Arbol de Juego

Nicolas Muñoz
UNPSJB – APU – ALGORITMICA Y PROGRAMACIÓN II

Game Tree

Arbol de juego (Game Tree)

Los árboles de juego, o "Game Tree", son una representación de los posibles movimientos en un juego. Son utilizados en inteligencia artificial y teoría de juegos para modelar la toma de decisiones y predecir los resultados de juegos con varios jugadores.

Son representados por Árboles donde cada nodo es un estado posible del juego y las aristas son las decisiones del jugador.

Características de los Árboles de Juego:

Nodos: Cada nodo en un árbol de juego representa un estado del juego en un punto específico.

Aristas: Las aristas (o ramas) conectan los nodos y representan las posibles acciones o movimientos que pueden tomarse para ir de un estado a otro.

Raíz: El nodo raíz representa el estado inicial del juego.

Hojas: Los nodos hoja representan los estados terminales del juego, es decir, cuando el juego ha terminado.

Profundidad: La profundidad de un nodo se refiere a la cantidad de movimientos desde la raíz hasta ese nodo.

Tipos de Juegos y Representación:

Los árboles de juego se pueden utilizar para juegos de dos jugadores (como el ajedrez, el go, el tic-tac-toe) y juegos de un solo jugador (como puzles o problemas de laberintos).

Ejemplo de Aplicación:

Consideremos un juego simple como el Ta-Te-Ti. El árbol de juego para el Ta-Te-Ti comenzaría con un tablero vacío como nodo raíz. Cada posible movimiento del jugador (colocar una X en una celda vacía) se representaría como un hijo del nodo raíz. Cada uno de estos nodos hijos representaría un nuevo estado del juego después de un movimiento, y así sucesivamente.

En juegos como el ajedrez, es algo más complejo, el árbol por lo general se vuelve muy grande y esto provoca ocupar muchísima memoria, por lo que siempre se tratará de mostrar una porción del arbol.

Existen varios algoritmos que tratan los arboles de juegos, me centraré en explicar dos de los más interesantes, Backtracking y Minimax.

Algoritmo Backtracking

El Backtracking es un enfoque utilizado en algoritmos para resolver problemas, especialmente aquellos relacionados con la búsqueda y exploración de soluciones en espacios de estados complejos. Este enfoque se utiliza comúnmente en problemas de optimización, combinación y permutación, así como en la resolución de problemas de decisión.

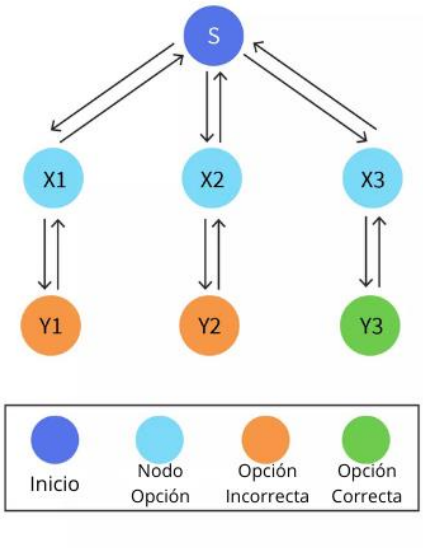
En el contexto de los algoritmos, el Backtracking implica explorar sistemáticamente todas las posibles soluciones de un problema mediante una búsqueda en profundidad, y retroceder (o hacer "backtrack") cuando se determina que la solución actual no puede llevar a una solución completa y válida.

El proceso general del Backtracking incluye:

1. ***Elección de una opción:*** Se elige una opción válida para avanzar en la solución del problema.
2. ***Exploración:*** Se explora más profundamente la opción elegida, y se repite el proceso.
3. ***Rechazo:*** Si se determina que la opción elegida no puede conducir a una solución válida, se retrocede (backtrack) al estado anterior y se elige otra opción.
4. ***Éxito:*** Si se encuentra una solución completa y válida, se la registra o se maneja según los requisitos del problema.

El ejemplo más claro es el Sudoku, aquí el algoritmo buscará la mejor opción para llenar cada casillero del tablero cumpliendo con ciertas restricciones principales del sudoku, los números no puede repetirse en el mismo cuadrado, fila o columna.

A partir de esto el algoritmo creará un árbol con cada opción posible y hará una búsqueda en profundidad hasta encontrar la mejor opción, si el nodo no contiene una opción correcta se hará un retroceso o backtrack para seguir con los demás nodos, descartando así pasar por absolutamente todos los nodos del arbol y encontrar mucho más rápido la mejor decisión.



Cómo podemos ver en el gráfico se muestra un arbol simple donde se explica el recorrido.

En el nodo X1 se encuentra una opción incorrecta que es Y1, al igual que en X2 el cual Y2 no es correcto, en X3 se encuentra que Y3 es la mejor opción para el árbol y por lo tanto se elige esa.

Análisis de complejidad de Backtracking

Dado que el algoritmo de retroceso es puramente de fuerza bruta, en términos de complejidad temporal, funciona muy mal. En general, se puede observar que el retroceso tiene las complejidades de tiempo que se mencionan a continuación:

- Exponencial (O(K^N))
- Factorial (O(N!))

Estas complejidades se deben al hecho de que en cada estado tenemos múltiples opciones, por lo que el número de caminos aumenta y los subárboles se expanden rápidamente.

Historia y evolución del backtracking

El concepto del backtracking tiene sus raíces en la metodología creada por el matemático británico Alan Turing durante los primeros años de la informática. A partir de ahí, ha evolucionado y ha ayudado a avanzar enormemente en el campo de los algoritmos, concibiéndose muchas nuevas variaciones del método.

Algoritmo Minimax

Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario. El objetivo del jugador puede resumirse en: elegir el mejor movimiento para ti mismo suponiendo que tu contrincante escogerá el peor movimiento para vos.

Historia Algoritmo Minimax

En 1921 el matemático francés Emile Borel fue el primero en mostrar una forma de ver a los juegos competitivos y de estudiar las estrategias que aplicaban a los juegos de suma cero. A pesar de ser considerarse el primero en estudiarlo, se le suele atribuir a John Von Neumann el mérito de indagar aún más sobre Minimax, ya que él en un artículo de 1928 planto las bases de lo que hoy se conoce como algoritmo Minimax.

Von Neumann dio la siguiente declaración en su artículo:

"Un juego es una situación conflictiva en la que uno debe tomar una decisión sabiendo que los demás también toman decisiones, y que el resultado del conflicto se determina, de algún modo, a partir de todas las decisiones realizadas."

Características

El algoritmo Minimax sirve para elegir la jugada más óptima entre todas las disponibles para un momento de la partida en concreto.

No todos los juegos pueden utilizarlo, deben cumplir con ciertas propiedades:

- 2 jugadores
- Por turnos
- Cada jugador debe saber con exactitud lo que ocurre en la partida.
- +0, es decir, que buscaremos que nuestra puntuación sea mejor que la del rival.

Un ejemplo más claro de este algoritmo es el TA-TE-TI (Tic-Tac-Toe), este se juego se basa en hacer la menor cantidad de movimientos para derrotar a tu rival.

Un punto importante en este juego es saber ¿Qué pasa cuando el juego termina? esto se llama estado terminal, estos son estados en los que el juego termina, en este ejemplo, es cuando X consigue 3 seguidos, O consigue 3 seguidos o cuando hay un empate, es decir, se llenaron todos los espacios del tablero.

Le asignaremos un número a cada estado terminal:

- Cuando X gane se asignará 1.
- Cuando O gane se asignará -1.
- Cuando ocurra un empate se asignará 0.

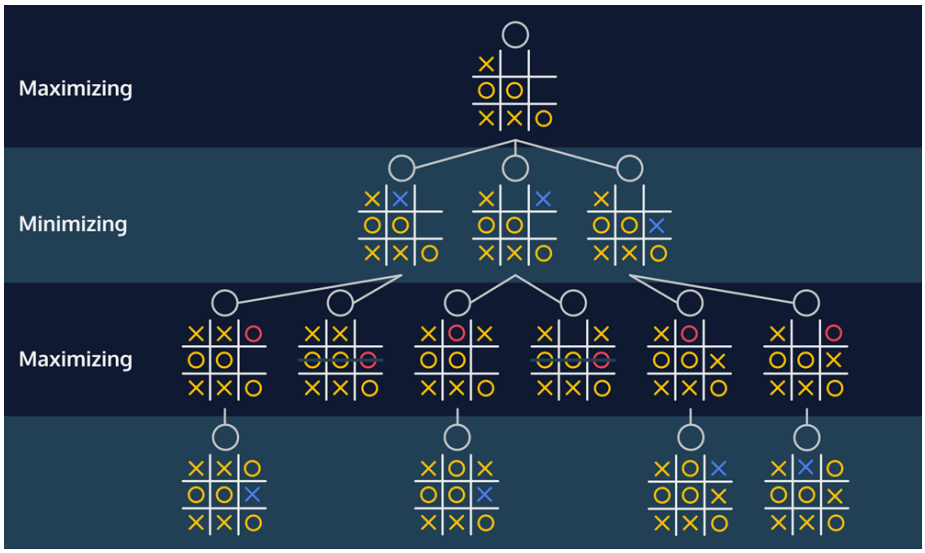
Esto da como resultado darles a los jugadores un objetivo matemático.

Podemos usar estos valores para ayudarnos a decidir qué elecciones tomar mientras estemos jugando.

En este ejemplo podemos ver como funciona el algoritmo de manera más gráfica.

El estado actual del juego es la raíz, los nodos siguientes son las posibles jugadas. A partir de acá, el algoritmo comienza a trabajar de manera recursiva para elegir el mejor movimiento para la máquina, en este caso, busca maximizar el contador, es por eso que siempre elige las jugadas marcadas con 0 y no las de -1.

En el siguiente GIF podemos ver un ejemplo simple de cómo podría ser un arbol de juego en Ta-Te-Ti.



En juegos más complejos como el ajedrez, el arbol se vuelve exponencialmente más grande y empiezan a ocurrir problemas con memoria.

Complejidad Temporal de Minimax

La complejidad en tiempo del algoritmo Minimax es O(b^m), donde b es el factor de ramificación y m es la profundidad máxima del árbol de decisión. Si bien la búsqueda se realiza primero en profundidad, se deben explorar todos los nodos terminales a menos que se imponga un límite de profundidad, en cuyo caso la decisión puede ser imperfecta y no se puede garantizar el resultado óptimo.

- **Complejidad en tiempo O(b^m):** Esto significa que la cantidad de tiempo que toma el algoritmo en el peor de los casos es proporcional a b (el número de posibles movimientos en cada turno) elevado a m (la profundidad máxima del árbol).
- **Factor de ramificación (b):** Es el número promedio de hijos por nodo, es decir, el número de posibles movimientos en un estado del juego.
- **Profundidad máxima (m):** Es el número máximo de niveles en el árbol de decisión, correspondiente al número de movimientos hasta llegar a un estado terminal (como una victoria, derrota o empate).

Cantidad de Hojas de Minimax en el peor caso

- El número de posibles secuencias de movimientos es dado por el factorial del número de celdas: 9! = 9×8×7×6×5×4×3×2×1= 362,880

- Este número 9! representa la cantidad total de permutaciones de movimientos si cada celda fuera ocupada en cada movimiento sin considerar condiciones de victoria.

Fuentes:

Minimax:

Videos utilizados:

https://youtu.be/QJjM7EKDRuc?si=X6Tp8nAfoKgXh-zX

https://youtu.be/-c8ExcmXDGs?si=KkV_onxbo7RaxdjR

https://youtu.be/SLgZhpDsrfc?si=2hao4Ek63abOyaup

Geeksforgeeks.com:

https://www.geeksforgeeks.org/introduction-to-evaluation-function-of-minimax-algorithm-in-game-theory/?ref=header_search

https://www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/?ref=ml_lbp

Toolify.ai:

https://www.toolify.ai/es/ai-news-es/algoritmo-minimax-estrategia-ganadora-en-ia-648868

PDF: “Introducción a la Inteligencia Artificial – Modulo 5 Juegos como problema de Búsqueda”

Backtracking:

Videos utilizados:

https://youtu.be/ip2jC_kXGtg?si=4S6kuNI0CIVI82oF

https://youtu.be/-bjTb0o6EAQ?si=liFFkB03An3q4uQh

swhosting.com:

https://www.swhosting.com/es/comunidad/manual/en-que-consiste-el-algoritmo-de-backtracking-y-como-aplicarlo-en-c#:~:text=El%20algoritmo%20de%20backtracking%20es,que%20no%20cumplen%20ciertas%20condiciones.

Documento Online:

https://docs.jjpeleato.com/