Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)

Prerequisites: Minimax Algorithm in Game Theory, Evaluation Function in Game Theory

Alpha-Beta pruning is not actually a new algorithm, but rather an optimization technique for the 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.

Let's define the parameters 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 below.

Pseudocode :

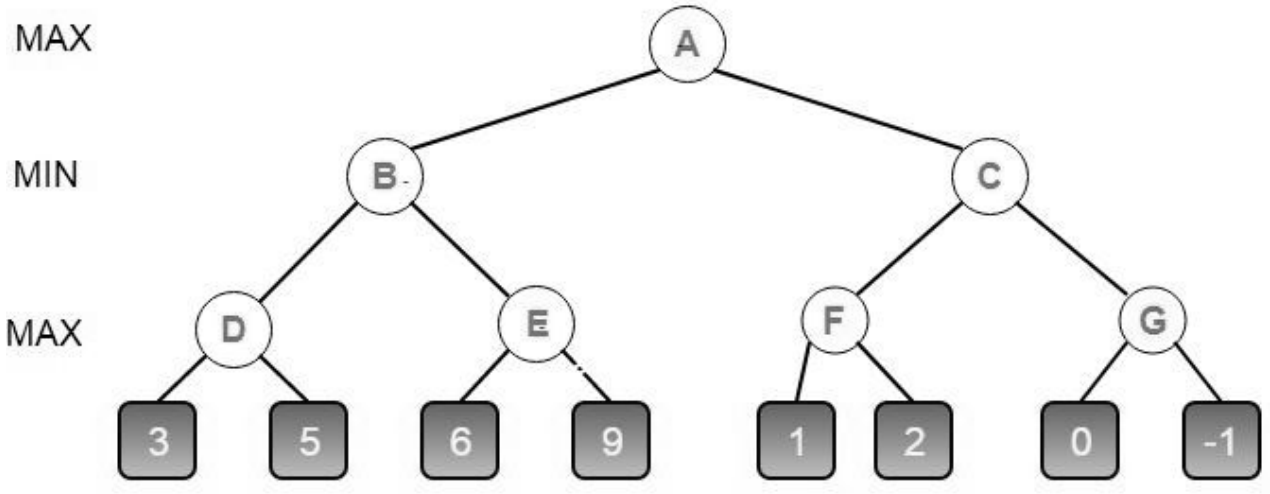
```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
```

```
if isMaximizingPlayer :
    bestVal = -INFINITY
    for each child node :
        value = minimax(node, depth+1, false, alpha, beta)
        bestVal = max( bestVal, value)
        alpha = max( alpha, bestVal)
        if beta <= alpha:
            break
    return bestVal

else :
    bestVal = +INFINITY
    for each child node :
        value = minimax(node, depth+1, true, alpha, beta)
        bestVal = min( bestVal, value)
        beta = min( beta, bestVal)
        if beta <= alpha:
            break
    return bestVal
```

```
// Calling the function for the first time.
minimax(0, 0, true, -INFINITY, +INFINITY)
```

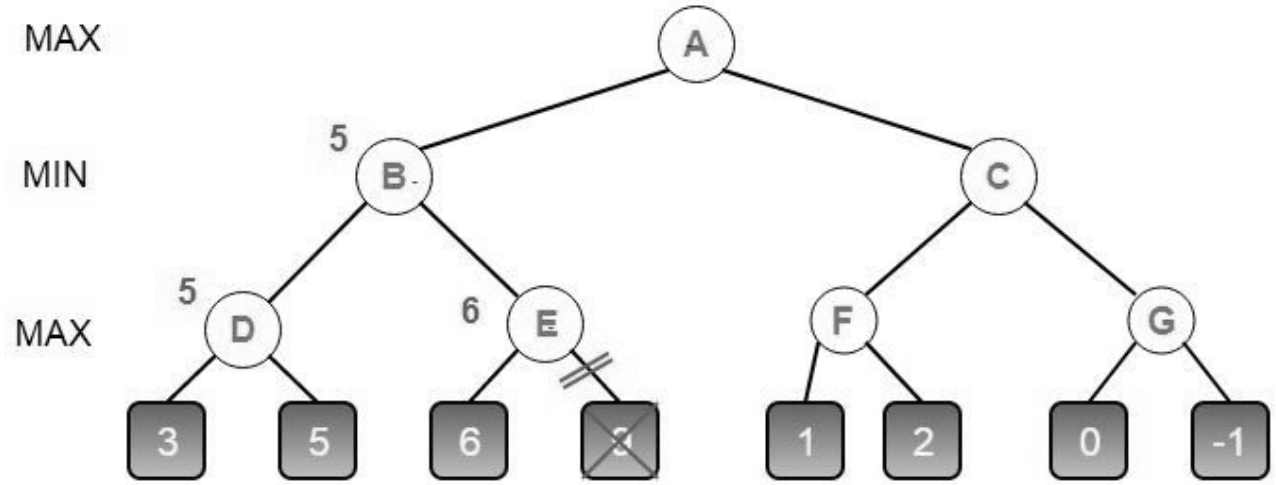
Let's make the above algorithm clear with an example.



