# α—β剪枝算法

## 实验目的

用α-β剪枝算法对目标树进行剪枝

## 二、算法分析

α-β剪枝用于裁剪搜索树中没有意义的不需要搜索的树枝，以提高运算速度。

假设α为下界，β为上界，对于α ≤ N ≤ β:

若 α ≤ β  则N有解。

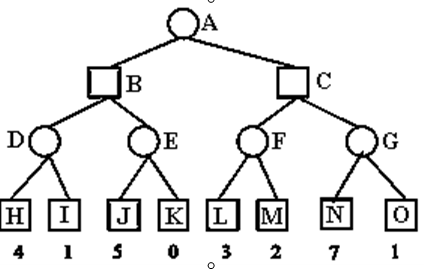
若 α > β 则N无解。

当搜索到N没有解时，则进行剪枝，该支点的其他子节点都不需要再搜索。

## 三、实验题目

利用α-β剪枝算法，对下图所示的博弈树进行搜索，搜索得到根节点选择的走步，以及没有必要进行评估的节点，并求出给出在何处发生了剪枝，以及剪枝的类型(属于α剪枝还是β剪枝)。

注：□表示MIN节点；○表示MAX节点



## 四、实验代码

Interface代码：

**package** lab\_7;

**public** **interface** Interface{

**public** **void** getStrategy(String InputFile);

}

AlphaBeta代码：

**package** lab\_7;

**import** java.util.\*;

**import** java.io.\*;

**public** **class** AlphaBeta **implements** Interface{

**final** **int** MAX\_INT=32767;

**final** **int** MIN\_INT=-32768;

**final** **int** MAX=1; //极大节点

**final** **int** MIN=0; //极小节点

**public** **class** Node

{

**private** String name;

**private** **int** value;

**private** **int** leval;//节点判断极大层还是极小层

**private** String pFather;

**private** ArrayList<String> pChildren;

Node(String name)

{

**this**.name=name;

value=-1;

pFather=**new** String();

pChildren=**new** ArrayList<String>();

}

}

**private** ArrayList<Node> NodeTree;

**private** String jianzhi[];

**private** **int** count;

**public** **void** getStrategy(String inputFile){

NodeTree=**new** ArrayList<Node>();

jianzhi=**new** String[20];

count=0;

readTree(inputFile);

Alph\_Beta(NodeTree.get(0).name);

System.***out***.println(count);

String bestRoute="";

**for**(**int** i=0;i<NodeTree.get(0).pChildren.size();i++)

{

**if**(NodeTree.get(0).value==NodeTree.get(search(NodeTree.get(0).pChildren.get(i))).value)

{

bestRoute=NodeTree.get(0).name+" "+NodeTree.get(0).value+" "+NodeTree.get(search(NodeTree.get(0).pChildren.get(i))).name;

**break**;

}

}

System.***out***.println(bestRoute);

**for**(**int** i=0;i<count;i++)

{

System.***out***.println(jianzhi[i]);

}

}

**void** Alph\_Beta(String str)

{

**boolean** flag=**false**;

Node nNode=NodeTree.get(search(str));

**if**(nNode.leval==MAX)

{

**for**(**int** i=0;i<nNode.pChildren.size();i++)

{

Alph\_Beta(nNode.pChildren.get(i));

**if**(nNode.value<NodeTree.get(search(nNode.pChildren.get(i))).value)

{

nNode.value=NodeTree.get(search(nNode.pChildren.get(i))).value;

**if**(Beta(str))//是否在极大点出执行Beta剪枝

{

jianzhi[count]=str+":";

**for**(**int** j=i+1;j<nNode.pChildren.size();j++)

{

jianzhi[count]=jianzhi[count]+" "+nNode.pChildren.get(j)+" β剪枝 ";

flag=**true**;

}

**if**(flag==**true**)

{

count++;

}

**return**;

}

}

}

}

**else**

{

**for**(**int** i=0;i<nNode.pChildren.size();i++)

{

Alph\_Beta(nNode.pChildren.get(i));

**if**(nNode.value>NodeTree.get(search(nNode.pChildren.get(i))).value)

{

nNode.value=NodeTree.get(search(nNode.pChildren.get(i))).value;

**if**(Alpha(str))

{

jianzhi[count]=str+":";

**for**(**int** j=i+1;j<nNode.pChildren.size();j++)

{

jianzhi[count]=jianzhi[count]+" "+nNode.pChildren.get(j)+" α剪枝";

flag=**true**;

}

**if**(flag==**true**)

{

count++;

}

**return**;

}

}

}

}

}

**boolean** Alpha(String str)

{

Node nNode=NodeTree.get(search(str));

**if**(nNode.pFather==**null**)

{

**return** **false**;

}

**int** i=search(nNode.pFather);

**while**(i>=0)

{

**if**((NodeTree.get(i).value>=nNode.value)&&

(NodeTree.get(i).leval==MAX)&&((NodeTree.get(i).value!=MIN\_INT)))

**return** **true**;

**else**

{

**if**(i!=0)

{

i=search(NodeTree.get(i).pFather);//其祖先节点

}

**else**

**break**;

}

}

**return** **false**;

}

**boolean** Beta(String str)

{

Node nNode=NodeTree.get(search(str));

**if**(nNode.pFather==**null**)

{

**return** **false**;

}

**int** i=search(nNode.pFather);

**while**(i>=0)

{

**if**((NodeTree.get(i).value<=nNode.value)&&

(NodeTree.get(i).leval==MIN)&&((NodeTree.get(i).value!=MAX\_INT)))

**return** **true**;

**else**

{

**if**(i!=0)

{

i=search(NodeTree.get(i).pFather);

}

**else**

**break**;

}

}

**return** **false**;

}

**public** **void** readTree(String filename)

{

File file=**new** File(filename);

String nodename[]=**new** String[10];

**try**

{

BufferedReader in=**new** BufferedReader(**new** FileReader(file));

String s;

s=in.readLine();

**if**(s.startsWith("ROOT"))

{

nodename=s.split("\\s+");

}

NodeTree.add(**new** Node(nodename[1]));

NodeTree.get(0).leval=MAX;

NodeTree.get(0).value=MIN\_INT;

NodeTree.get(0).pFather=**null**;

**while**(!(s=in.readLine()).equals("VALUE"))

{

nodename=s.split("\\s+");

**for**(**int** i=1;i<nodename.length-1;i++)

{

NodeTree.get(search(nodename[0])).pChildren.add(nodename[i]);

Node nNode=**new** Node(nodename[i]); //value为-1；

nNode.pFather=nodename[0];

**if**(NodeTree.get(search(nodename[0])).leval==MAX)

{

nNode.leval=MIN;

nNode.value=MAX\_INT;

}

**else**

{

nNode.leval=MAX;

nNode.value=MIN\_INT;

}

NodeTree.add(nNode);

}

}

String nodeValue[]=**new** String[10];

**while**(!(s=in.readLine()).equals("END"))

{

nodeValue=s.split("\\s+");

NodeTree.get(search(nodeValue[0])).value=Integer.*parseInt*(nodeValue[1]);

}

in.close();

}**catch**(Exception e){

System.***out***.println("Error!!");}

}

**int** search(String str)

{

**for**(**int** i=0;i<NodeTree.size();i++)

{

**if**(NodeTree.get(i).name.equals(str))

**return** i;

}

**return** -1;

}

**public** **static** **void** main(String argv[]){

String test = "test.txt";

**new** AlphaBeta().getStrategy(test);

}

}

Test.txt代码：

ROOT A

A B C END

B D E END

C F G END

D H I END

E J K END

F L M END

G N O END

VALUE

H 4

I 1

J 5

K 0

L 3

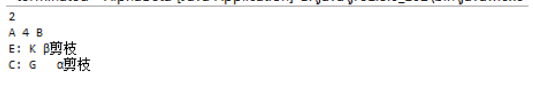
M 2

N 7

O 1

END

## 五、实验结果：



## 六、实验总结

Minimax算法有一个很大的问题就是计算复杂性。α-β剪枝是对Minimax的补充和改进。采用α-β剪枝后，我们可不必构造和搜索最大深度D内的所有节点，在构造过程中，如果发现当前格局再往下不能找到更好的解，我们就停止在这个格局及以下的搜索，也就是剪枝。α-β算法基本方法就是从根节点开始采用深度优先的方式构造格局树。