Name : Ashutosh Ardu

Regno : 20BRS1262

MiniMax Algorithm

Question

Two player game problem

Rama and Samuel are good friends, one day both have played a two-person game P. Assume the game tree of game P is a full binary tree. A full binary tree is also known as a 2-tree in which every node other than the leaf nodes has two child nodes. The game P is a zero-sum game, the total payoff is zero. That is, One wins(+1), the other loses(-1). The game P is also a complete information game, both the payers have access to all the information. That is, both can see the board and thus know the options the other player has. The game P is an alternative move game, the players take turns to make their moves. Assume Samuel has started the game. Design an algorithm to compute the best moves for each player takes turn to make their moves. Implement your algorithm in any programming language.

Hint : To **represent** a **binary tree** using **array** first we need to convert a binary tree into a full binary tree. and then we give the number to each node and store it into their respective locations. If depth of the complete binary tree is n then its array size is 2^(n+1)-1.

Input format

Enter the depth ‘n’ of the full binary tree

Enter the values (+1, -1, 0) for the leaf nodes

Output format

Display the best move for each state.

Display the best moves for each player.

Code:

```
#include "iostream"
#include "cmath"
using namespace std;

struct bucket{
    int data[30];
    int n;
};

int miniMax(int *ar,bucket arr[],int index,int height,bool isMax){
    if(height == 0)
        return ar[index];

    int x,y,cur;
    if(isMax){
        x = miniMax(ar,arr,index*2,height-1,false);
        y = miniMax(ar,arr,index*2+1,height-1,false);
        cur = max(x,y);
        arr[height].data[arr[height].n] = x;
        arr[height].n+=1;
        arr[height].data[arr[height].n] = y;
        arr[height].n+=1;
        cout<<"Current Height : "<<height<<"\n";
        cout<<"Options : "<<x<<" , "<<y<<"\n";
        cout<<"Current Choice : "<<cur<<" by Samuel"<<"\n";
        return cur;
    }
    else{
        x = miniMax(ar,arr,index*2,height-1,true);
        y = miniMax(ar,arr,index*2+1,height-1,true);
        cur = min(x,y);
        arr[height].data[arr[height].n] = x;
        arr[height].n+=1;
        arr[height].data[arr[height].n] = y;
        arr[height].n+=1;
        cout<<"Current Height : "<<height<<"\n";
        cout<<"Options : "<<x<<" , "<<y<<"\n";
        cout<<"Current Choice : "<<cur<<" by Samuel"<<"\n";
        return cur;
    }
}
```

```
void print(bucket *ar,int n){
    for(int i=n+1;i>=1;i--){
        for(int j=0;j<ar[i].n;j++)
            cout<<ar[i].data[j]<<" ";
        cout<<"\n";
    }
}

int main(){
    int depth,n;
    cout<<"Depth : ";
    cin>>depth;

    n = (int)pow(2,depth+1) - 1;

    int *ar = new int[n];
    bucket arr[depth+1];

    for(int i=1;i<=depth+1;i++)
        arr[i].n = 0;

    cout<<"Leaf Nodes: ";
    for(int i=0;i<(int)pow(2,depth);i++)
        cin>>ar[i];

    int output = miniMax(ar,arr,0,depth,true);
    arr[depth+1].data[arr[depth+1].n] = output;
    arr[depth+1].n+=1;
    cout<<"The Final Tree:\n";
    print(arr,depth);
}
```

Output

Input

```
$ ./out
Depth : 3
Leaf Nodes: -1 -1 0 1 1 -1 1 1
```

```
Current Height : 1
Options : -1 , -1
Current Choice : -1 by Samuel
Current Height : 1
Options : 0 , 1
Current Choice : 1 by Samuel
Current Height : 2
Options : -1 , 1
Current Choice : -1 by Rama
Current Height : 1
Options : 1 , -1
Current Choice : 1 by Samuel
Current Height : 1
Options : 1 , 1
Current Choice : 1 by Samuel
Current Height : 2
Options : 1 , 1
Current Choice : 1 by Rama
Current Height : 3
Options : -1 , 1
Current Choice : 1 by Samuel
The Final Tree:
1
-1 1
-1 1 1 1
-1 -1 0 1 1 -1 1 1
```