- The initial call starts from A. The value of alpha here is -INFINITY and the value of beta is +INFINITY.

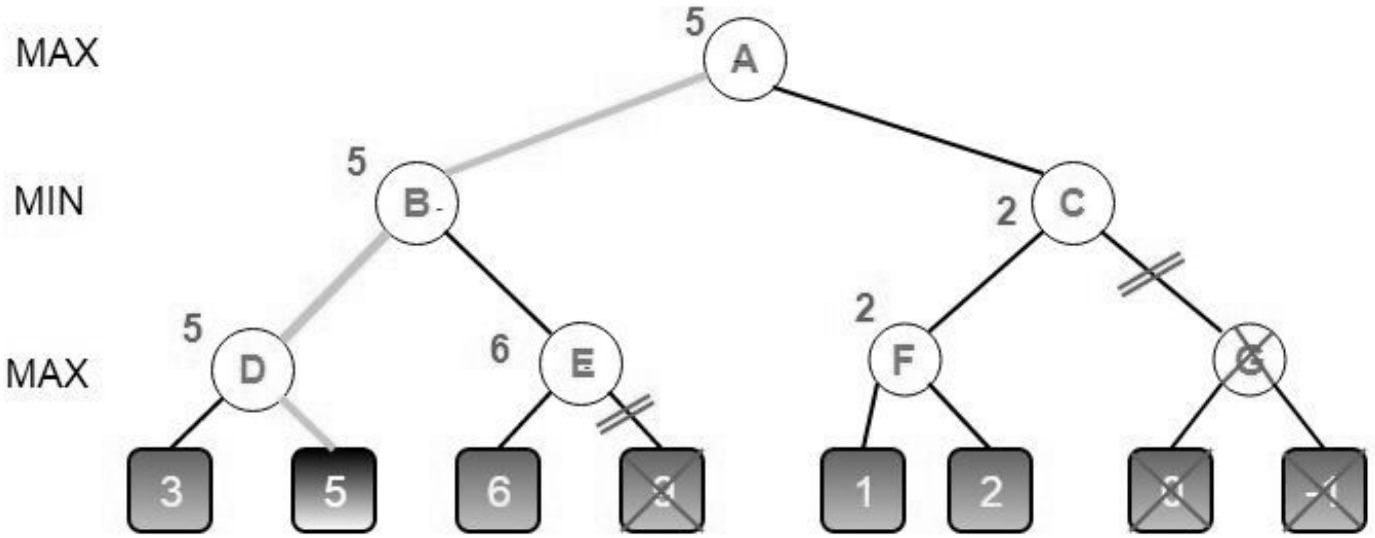
- At B it the minimizer must choose min of D and E and hence calls D first.
- At D, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at D is max(-INF, 3) which is 3.
- To decide whether its worth looking at its right node or not, it checks the condition beta<=alpha. This is false since beta = +INF and alpha = 3. So it continues the search.
- D now looks at its right child which returns a value of 5.At D, alpha = max(3, 5) which is 5. Now the value of node D is 5
- D returns a value of 5 to B. At B, beta = min(+INF, 5) which is 5. The minimizer is now guaranteed a value of 5 or lesser. B now calls E to see if he can get a lower value than 5.
- At E the values of alpha and beta is not -INF and +INF but instead -INF and 5 respectively, because the value of beta was changed at B and that is what B passed down to E
- Now E looks at its left child which is 6. At E, alpha = max(-INF, 6) which is 6. Here the condition becomes true. beta is 5 and alpha is 6. So beta<=alpha is true. Hence it breaks and E returns 6 to B
- Note how it did not matter what the value of E's right child is. It could have been +INF or -INF, it still wouldn't matter, We never even had to look at it because the minimizer was guaranteed a value of 5 or lesser. So as soon as the maximizer saw the 6 he knew the minimizer would never come this way because he can get a 5 on the left side of B. This way we didn't have to look at that 9 and hence saved computation time.
- E returns a value of 6 to B. At B, beta = min(5, 6) which is 5.The value of node B is also 5

