[1] 5
>
```

Test case 2:-



Output

```
> test_case2 = c(2,3,5,9,0,0,7,4,2,1,5,6)
> len2=length(test_case2)
> treeDepth2 = ceiling(log(len2, base=2))
> alphabeta(0, 1, TRUE, test_case2,MIN,MAX, treeDepth1)
[1] 3
>
```