Experiment 5

Objective

To be able to apply alpha-beta pruning algorithm for game-playing.

Problem Statement

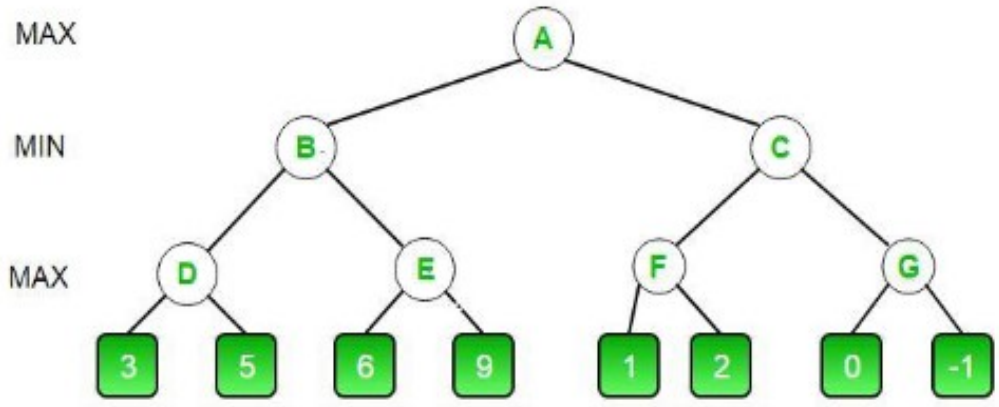
To apply an adversarial search technique (alpha-beta pruning) for finding out the minimum cost path from a source node to a goal node in a graph.

Lab Exercise

Write the program for finding out the path from a source node to a goal node in a graph using the alpha-beta pruning algorithm in a game tree. Ask the user to provide the details of the tree, from the user i.e. number of nodes, number of edges, utility values at the edge nodes, etc.

Program Code

Example:




```
import sys

class node:
    def __init__(self,name,utility,children=[]):
        self.name = name
        self.utility = utility
        self.children = children

    def terminal(self):
        return not self.children



def alphabeta(node,maxi,alpha,beta,visited=[]):
    if node.terminal():
        return node.utility,visited

    if maxi:
        best_val = -sys.maxsize
        for child in node.children:
            val,list1 = alphabeta(child,False,alpha,beta,visited)
            best_val = max(best_val,val)
            alpha = max(alpha , best_val)
            visited.append(child.name)

            if beta<=alpha:
                break
        return best_val,visited
    else:
```

```
        best_val = sys.maxsize
        for child in node.children:
            val,list1 = alphabeta(child,True,alpha,beta,visited)
            best_val = min(best_val,val)
            beta = min(beta , best_val)
            visited.append(child.name)

            if beta<=alpha:
                break
        return best_val,visited

if __name__=='__main__':
# h = node('H',3)
# i = node('I',5)
# j = node('J',6)
# k = node('K',9)
# l = node('L',1)
# m = node('M',2)
# n = node('N',0)
# o = node('O',-1)

# d = node('D',0,[h,i])
# g = node('G',0,[n,o])
# f = node('F',0,[l,m])
# e = node('E',0,[j,k])
# d = node('D',0,[h,i])
# c = node('C',0,[f,g])
# b = node('B',0,[d,e])
# a = node('A',0,[b,c])

# result,val = alphabeta(a, True, mini, maxi,visited=[])
# print(result)
# print(len(val))

    h =[]
    t=int(input("Enter total no of node: "))
    l=int(input("Enter total no of terminal node: "))
    p=t-l

    for i in range(l):
        tem1 = input('Enter Terminal Node Name: ')
        tem2 = int(input('Enter Value: '))
        h.insert(i,node(tem1,tem2))

    for w in range(0,l):
            print(w,h[w].name)

    print(" %%%% Terminal nodes finished %%%% ")

    for i in range(l,t):
        tem1 = input('Enter New Node Name : ')
        tem2 = int(input('Enter Value : '))
        ncn=int(input("Enter no of child nodes : "))

        for w in range(0,i):
            print(w,h[w].name)
        listt=[]

        for tob in range(ncn):
            t = int(input("Enter "+str(tob)+ " th child number from list: "))
            listt.append(h[t])

        h.insert(i,node(tem1,tem2,listt))

    for w in range(0,(t+2)):
            print(w,h[w].name)
    k = int(input("Enter index of node you want to search: "))
    mm=bool(int(input("Do yo want Max search: ")))

    mini = -sys.maxsize
    maxi = sys.maxsize
    result,val = alphabeta(h[k],mm,mini,maxi,visited=[])
    print("Root Node Value : ")
    print(result)
    print("Total Visited Nodes : ")
    print(val,len(val))
```

Output

```
PS C:\Users\ASUS> & C:/Users/ASUS/anaconda3/python.exe d:/SEM-5/AI/alphabeta.py
Enter total no of node: 15
Enter total no of terminal node: 8
Enter Terminal Node Name: h
Enter Value: 3
Enter Terminal Node Name: i
Enter Value: 5
Enter Terminal Node Name: j
Enter Value: 6
Enter Terminal Node Name: k
Enter Value: 9
Enter Terminal Node Name: l
Enter Value: 1
Enter Terminal Node Name: m
Enter Value: 2
Enter Terminal Node Name: n
Enter Value: 0
Enter Terminal Node Name: o
Enter Value: -1
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
 %%%% Terminal nodes finished %%%%
Enter New Node Name : d
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
Enter 0 th child number from list: 0
Enter 1 th child number from list: 1
Enter New Node Name : e
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
Enter 0 th child number from list: 2
Enter 1 th child number from list: 3
Enter New Node Name : f
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
Enter 0 th child number from list: 4
Enter 1 th child number from list: 5
Enter New Node Name : g
Enter Value : 0
Enter no of child nodes : 2
0 h
```

```
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
10 f
Enter 0 th child number from list: 6
Enter 1 th child number from list: 7
Enter New Node Name : b
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
10 f
11 g
Enter 0 th child number from list: 8
Enter 1 th child number from list: 9
Enter New Node Name : c
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
10 f
11 g
12 b
Enter 0 th child number from list: 10
Enter 1 th child number from list: 11
Enter New Node Name : a
Enter Value : 0
Enter no of child nodes : 2
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
10 f
11 g
12 b
13 c
Enter 0 th child number from list: 12
Enter 1 th child number from list: 13
0 h
1 i
2 j
3 k
4 l
5 m
6 n
7 o
8 d
9 e
```

```
10 f
11 g
12 b
13 c
14 a
Enter index of node you want to search: 14
Do yo want Max search: 1
Root Node Value :
5
Total Visited Nodes :
['h', 'i', 'd', 'j', 'e', 'b', 'l', 'm', 'f', 'c'] 10
PS C:\Users\ASUS>
```

Conclusion

The alpha-beta algorithm is an optimization technique widely used in game tree search algorithms to reduce the number of nodes that need to be explored. The algorithm prunes the search tree branches that may not produce a better solution than the current best solution. Experimental analysis of the alpha-beta algorithm shows that it can significantly reduce the number of nodes explored in the game tree.**In this experiment total number of nodes are 15 and visited nodes other than root node is only 10, So we visited 11 nodes out of 15. This reduces seek time and improves performance.**The effectiveness of the algorithm depends on the structure of the search tree and the score function used to estimate the game state values.It is important to note that algorithms have limitations and do not always provide optimal solutions. Algorithm performance can be affected by factors such as search tree depth and tree branching factors.