So far this is how our game tree looks. The 9 is crossed out because it was never computed.



- B returns 5 to A. At A, alpha = max(-INF, 5) which is 5. Now the maximizer is guaranteed a value of 5 or greater. A now calls C to see if it can get a higher value than 5.
- At C, alpha = 5 and beta = +INF. C calls F
- At F, alpha = 5 and beta = +INF. F looks at its left child which is a 1. alpha = max(5, 1) which is still 5.
- F looks at its right child which is a 2. Hence the best value of this node is 2. Alpha still remains 5
- F returns a value of 2 to C. At C, beta = min(+INF, 2). The condition beta <= alpha becomes true as beta = 2 and alpha = 5. So it breaks and it does not even have to compute the entire sub-tree of G.
- The intuition behind this break-off is that, at C the minimizer was guaranteed a value of 2 or lesser. But the maximizer was already guaranteed a value of 5 if he choose B. So why would the maximizer ever choose C and get a value less than 2 ? Again you can see that it did not matter what those last 2 values were. We also saved a lot of computation by skipping a whole sub-tree.

This is how our final game tree looks like. As you can see G has been crossed out as it was never computed.



CPP

```
// C++ program to demonstrate
// working of Alpha-Beta Pruning
#include<bits/stdc++.h>
using namespace std;
```

```
const int MIN = -1000;

// Returns optimal value for
// current player(Initially called
// for root and maximizer)
int minimax(int depth, int nodeIndex,
            bool 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 = max(best, val);
            alpha = max(alpha, best);

            // Alpha Beta Pruning
            if (beta <= alpha)
                break;
```

```
    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 = min(best, val);
            beta = min(beta, best);

            // Alpha Beta Pruning
            if (beta <= alpha)
                break;
        }
        return best;
    }
}

// Driver Code
int main()
{
    int values[8] = { 3, 5, 6, 9, 1, 2, 0, -1 };
    cout <<"The optimal value is : "<< minimax(0, 0, true, values, MIN, MAX);;
    return 0;
}
```

Java

```
// Java program to demonstrate
```

```
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
```

```
        }
        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));

    }
```

```
// This code is contributed by vt_m.
```

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)
```

```
        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))

# This code is contributed by Rituraj Jain
```

C#

```
// C# program to demonstrate
```

```
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);
```

```
                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};
    Console.WriteLine("The optimal value is : " +
                      minimax(0, 0, true, values, MIN, MAX));
```

```
// This code is contributed by 29AjayKumar
```

Javascript

```
// Javascript program to demonstrate
// working of Alpha-Beta Pruning

// Initial values of
// Alpha and Beta
let MAX = 1000;
let MIN = -1000;

// Returns optimal value for
// current player (Initially called
// for root and maximizer)
function minimax(depth,nodeIndex,maximizingPlayer,values,alpha,beta)
{
    // Terminating condition. i.e
    // leaf node is reached
    if (depth == 3)
        return values[nodeIndex];

    if (maximizingPlayer)
    {
        let best = MIN;

        // Recur for left and
        // right children
        for (let i = 0; i < 2; i++)
        {
            let val = minimax(depth + 1, nodeIndex * 2 + i,
```

```
            // Alpha Beta Pruning
            if (beta <= alpha)
                break;
        }
        return best;
    }
    else
    {
        let best = MAX;

