**fTema 3 Algoritmos de Búsqueda**

**Algoritmos de búsqueda adversarial**

**Problemas y espacios de estados**

* Muchos problemas que pueden resolverse aplicando técnicas de Inteligencia Artificial, se modelan en forma **simbólica** y **discreta,** definiendo las configuraciones posibles del universo que describe el problema
* El **problema** se plantea en términos de encontrar una **configuración objetivo** a partir de una **configuración inicial** dada, aplicando **transformaciones validas** según el modelo del universo. La **respuesta** es la secuencia de transformaciones cuya aplicación sucesiva lleva a la configuración deseada

**Estados y reglas del problema**

* Los ejemplos más característicos de esta categoría son los juegos (Universos restringidos fáciles de modelar)
* En un juego, las configuraciones del **universo** corresponden directamente a las configuraciones del tablero. Cada configuración es un **estado** que puede ser esquematizado gráficamente y representado de forma simbólica
* Las transformaciones permitidas corresponden a las reglas o movimientos del juego, formalizadas como transiciones de estados

**Soluciones de los problemas**

* Problemas con mas de una solución valida
* Entre las soluciones posibles
  + Buscar la menos costosa
  + Buscar la más rápida

**Estados y operadores**

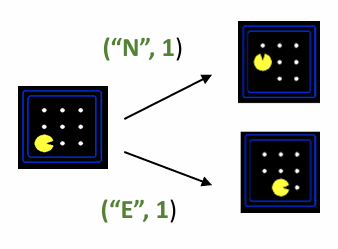
* Formalmente, un **espacio de estados** se define por una cuádrupla [N,A,I,F] donde:
  + **N**  es un **conjunto de nodos que representan estados** en el proceso de resolución de un problema
  + **A** es un conjunto de **arcos entre nodos,** que **corresponden a los posibles pasos (**aplicación de un operador**) en el proceso de resolución de un problema**
  + **I** es un **subconjunto no vacío** en N que contiene el/los **estados iniciales** del problema
  + **F** es un **subconjunto no vacío** en **N** que contiene el/los **estados finales** del problema

**Problemas de búsqueda**

* Un problema de búsqueda consiste en:
  + Un **espacio de estados**

****

* + Una **función** sucesor (con **acciones** y costes)

****

* + Un estado **inicial** y un **test** de haber llegado al objetivo (goal)
  + Una solución es **una secuencia de acciones (**un plan**)** que **transforma el estado inicial** en un **estado objetivo** (goal state)

**¿En qué consiste realmente la solución del problema?**

* La solución del problema consiste en **alcanzar el estado final del problema**
* **Una solución** para un problema **viene dada por la secuencia de operadores** que se han aplicado para **pasar del estado inicial** hasta el **estado objetivo**

**¿Se puede descomponer el problema?**

* Existe un determinado conjunto de problemas (a los que llamaremos **divisibles**)a los cuales se puede aplicar la técnica de descomposición o divide y vencerás

**¿Pueden darse ciclos en el camino hacia la solución?**

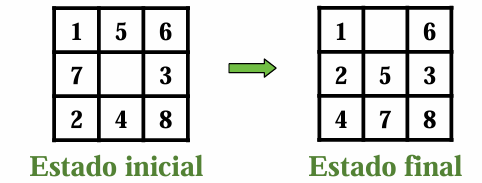
* El sistema debe **memorizar los estados** por los que ha pasado anteriormente para **evitar ciclos innecesarios**

**¿Es posible ignorar o deshacer pasos de la solución?**

* Problemas en los cuales, los **pasos dados** en la búsqueda hacia la solución **se pueden deshacer**: por ejemplo, recorrer un laberinto
* Problemas en los que **una operación es irreversible**: por ejemplo, realizar una jugada de cualquier tipo de juego

**¿Es predecible el universo del problema?**

* Problemas en los que puede plantearse la secuencia de movimientos que llevaran hasta un estado objetivo,puesto que **se sabe perfectamente como actúan los operadores** sobre un estado
  + Ejemplo el problema del puzzle



* Problemas en los que **no se conoce con precisión** cual va a ser el resultado de cada movimiento que se hace
  + Ejemplo el juego del mus Interviene el azar y los movimientos del resto de los jugadores

**¿Cuál es el papel del conocimiento?**

* En algunos problemas es **importante el conocimiento** para restringir la búsqueda de la solución
  + Ejemplo el ajedrez Conocimiento plasmado en tácticas y estrategias basadas en la experiencia de los jugadores
* En otros problemas se precisa de un conocimiento **muy amplio** tan solo para poder reconocer una solución
  + Ejemplo responder una pregunta formulada en lenguaje natural

**¿Requiere el problema la interacción con una persona?**

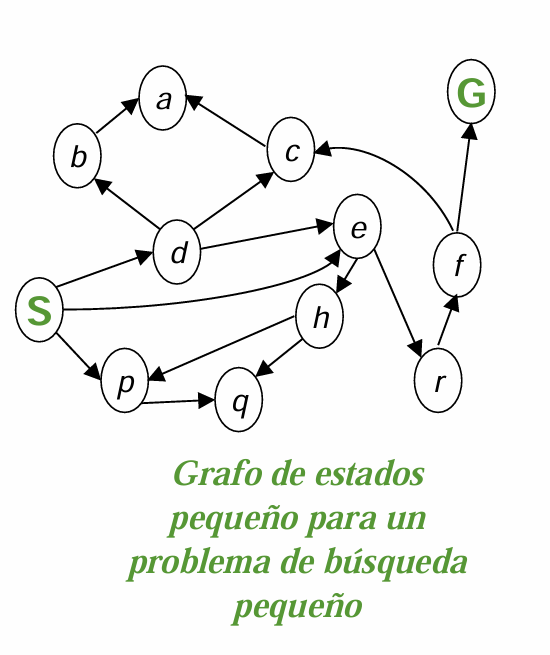
* Problemas en los que la computadora recibe una descripción del problema y produce una respuesta **sin necesitar ningún dato adicional** ni interacción alguna con el usuario
  + Ejemplo el puzzle de las 8 piezas tan solo requiere del exterior **una configuración inicial y otra configuración final**, y nos devuelve una secuencia de movimientos que son la solución
* Conversacionales Habrá comunicación entre una persona y el ordenador, bien para proporcionar **asistencia adicional** a la computadora, bien para proporcionar **información adicional** al usuario o ambas
  + Ejemplo tutores inteligentes

**¿Qué tiene un espacio de estados?**

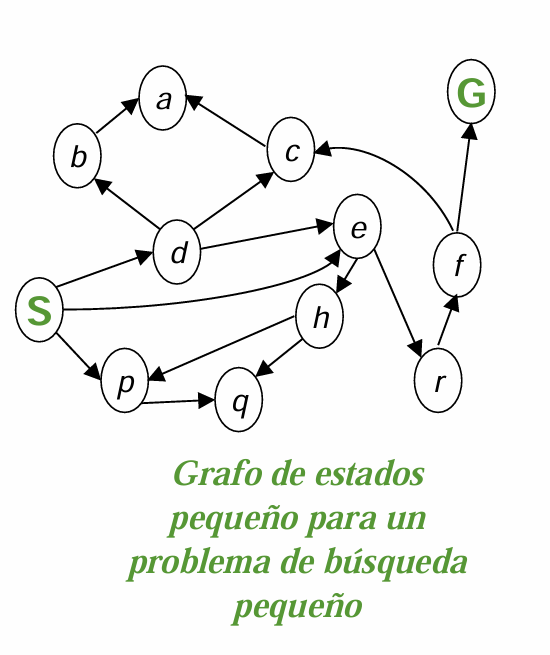
* El estado del mundo incluye todos los detalles del entorno
* Un estado de búsqueda contiene únicamente los detalles necesarios para la planificación (abstracción)

**Representación: Grafos de espacio de estados**

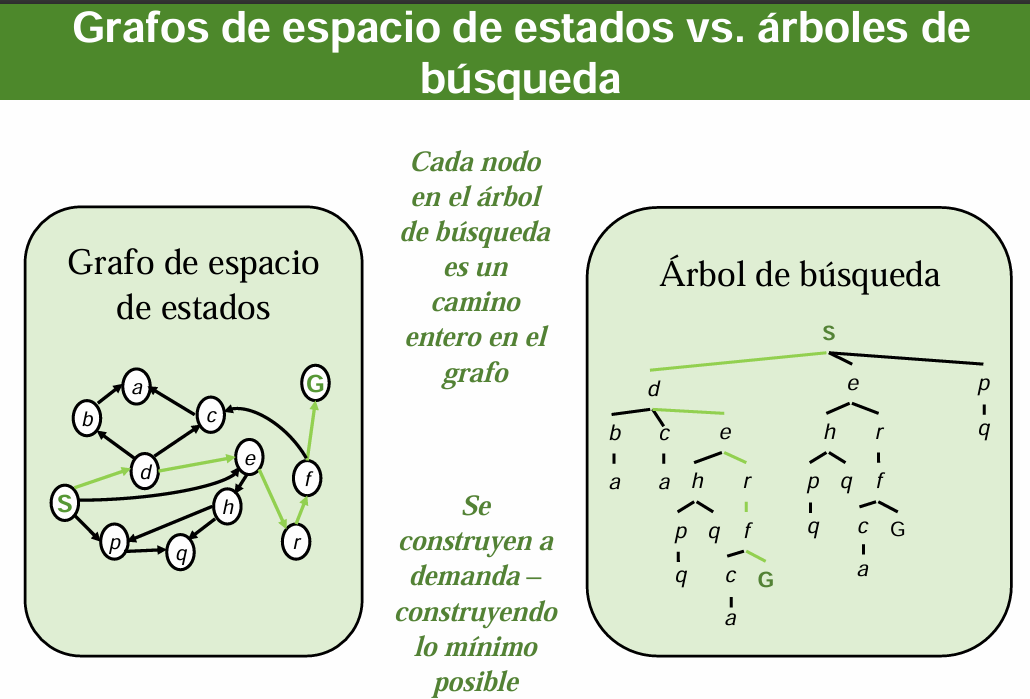
* Grafo de espacios de estados: representación matemática de un problema de búsqueda
  + Los nodos son configuraciones (abstractas) del mundo
  + Los arcos representan sucesores (resultados de accion)
  + El test de objetivo es un conjunto de nodos objetivo (puede ser solo uno)
* En un grafo de estados,cada estado aparece una sola vez
* Pocas veces podremos construir este grafo en memoria (demasiado grande),pero es una idea útil



**Representación: Arboles de búsqueda**

****

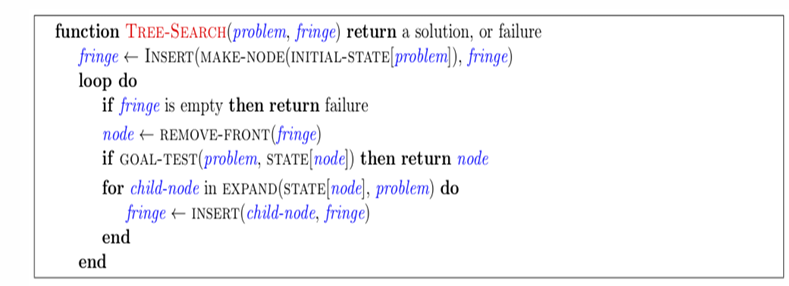
* **Árbol de búsqueda**
  + El estado inicial es el nodo raíz
  + Los hijos son los sucesores
  + Los nodos muestran estados, pero corresponden a PLANES que llevan a esos estados
  + Para la mayoría de problemas, **nunca podremos construir realmente ese árbol (demasiado grande)**

}

**Métodos de resolución de problemas. Técnicas de búsqueda**

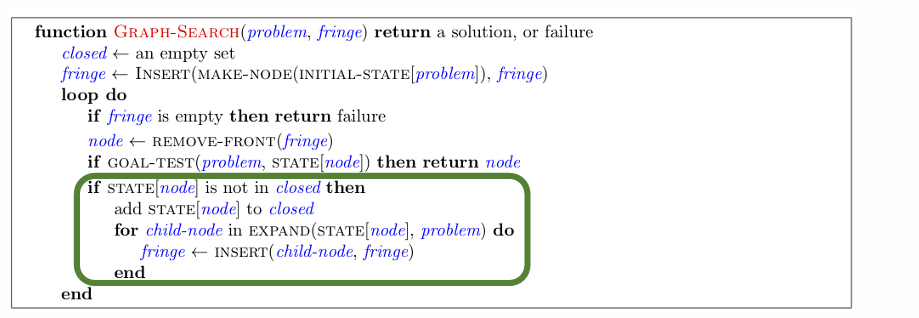
* Métodos de búsqueda no informados (Uninformed Search Method)
  + Búsqueda en profundidad (Depth First Search)
  + Busqueda en anchura (Breadth First Search)
  + British museum
  + Busqueda de coste uniforme (Uniform cost search)

**Busqueda en arbol**

****

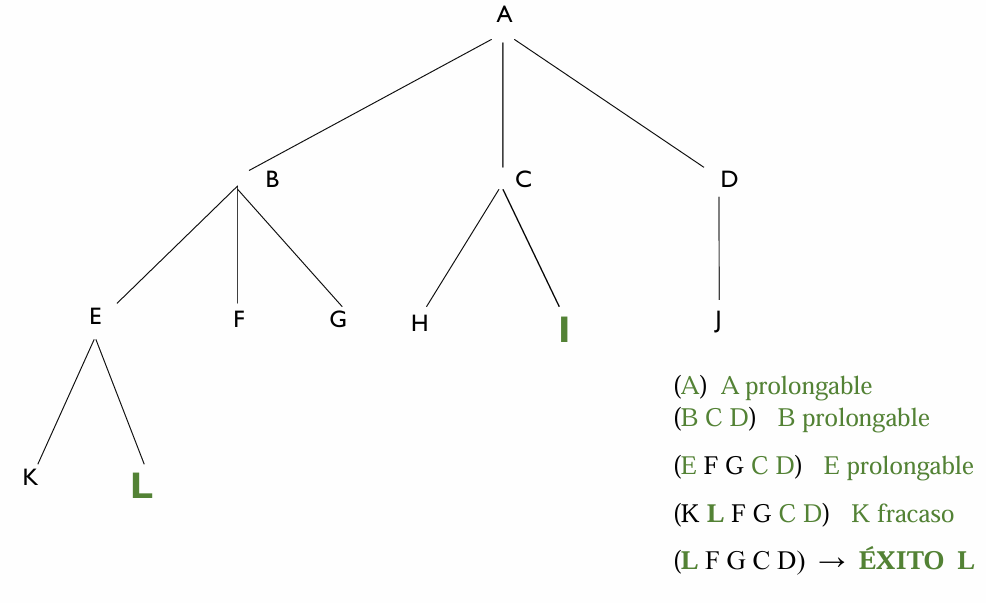
* Idea
  + Expandir planes potenciales (Nodos del árbol )
  + Mantener un borde (fringe) de planes parciales en consideración
  + Intentar expandir el mínimo numero posible de nodos del arbol

**Busqueda en grafos**

****

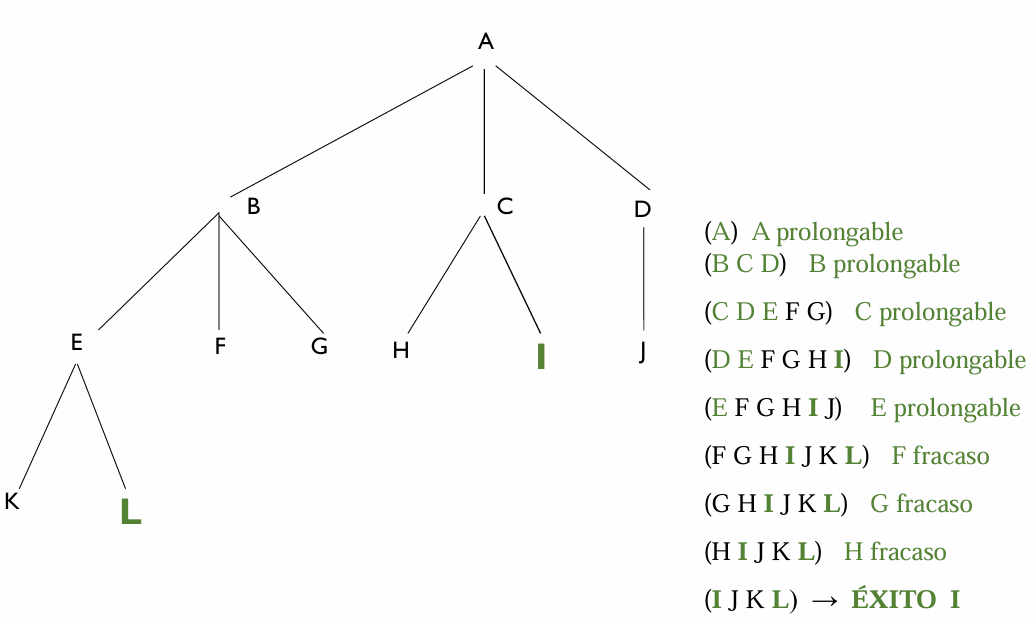
**Busqueda en profundiad**

* Algoritmo
  + Construir una lista con el nodo raíz como único elemento
  + Hasta que la lista esta vacía o el primer elemento de la lista sea el elemento objetivo
    - Eliminar el primer elemento de la lista y añadir los hijos de este elemento (si hubiera) al principio de la lista
  + Si se ha encontrado el nodo objetivo, anunciar éxito, si no fallo
* Problema
  + Caer en un camino infinitamente largo
* Estrategia
  + Expandir un nodo más profundo primero
* Implementación
  + Fringe es una pila LIFO
* ¿Es optimo?
  + No, encuentra el camino más a la izquierda sin tener en cuenta la profundidad o coste

****

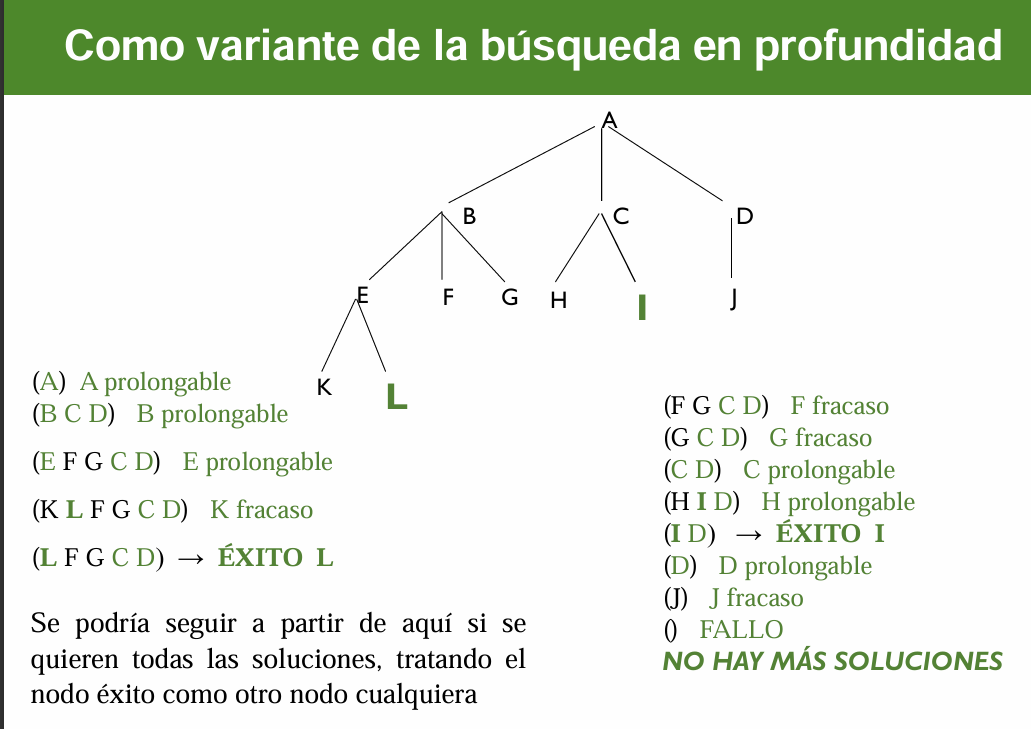
**Busqueda en anchura (Breadth First Seacrh)**

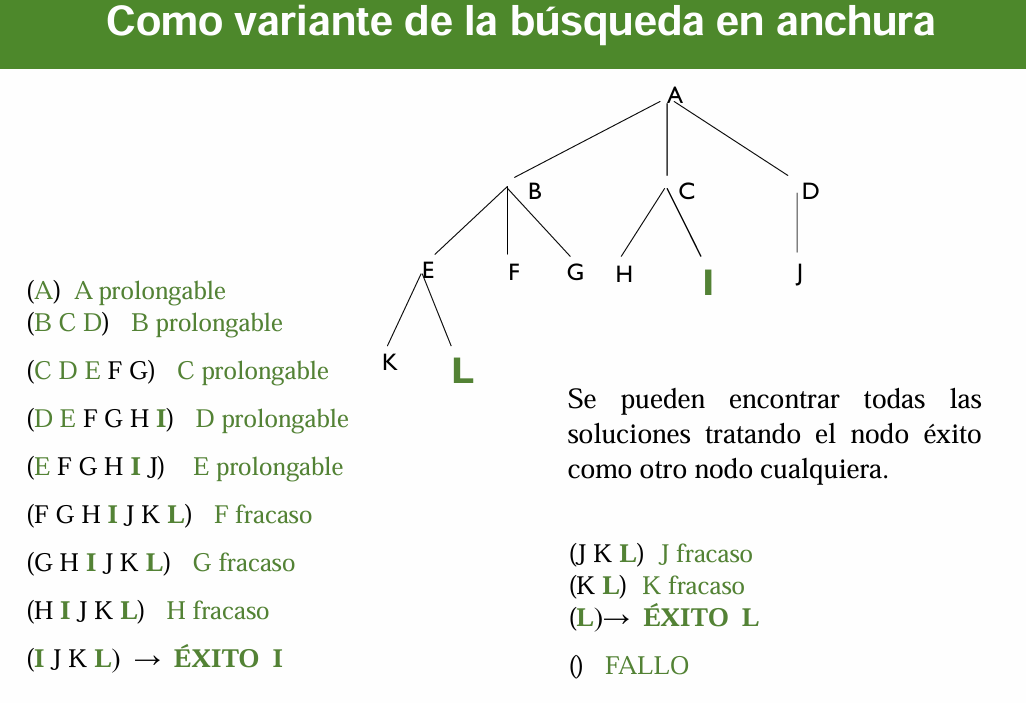
* Algoritmo
  + Construir una lista con el nodo raíz como único elemento
  + Hasta que la lista esta vacía o el primer elemento de la lista sea el elemento objetivo
    - Eliminar el primer elemento de la lista y añadir los hijos de este elemento (si los hubiera) al final de la lista
    - Si se ha encontrado el nodo objetivo, anunciar éxito, si no fallo
* Problema
  + Caer en un camino infinitamente largo
* Estrategia
  + Expandir primero un nodo del nivel más bajo
* Implementación
  + El borde (fringe) es una cola FIFO

****

**British museum**

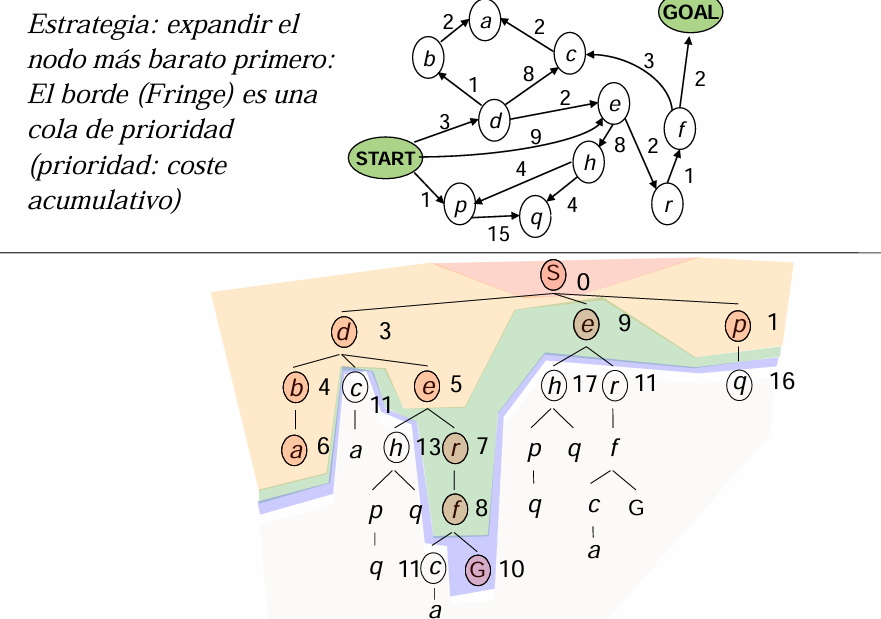
* Algortimo
  + Busca todos los posibles caminos y selecciona el mejor entre ellos
    - Puede implementarse modificando cualquiera de los dos anteriores
* Problema
  + Ineficiente en espacios de estados grandes
  + Inaplicable en problemas con explosión combinatoria





**Busqueda de coste uniforme Uniform Cost Search**

* Estrategia
  + Expandir el nodo más barato primero.
  + El borde (Fringe) es una cola de prioridad
    - Prioridad coste acumulativo



**Algoritmos de búsqueda heurística**

* Búsqueda en modelos
  + La búsqueda opera sobre modelos del mundo
    - El agente no prueba todos los planes del mundo real
    - La planificación se hace en modo “simulación”
    - La búsqueda es tan buena como nuestros modelos
* Heurísticos de búsqueda
  + Un heurístico es:
    - Conjunto de reglas que nos sirven para escoger aquellas ramas del espacio de estados que habitualmente llevan hacia una solución aceptable del problema
    - Se obtiene de la experiencia considerando las características específicas de cada problema
  + Características
    - Algunos algoritmos no garantizan que se encuentre una solución,aunque exista
    - En caso de encontrar una solución no siempre aseguran que sea la de longitud mínima o la de coste optimo (La mejor)
    - En algunas ocasiones encontrara una solución en un tiempo razonable
  + Un heurístico es una función que estima cuan cerca un estado del objetivo
  + Ejemplos
    - Distancia euclídea



* + - Distancia de manhattan



**Búsqueda Voraz (Greedy Search)**

* Expandir el nodo que parece mas cercano
  + F(n)=h(n)
* Estrategia
  + Expandir el nodo que pensamos que esta mas cerca de un estado objetivo
  + Heuristico Estimación de la distancia al objetivo mas cercano desde cada estado
* En el mejor de los casos
  + Nos lleva directamente a un objetivo (Quizás no el mejor)
* Caso peor
  + O(b^n)
* Optimo
  + No
* Completo
  + No, en la búsqueda de un árbol. Incluso con un estado de espacios finito, puede entrar en un bucle infinito

**Hill climbing V1**

* No garantiza que se encuentre una solucion aunque exista
* Algoritmo
  + Construir una lista con el nodo raíz como único elemento
  + Hasta que la lista este vacía o el primer (y unico) elemento sea el elemento objetivo
    - Eliminar el primer elemento de la lista (Nodo padre) y ordenar sus hijos(si los hubiera) estimando cual de ellos esta mas cerca de la solución
    - Si el primer nodo hijo x1,de los ordenados esta MAS CERCA de la solucion de lo que estaba el nodo padre,es decir h(padre)>h(x1) añadirlo al principio de la lista (el único de la lista) si no acabar
  + Si ha encontrado un nodo objetivo,anunciar éxito,si no fallo

**Hill climbing V2**

* No garantiza que se encuentre una solución, aunque exista
* Algortimo
  + Construir una lista con el nodo raíz como único elemento
  + Hasta que la lista esta vacía o el primer (y único) elemento de la lista sea el elemento objetivo
    - Eliminar el primer elemento de la lista (nodo padre) y ordenar sus hijos (si los hubiera) estimando cual de ellos esta mas cerca de la solución
    - Añadir el primer hijo de los ordenados x1,al principio de la lista (el único de la lista).Elimina la restricción de mejorar el heurístico
  + Si ha encontrado un nodo objetivo, anunciar éxito, si no fallo

**Best first**

* Construir una lista con el nodo raíz como único elemento
* Hasta que la lista esta vacía o el primer elemento de la lista sea el elemento objetivo
  + Eliminar el primer elemento de la lista, añadir los hijos de este elemento (Si los hubiera) y ordenar la lista estimando lo que falta hasta la solución
* Si se ha encontrado el nodo objetivo, anunciar éxito, si no, fallo
* Problema
  + Problemático para problemas con ramificación alta

**Beam search**

* Alternativa entre hill climbing y best first
* Mantiene las X mejores alternativas
* Algoritmo
  + Determinar X como el número máximo de nodos por nivel a expandir, Lista\_X=(nodo\_raiz);Lista\_hijos=()
  + Hasta que la Lista\_X este vacia o Lista\_X contenga un nodo objetivo
    - Si Lista\_X contiene una solución
      * Salir del bucle anunciando exito para el primer nodo solución
    - Si no
      * Eliminar todos los elementos de Lista\_X y añadir sus hijos (Si los hubiera) a Lista\_hijos, ordenados crecientemente según h
    - Meter en Lista\_X los X primeros nodos de Lista\_hijos
  + Si Lista\_X esta vacia,anunciar fracaso

**Búsqueda A\* (A\* search)**

* **Heurístico** h(e) Estimación del coste de lo que falta para llegar desde el estado e hasta el estado solución
* El termino h\*(e) se define como el coste real que existe entre un estado e y un estado solución
* El termino g() coste real del camino que lleva del estado inicial al estado e
* Siendo z el estado final del problema
  + f(z)=g(z)+h(z) representa el coste estimado del camino que lleva hasta dicha solución
  + f\*(z)=g(z)+h\*(z) es el coste real del camino a la solución, desdé el estado inicial hasta el estado final

**Ramificar y acotar v1**

* Algoritmo
  + Construir una lista de caminos parciales con el nodo raíz (coste acumulado = g = 0)
  + Hasta que la lista este vacia o el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino
    - Eliminar primer camino de la lista
    - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el ultimo nodo de ese camino
    - Añadir esos nuevos caminos a la lista junto con su coste
    - Ordenar la lista de menor a mayor considerando la función del coste de ese momento
  + Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito si no fallo

**Ramificar y acotar v2**

* Algoritmo
  + Construir una lista de caminos parciales con el nodo raíz (f =g+h;g coste acumulado, h heurístico hasta la solución)
  + Hasta que la lista esta vacia o el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino
    - Eliminar primer camino de la lista
    - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el ultimo nodo de ese camino
    - Añadir esos nuevos caminos a la lista junto con su coste total (f)
    - Ordenar la lista de mayor a menor considerando la función del coste en ese momento
  + Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito, si no fallo

**Ramificar y acotar v3**

* Algoritmo
  + Construir una lista de caminos parciales con el nodo raíz (coste acumulado = g = 0)
  + Hasta que la lista este vacía o el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino
    - Eliminar primer camino de la lista
    - Formar nuevos caminos apartir del eliminado añadiendo tantos caminos como hijos tenga el ultimo nodo de ese camino
    - Añadir esos nuevos caminos a la lista junto a con su coste
    - Ordenar la lista de menor a mayor considerando la funcón del coste en ese momento
    - Si hay caminos repetidos borrar todos ellos,excepto el que alcanza el nodo con el costo mínimo
  + Si el primer camino de la lista encuentra el nodo objetivo,anunciar éxito,si no fallo

**Ramificar y acotar v4**

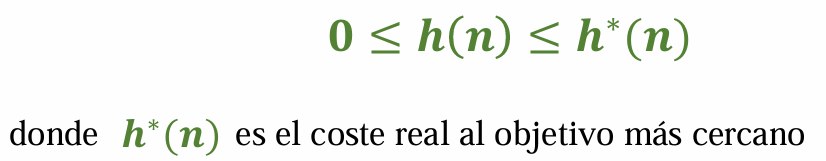
* Algoritmo
  + Construir una lista de caminos parciales con el nodo raíz (f=g+h;g=coste acumulado, h=coste heurístico hasta la solución)
  + Hasta que la lista este vacía o el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino
    - Eliminar primer camino de la lista
    - Formar nuevos caminos a partir del eliminado añadiendo tantos caminos como hijos tenga el ultimo nodo de ese camino
    - Añadir esos nuevos caminos a la lista junto con su coste total (f)
    - Ordenar la lista de menor a mayor considerando la función del coste en ese momento
    - Si hay CAMINIOS REPETIDOS borrar todos ellos excepto aquel que alcanza el nodo con el coste minimo

**Combinando UCS y Greedy**

* Coste uniforme (UCS) ordena por **el coste del camino**, o backward cost f(n)=g(n)
* Greedy ordena por la cercanía al objetivo (Best First) o forward cost f(n)=h(n)
* A\* Search ordena por la suma f(n)=g(n)+h(n)

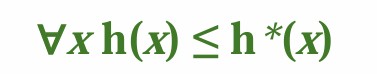
**Heurísticos admisibles**

* Heurísticos **inadmisibles**(pesimistas) rompen la optimalidad al atrapar buenos planes en el borde
* Heurísticos **admisibles** (optimistas) relentizan el proceso (planes malos) pero nunca superan el coste real
* Un heurístico **h** es **admisible (**optimista**)** si

****

**Accesibilidad y monotonía en algoritmos A\***

* Sea (e0 … ei) un camino del estado inicial a ei y sea ei+1 un estado al que se puede acceder desde ei.Abreviamos g(e0…ei) =g(ei)
* Decimos que la función de coste g es **monótona** si para todo ei, ei+1 se cumple que g(e0…ei)=g(ei)=<g(ei+1)
* Se dice que un heurístico h es **admisible** si

****

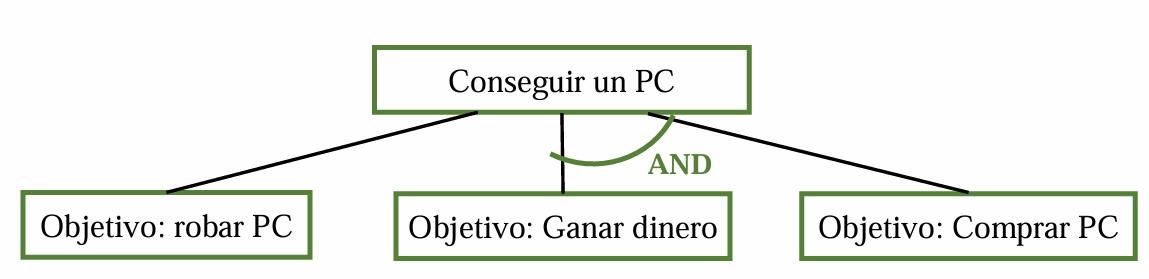
* **Para todo x ,h(x) es menor o igual a h\*(x)**
* Para asegurar la solución óptima se necesita que los heurísticos sean **admisibles** y el **coste monótona.** A estos algoritmos también se les llama algoritmos A\*

**Heurísticos más informados o dominantes**

* Un heurístico h admisible es **un heurístico optimista** si, siempre estima que el coste de llegar a las soluciones es menor o igual al coste real h\*
* Decimos que un heurístico h2 esta **mas informado** que **h1** o que h2 domina h1,si **h1<=h2**
* Todos los nodos **e**  con **f(e)=g(e)+h(e)<=f\*** se expanden en una búsqueda A\*.Por lo tanto, todos los nodos **e** con **h(e) <=f\*-g(e)** se expanden
* Si **h1** y **h2** son admisibles y **h2** esta informado, siempre se cumple:
  + h1(e)<=h2(e)<=h\*(e)=f\*-g(e).Por lo tanto,el **heurístico más informado expande menos nodos y la búsqueda es mas eficiente**

**Grafos and or**

* Para problemas descomponibles
* Hasta ahora todos los grafos eran tipo OR ,en cada instante un solo hijo

****

* **Características**
  + Util para descomponer el problema en subproblemas
    - La descomposición crea grafos AND
  + Un arco AND puede descomponerse en cualquier numero de sucesores y **todos deben resolverse para alcanzar la solución**
  + Cada rama AND puede llevar a tener **una solución diferente**
  + Algoritmos modificaciones del Ramificar y Acotar

**Algoritmos de búsqueda adversarial**

**Tipos de juegos**

* Queremos algoritmos para calcular una estrategia que recomiende un movimiento desde cada estado

**Juegos determinísticos**

* Muchas formalizaciones posibles, una es:
  + Estados **S** (inicio en s0)
  + Jugadores **P={1…N}**  Normalmente a turnos
  + Acciones **A**
  + Función de Transición **S x A -> S**
  + Test de terminación (Estado objetivo o final) **S->{t,f}**
  + **Función de utilidad** para estados terminales (borde)
* La solución de un jugador es una política  **S->A**

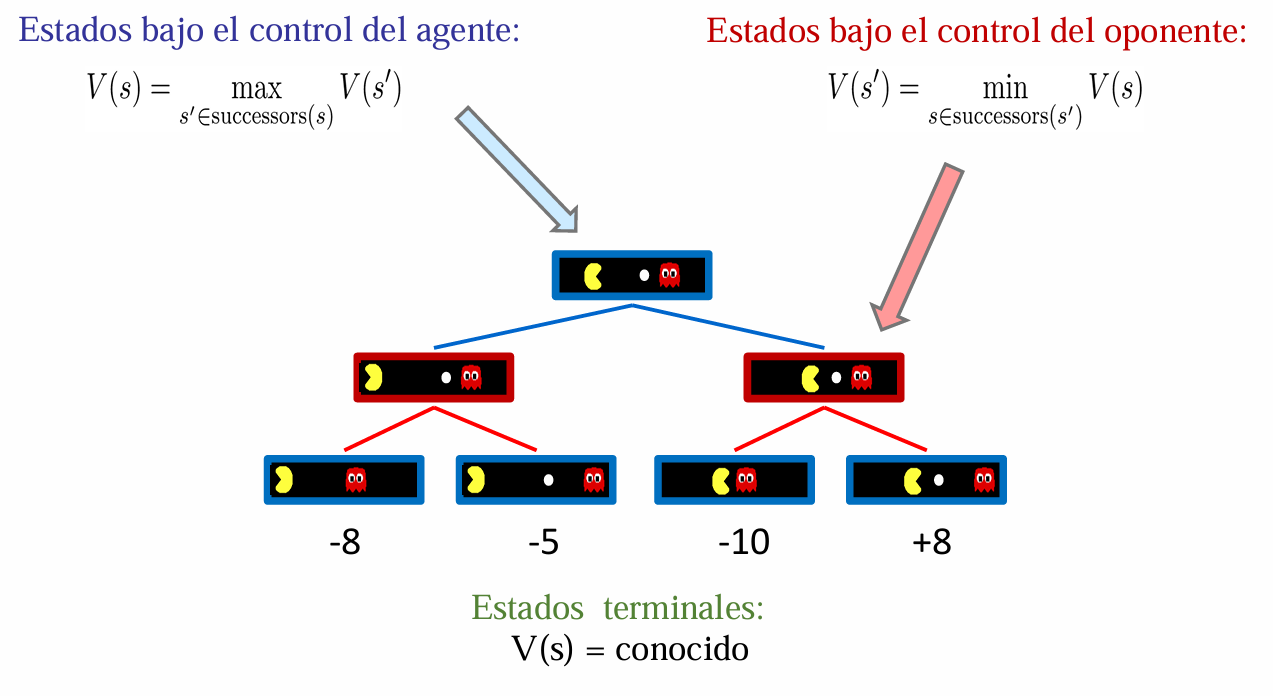
**Juegos de suma cero**

* Juegos de suma cero
  + Los agentes tienen utilidades opuestas (valores)
  + Podemos pensar en un único valor que uno maximiza y el otro minimiza
  + Adversarial, competición pura
* Juegos generales
  + Los agentes tienen utilidades independientes (valores)
  + Cooperación ,competición y más, todo es posible

**Valor de un estado**

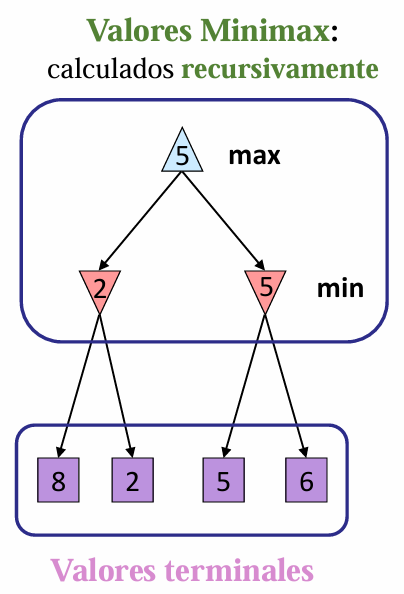
* El mejor resultado (utility) desde ese estado

**Valores minimax**

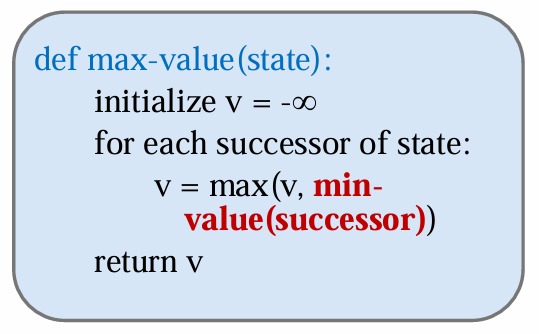
****

**Búsqueda adversarial (Minimax) 1 Adversario**

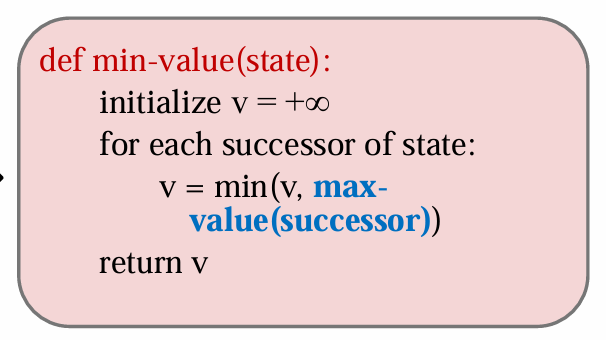
* Juegos determinísticos, de suma cero:
  + Un jugador maximiza el resultado
  + El otro minimiza el resultado
* Búsqueda **minimax**
  + Árbol de búsqueda en un espacio de estados
  + Los jugadores alternan turnos
  + Se calcula el valor minimax de cada estado:
    - El máximo resultado (utility) posible, contra un adversario racional (optimo)

****

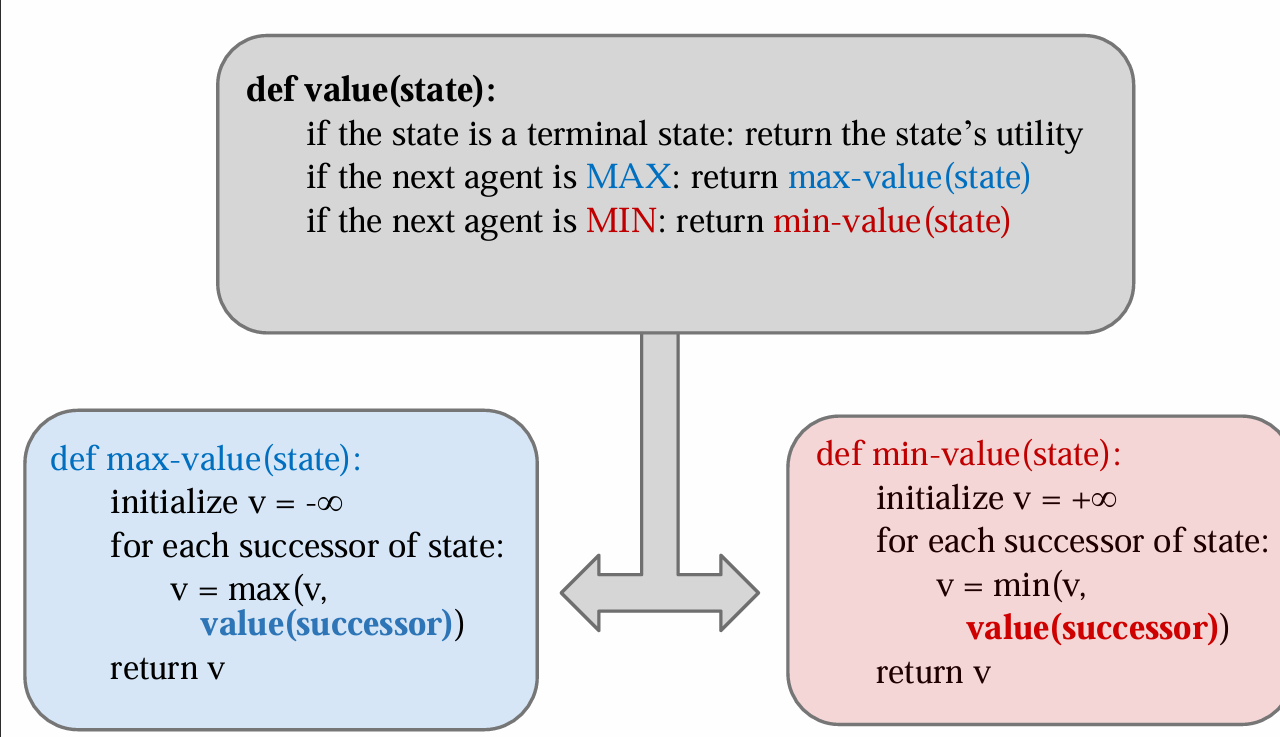
* + **Implementación**

****

* + El valor de v comienza en – infinito y va aumentando

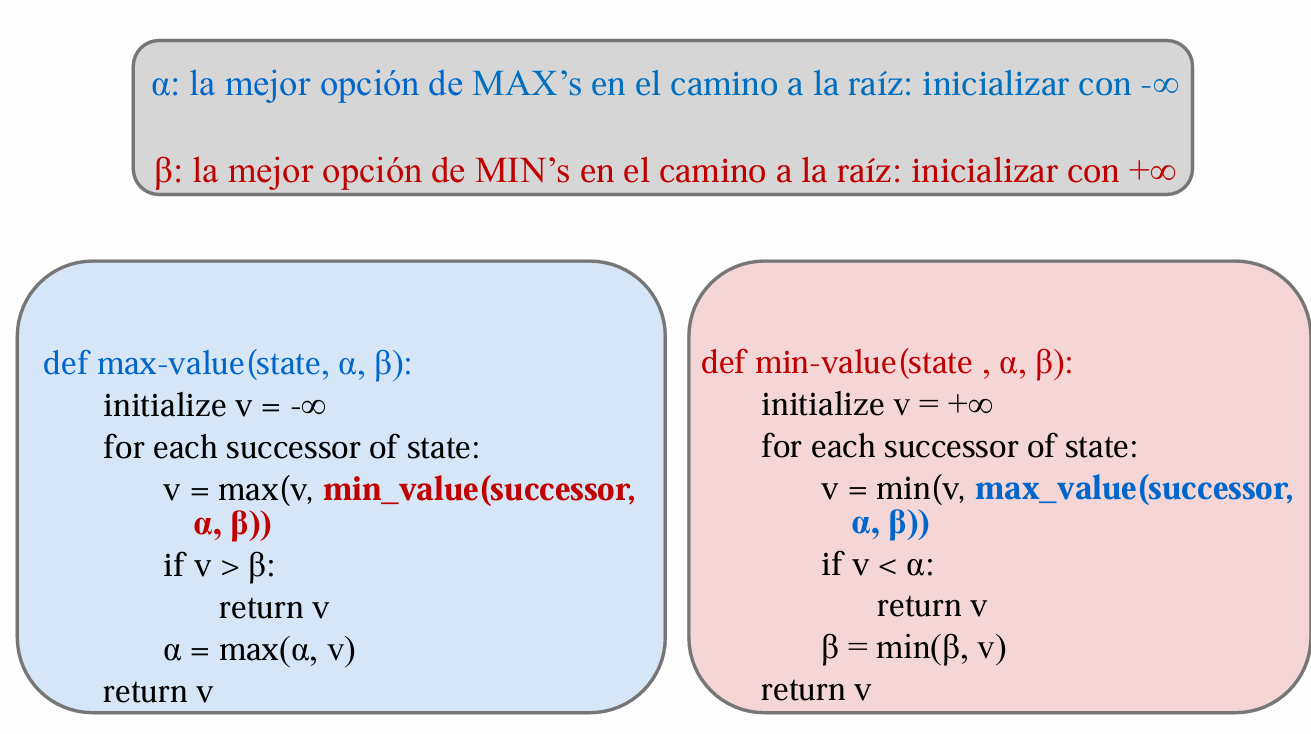


* + El valor de v empieza en +infinito y va disminuyendo



**Podado Alpha-Beta**

* Configuración general(versión MIN)
  + Estamos calculando un valor MIN en un nodo n
  + Estamos iterando sobre los hijos de n
  + Al estar minimizando el valor n ira reduciéndose
  + ¿Dónde se usa el valor de n? MAX
  + Sea a el mejor valor que MAX puede obtener a lo largo del camino actual desde la raíz
  + Si n es peor que a,MAX lo evitara, por ello podemos evitar el considerar los demás hijos de n
* Implementación

****

* **Alpha mayor o igual a beta podamos**

**Funciones de evaluación**

* Las funciones de evaluación puntúan estados no terminales, con búsqueda de profundidad limitada
* Función ideal Devuelve el valor real de mínima en esa posición hasta estado terminal
* En la practica normalmente suma lineal ponderada de características (features)

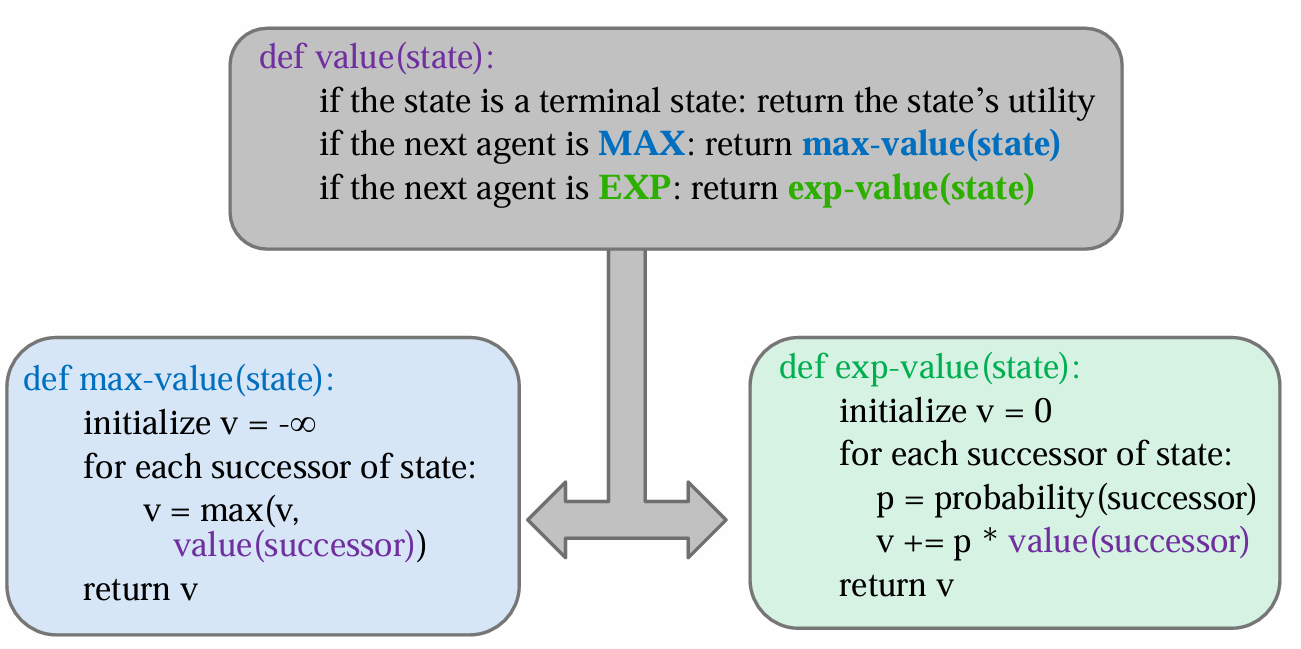
****

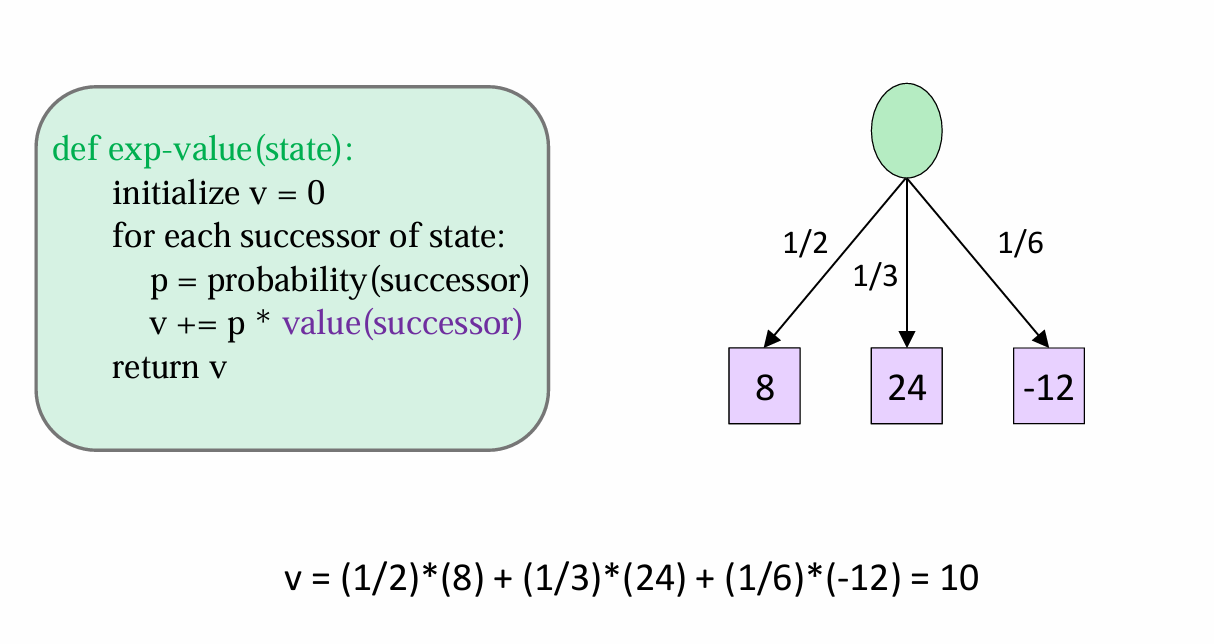
**La profundidad es importante**

* Las funciones de evaluación son siempre imperfectas
* Cuando mas profundamente en el árbol probemos la función de evaluación, es menos importante la calidad de la función de evaluación

**Búsqueda expectimax**

* Calcular la puntuación media con un juego optimo
  + Estados MAX como en búsqueda minimax
  + Los estados aleatorios son como los estados MIN pero el resultado es incierto
  + Se calcularan las utilidades esperadas

****

****

* + En la búsqueda expectimax, tenemos un modelo probabilístico de como el oponente (o entorno) se comportará en cualquier estado
    - El modelo podría ser una distribución uniforme (echar los dados)
    - El modelo podría ser sofisticado y requerir un montón de computación
    - Tendremos un nodo aleatorio por cada situación fuera de nuestro control: oponente o entorno
    - El modelo podría decir que acciones adversariales son probables