        // Recur for left and
        // right children
        for (let i = 0; i < 2; i++)
        {

            let 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
let values=[3, 5, 6, 9, 1, 2, 0, -1];
document.write("The optimal value is : " +
                           minimax(0, 0, true, values, MIN, MAX));
```

Output

```
The optimal value is : 5
```

Feeling lost in the world of random DSA topics, wasting time without progress? It's time for a change! Join our DSA course, where we'll guide you on an exciting journey to master DSA efficiently and on schedule. Ready to dive in? Explore our Free Demo Content and join our DSA course, trusted by over 100,000 geeks!

- DSA in C++
- DSA in Java
- DSA in Python
- DSA in JavaScript

Recommended Problems

Frequently asked DSA Problems

Solve Problems

Previous

Check if the game is valid or not

Next

Longest Common Extension | LCE using Segment Tree

Share your thoughts in the comments

Add Your Comment

Similar Reads

- Minimax Algorithm in Game Theory | Set 1 (Introduction)
- Minimax Algorithm in Game Theory | Set 5 (Zobrist Hashing)
- Introduction to Evaluation Function of Minimax Algorithm in Game Theory
- Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory
- Game Theory (Normal form game) | Set 2 (Game with Pure Strategy)
- Game Theory (Normal-form game) | Set 3 (Game with Mixed Strategy)
- Game Theory (Normal-form Game) | Set 4 (Graphical Method | M X 2 | Game)
- Game Theory (Normal-form Game) | Set 6 (Graphical Method | 2 X N | Game)
- Game Theory (Normal - form game) | Set 1 (Introduction)
- Game Theory (Normal-form Game) | Set 5 (Dominance Property-Mixed Strategy)

A Akshay L Arachya

Article Tags : DSA , Game Theory

Practice Tags : Game Theory

Additional Information

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Company

About Us

Legal

Careers

In Media

Contact Us

Advertise with us

GFG Corporate Solution

Placement Training Program

Explore

Job-A-Thon Hiring Challenge

Hack-A-Thon

GFG Weekly Contest

Offline Classes (Delhi-NCR)

DSA in JAVA/C++

Master System Design

Master CP

GeeksforGeeks Videos

Geeks Community

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

Tutorials Archive

DSA

Data Structures

Algorithms

DSA for Beginners

Basic DSA Problems

DSA Roadmap

Top 100 DSA Interview Problems

DSA Roadmap by Sandeep Jain

All Cheat Sheets

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

ML Maths

Data Visualisation Tutorial

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

Web Scraping

OpenCV Python Tutorial

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

– VI

CSS

Web Templates

CSS Frameworks

Bootstrap

Tailwind CSS

SASS

FSS

Web Design

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Competitive Programming

Top DS or Algo for CP

Top 50 Tree

Top 50 Graph

Top 50 Array

GCP

DevOps Roadmap

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

NCERT Solutions

Class 12

Class 11

Class 10

Class 9

Class 8

Complete Study Material

Commerce

Accountancy

Business Studies

Top 50 DP

Top 15 Websites for CP

JavaScript

JavaScript Examples

TypeScript

ReactJS

NextJS

AngularJS

NodeJS

Lodash

Web Browser

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

UPSC Study Material

Polity Notes

Geography Notes

HR Management

Finance

Income Tax

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

SSC CGL Practice Papers

Companies

TCS Owned Companies

Alphabet Owned Companies

TATA Group Owned Companies

Reliance Owned Companies

Fintech Companies

EdTech Companies

Exams

JEE Mains

JEE Advanced

GATE CS

Economy Notes

Ethics Notes

Previous Year Papers

Colleges

Indian Colleges Admission & Campus Experiences

List of Central Universities - In India

Colleges in Delhi University

IIM Colleges

NIT Colleges

IIIT Colleges

Preparation Corner

Company-Wise Recruitment Process

Resume Templates

Aptitude Preparation

Puzzles

Company-Wise Preparation

More Tutorials

Software Development

Software Testing

Product Management

Linux

Excel

Free Online Tools

Typing Test

Image Editor

Code Formatters

Code Converters

Currency Converter

Random Number Generator

Random Password Generator

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships