Busqueda

El primer paso para solucionar un problema es la **formulación del objetivo**, basado en la situación actual y la medida de rendimiento del agente.

Consideraremos un objetivo como un conjunto de estados del mundo, aquellos estados que satisfacen el objetivo.

La tarea del agente es encontrar qué secuencia de acciones permite obtener un estado objetivo, para lo cual necesitamos decidir que acciones y estados considerar lo que constituye la **formulación del problema**.

Consideremos un agente en la ciudad de Arad, Rumania que ha adoptado el objetivo de conducir a Bucarest y considera a dónde ir desde Arad. Existen tres carreteras desde Arad, una hacia Sibu, una a Timisoara y una a Zerind.

Dado que el agente no tiene conocimiento adicional, lo mejor que puede hacer es escoger al azar una de las acciones.

Un agente con distintas opciones inmediatas de valores conocidos puede decidir que hacer, examinando las diferentes secuencias posibles de acciones que le conduzcan a estados de valores conocidos, y entonces escoger la mejor secuencia.

Búsqueda

El agente supone que el entorno es **estático**, porque la formulación y búsqueda del problema se hace sin prestar atención a cualquier cambio que puede ocurrir en el entorno. El agente diseñado también supone que se conoce el estado inicial; conocerlo es fácil si el entorno es **observable** y supone además que el entorno es **determinista**.

Un problema puede definirse por cuatro componentes:

- El **estado inicial** en el que comienza el agente
- Una **descripción de las posibles acciones disponibles del agente**, es decir utilizar una función sucesor, dado un estado x la función SUCESOR-FN(x) devuelve un conjunto de pares ordenados (acción, sucesor) donde cada acción es una de las acciones legales en el estado x y cada sucesor es un estado que puede alcanzarse desde x, aplicando la acción. Implícitamente el estado inicial y la función sucesor definen el **espacio de estados** del problema, es decir el conjunto de todos los estados alcanzables desde el estado inicial. Un **camino** en un espacio de estados es una secuencia de estados conectados por una secuencia de acciones.
- El **test objetivo**, el cual determina si un estado es un estado objetivo.
- Una función **costo del camino** que asigna un costo numérico a cada camino, esta función costo debe reflejar una medida de rendimiento.

Búsqueda

Una **solución** de un problema es un camino desde el estado inicial a un estado objetivo. La calidad de la solución se mide por la función costo del camino, y una solución optima tiene el costo más pequeño del camino entre todas las soluciones.

Un **problema de juguete** se utiliza para ilustrar o ejercitar lo métodos de resolución de problemas, mientras que un **problema del mundo real** es aquel en el que la gente se preocupa por sus soluciones.

Problemas de juguete:

Mundo de la aspiradora:

Estados: el agente está en una de dos localizaciones, cada una con o sin suciedad. Hay $2 * 2^2 = 8$ posibles estados del mundo.

Estado inicial: cualquier estado

Función sucesor: los estados legales que resultan al intentar las tres acciones, izquierda, derecha, aspirar.

Test objetivo: si el cuadrado está limpio.

Costo del camino: cada costo individual es 1, así que el costo del camino es el número de pasos que lo compone.

Búsqueda

Problemas de juguete:

8-puzle consiste en un tablero de 3 x 3 con ocho fichas numeradas y un espacio en blanco.

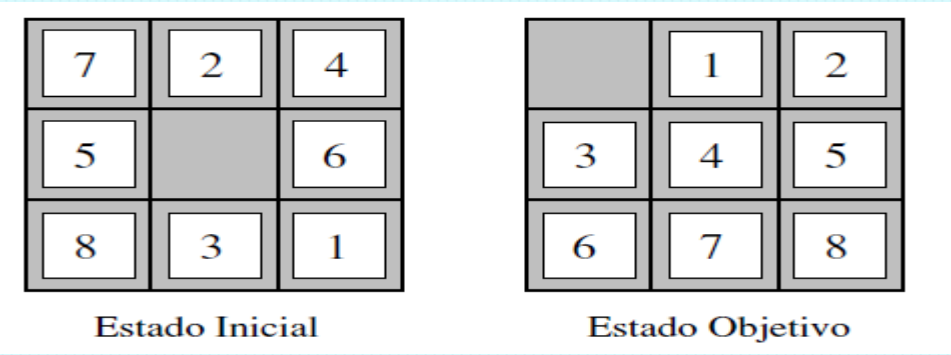
Estados: La descripción de un estado especifica la localización de cada una de las ochos fichas y el blanco en cada uno de los nueve cuadrados.

Estado inicial: cualquier estado

Función sucesor: los estados legales que resultan al intentar las cuatro acciones, mover el blanco a la izquierda, derecha, arriba y abajo.

Test objetivo: comprueba si el estado coincide con la configuración objetivo.

Costo del camino: cada costo individual es 1, así que el costo del camino es el número de pasos que lo compone.



Búsqueda

Problemas de juguete:

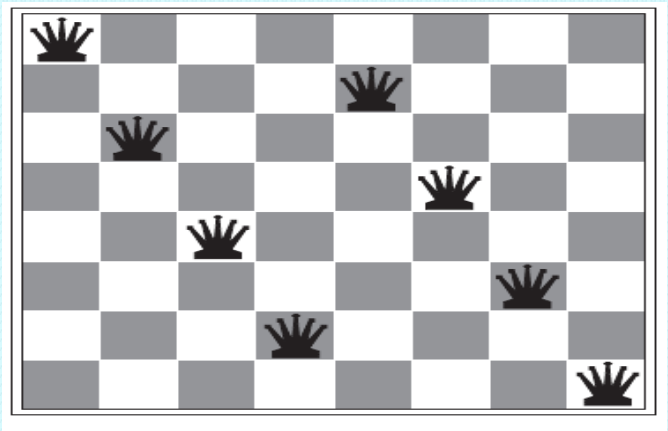
El problema de las 8-reinas en un tablero de ajedrez de manera que cada reina no ataque a ninguna otra, una reina ataca alguna pieza si está en la misma fila, columna o diagonal.

Estados: cualquier combinación de cero a ocho reinas en el tablero es un estado.

Estado inicial: ninguna reina sobre el tablero.

Función sucesor: añadir una reina a cualquier cuadrado vacío.

Test objetivo: ocho reinas sobre el tablero, ninguna es atacada.



Búsqueda

Una mejor formulación deberá prohibir colocar una reina en cualquier cuadrado que este atacado.

Estados: son estados la combinación n reinas $0 \leq n \leq 8$ una por columna desde la columna de más a la izquierda sin que una reina ataque a otra.

Estado inicial: ninguna reina sobre el tablero.

Función sucesor: añadir una reina a cualquier cuadrado en la columna más a la izquierda vacía tal que no sea atacada por cualquier otra reina.

Test objetivo: ocho reinas sobre el tablero, ninguna es atacada.

Búsqueda

Problemas del mundo real

Consideremos un ejemplo simplificado de un problema de viajes de líneas aéreas que especificamos como:

Estados: cada uno esta representado por una localización, por ejemplo un aeropuerto y la hora actual.

Estado inicial: especificado por el problema.

Función sucesor: devuelve los estados que resultan de tomar cualquier vuelo programado desde el aeropuerto actual a otro, que salgan a la hora actual más el tiempo de tránsito del aeropuerto.

Test objetivo: Tenemos nuestro destino para una cierta hora especificada?

Costo del camino: esto depende del costo en dinero, tiempo de espera, tiempo de vuelo, procedimientos de inmigración, calidad del asiento, hora, tipo de avión, etc.

Búsqueda

Para resolver el problema de búsqueda se utiliza un **árbol de búsqueda** explicito generado por el estado inicial y la **función sucesor**, definiendo así **el espacio de estados**. La raíz del árbol corresponde al estado inicial. **El primer paso es comprobar si éste es un estado objetivo**, Como no estamos en un estado objetivo, tenemos que considerar otros estados. Esto se hace expandiendo el estado actual y generar así un nuevo conjunto de estados. El estado a expandir está determinado por la estrategia de búsqueda.

Para el problema de búsqueda de una ruta hay una determinada cantidad de estados en el espacio de estado, pero **hay un número infinito de caminos** en este espacio de estados, por lo que el árbol de búsqueda tiene un número infinito de nodos.

Un nodo es una estructura de datos con cinco componentes:

- **Estado**: el estado, del espacio de estados, que corresponde con el nodo
- **Nodo padre**: el nodo en el árbol de búsqueda que ha generado ese nodo
- **Acción**: la acción que se aplicará al padre para generar el nodo.
- **Costo del camino**: el costo, g(n) de un camino desde el estado inicial al nodo, indicando por los punteros a los padres
- **Profundidad**: el número de pasos a lo largo del camino desde el estado inicial .

Búsqueda

Un nodo es una estructura de datos usada para representar **el árbol de búsqueda**. Un **estado** corresponde a una configuración del mundo Así los nodos están en caminos particulares, según lo definido por los punteros del nodo padre, mientras que los estados no lo están.

A la colección de nodos que se han generado pero todavía no se han expandido le llamaremos **frontera**, cada uno de sus elementos es una hoja.

La **estrategia de búsqueda** será una función que seleccione de este conjunto el siguiente nodo a expandir, asumiremos que la colección de nodos se implementa en una cola. Las operaciones en una cola son como siguen:

- **Hacer-cola(elemento, …)** crea una cola con el elemento dado
- **Vacia?(cola)** devuelve verdadero si no hay ningún elemento en la cola
- **Primero(cola)** devuelve el primer elemento de la cola
- **Borrar-Primero(cola)** devuelve **Primero(cola)** y lo borra de la cola
- **Inserta(elemento, cola)** inserta un elemento en la cola y devuelve la cola resultado.
- **Insertar-Todo(elementos, cola)** inserta un conjunto de elementos en la cola y devuelve el resultado.

Búsqueda

Medir el rendimiento de la resolución del problema. Hay cuatro formas:

- **Completitud**: está garantizado que el algoritmo encuentre una solución cuando esta exista?
- **Optimización**: encuentra la estrategia la solución optima, es decir la que maximiza el valor esperado de la medida de rendimiento.
- **Complejidad en el tiempo**: cuanto tarda en encontrar una solución?
- **Complejidad en el espacio**: cuánta memoria se necesita para el funcionamiento de la búsqueda ?

En IA el árbol está representado de forma implícita por el estado inicial y la función sucesor, y se expresa en términos de tres cantidades: **b** el factor de ramificación o el máximo número de sucesores de cualquier nodo, **d** la profundidad del nodo objetivo más superficial, y **m** la longitud máxima de cualquier camino en el espacio de estados. El tiempo se mide en términos de números de nodos generados durante la búsqueda, y el espacio en términos de máximo número de nodos que se almacena en memoria. Para valorar la eficacia de un algoritmo de búsqueda podemos utilizar el **costo total** que combina el costo de la búsqueda y el costo del camino solución encontrado.

Búsqueda no informada

En la **búsqueda no informada** no se tiene información adicional acerca de los estados más allá de la que proporciona la definición del problema.

Las estrategias que saben si un estado no objetivo es mas prometedor que otro se llama **búsqueda informada o búsqueda heurística**.

Búsqueda primero en anchura:

Es una estrategia sencilla en la que se expande primero el nodo raíz, a continuación se expanden todos los sucesores del nodo raíz, después sus sucesores, etc. En general se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo el próximo nivel.

La búsqueda primero en anchura se puede implementar comenzando con una frontera vacía que sea una cola primero en entrar primer en salir (FIFO). La cola FIFO pone todos los nuevos sucesores generados al final de la cola, lo que significa que los nodos más superficiales se expanden antes que los nodos más profundos.

Esta **búsqueda primero en anchura** es completa, si el nodo objetivo más superficial está en una cierta profundidad finita **d**, la búsqueda primero en anchura lo encontrará después de expandir todos los nodos más superficiales, con tal que el factor de ramficación **b** sea finito.

Búsqueda no informada Primero en anchura

La **búsqueda primero en anchura** es óptima si el costo del camino es una función no decreciente de la profundidad del nodo.

Consideremos un espacio de estados hipotético donde cada estado tiene **b** sucesores, la raíz del árbol de búsqueda genera **b** nodos en el primer nivel, cada uno de ellos genera **b** nodos más, teniendo un total de b^2 en el segundo nivel. Supongamos que la solución tiene profundidad **d**, en el peor caso expandiremos todo excepto el último nodo en el nivel **d** ya que el objetivo no se expande, generando $b^{d+1} – b$ nodos en el nivel d+1. El número total de nodos generados es:

$$b + b^2 + b^3 + \ldots. + b^d + (b^{d+1} – b) = O(b^{d+1})$$

La complejidad espacial es la misma que la complejidad en tiempo más un nodo para la raíz. En el siguiente cuadro se enumera el tiempo y la memoria requerida para una búsqueda primero en anchura con el factor de ramificación b = 10, 10,000 nodos/segundo, 1000 bytes/nodo.

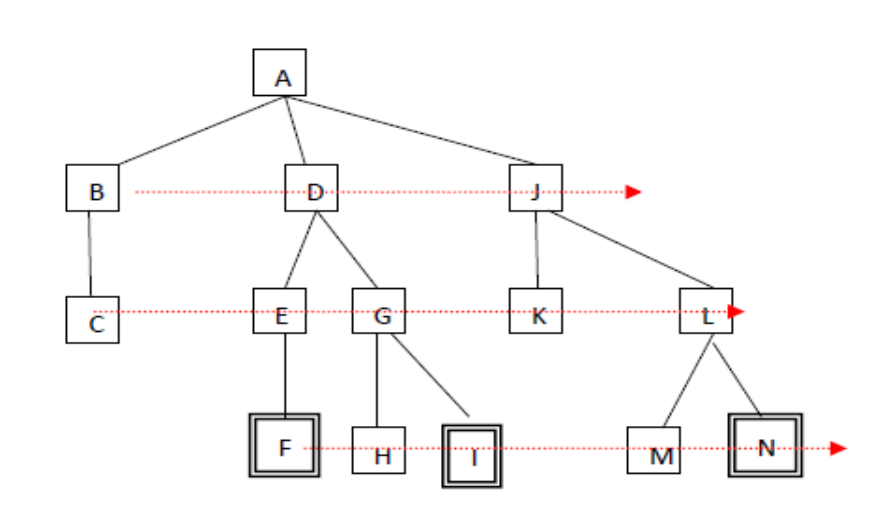
Profundidad	Nodos	Tiempo	Memoria
2	1.100	11 segundos	1 megabyte
4	111.100	11 segundos	106 megabytes
6	10^7	19 minutos	10 gigabytes
8	10^9	31 horas	1 terabytes
10	10^{11}	129 días	101 terabytes
12	10^{13}	35 años	10 petabytes
14	10^{15}	3.523 años	1 exabyte

De la figura anterior se desprende que los requisitos de memoria para la búsqueda primero en anchura son un problema más grande que el tiempo de ejecución.

Hace falta incluir en los algoritmos una estructura de datos llamada lista cerrada, que almacene cada nodo expandido, la frontera de nodos no expandidos se llama lista abierta.

Búsqueda no informada Primero en anchura

Los nodos meta son F, I y N.



Lista Abierta		Lista Cerrada
	A	-
m=A	B, D, J	A
m=B	D, J , C	A, B
m= D	J, C, E, G	A, B, D
m= J	C, E, G, K, L	A, B, D, J
m= C	E, G, K , L	A, B, D, J, C
m= E	G, k, L , F	A, B, D, J, C, E
m = G	K, L, F, H, I	A, B, D, J, C, E, G
m= K	L,F, H, I	A, B, D, J, C, E, G,K
m= L	F, H, I, M, N	A, B, D, J, C, E, G, K, L
m= F		
Llega a F nodo meta		

Búsqueda no informada Primero en profundidad

Búsqueda primero en profundidad:

En este tipo de búsqueda se expande el nodo más profundo en la frontera actual del árbol de búsqueda. Cuando esos nodos se expanden son quitados de la frontera, así entonces la búsqueda retrocede al siguiente nodo más superficial que todavía tenga sucesores inexplorados.

El algoritmo se implementa con una cola último en entrar primero en salir (LIFO).

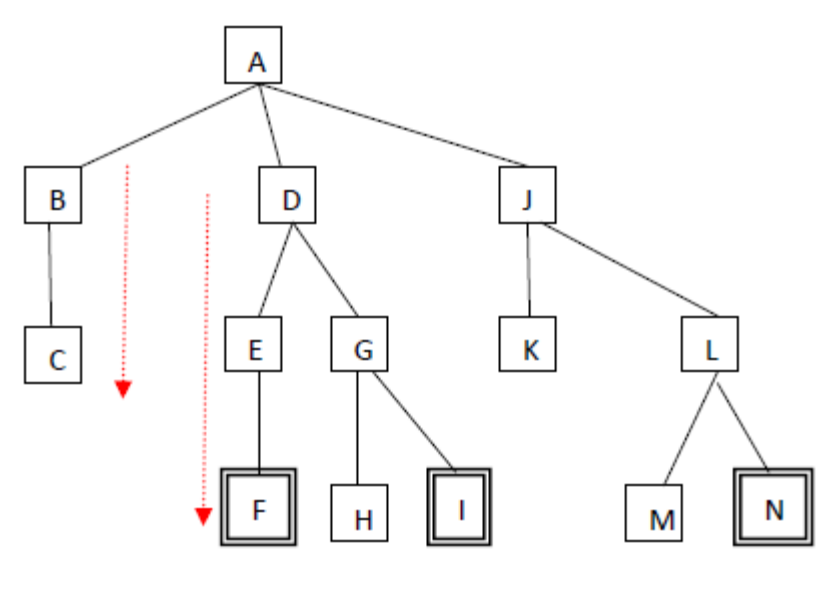
La búsqueda primero en profundidad tiene unos requisitos muy modestos de memoria. Necesita almacenar sólo un camino desde la raíz a un nodo hoja, junto con los nodos restantes no expandidos para cada nodo del camino. Una vez que un nodo se ha expandido se puede quitar de la memoria tan pronto como todos sus descendientes han sido explorados. Para un espacio de estados con factor de ramificación **b** y máxima profundidad **m**, la búsqueda primero en profundidad requiere almacenar solo **b*m + 1** nodos.

Una variante de la búsqueda primero en profundidad, llamada búsqueda hacia atrás, utiliza menos memoria. En la búsqueda hacia atrás, sólo se genera un sucesor a la vez, cada nodo parcialmente expandido recuerda que sucesor se expande a continuación. De esta manera se necesita **O(m)** memoria en lugar que el **O(b*m)** anterior.

El inconveniente de la búsqueda primero en profundidad es que puede hacer una elección equivocada y obtener un camino muy largo aún cuando una elección diferente llevaría a una solución cerca de la raíz del árbol de búsqueda.

Búsqueda no informada Primero en profundidad

Los nodos metas son F, I y N



Lista Abierta		Lista Cerrada
	A	-
m=A	B, D, J	A
m=B	C, D, J	A, B
m= C	D, J	A, B, C
m=D	E, G ,J	A, B, C, D
m= E	F , G, J	A, B, C , D, E
m= F		
Llega a F nodo meta		

Búsqueda no informada Primero en profundidad

Búsqueda de profundidad limitada:

Se puede aliviar el problema de árboles ilimitados aplicando la búsqueda primero en profundidad con un **límite de profundidad ℓ** predeterminado, es decir, **los nodos a profundidad ℓ** se tratan como si no tuvieran ningún sucesor. A este algoritmo se lo denomina búsqueda de profundidad limitada.

Búsqueda primero en profundidad con profundidad iterativa:

La búsqueda con profundidad iterativa es en general usada en combinación con la búsqueda primero en profundidad, la cual encuentra el mejor límite de profundidad. Esto se hace aumentando gradualmente el límite, primero 0, después 1, después 2, etc. Hasta que encontremos el objetivo. Esto ocurrirá cuando el límite de profundidad alcanza **d** profundidad del nodo objetivo.

La profundidad iterativa combina las ventajas de la búsqueda primero en profundidad y primero en anchura. En la búsqueda primero en profundidad su exigencia de memoria es O(b*d).

En una búsqueda de profundidad iterativa, los nodos sobre el nivel inferior (profundidad d) son generados una vez, los anteriores al nivel inferior dos veces, etc. hasta los hijos del nodo raíz que son generados **d** veces. El número total de nodos generados es:

$$N(BPI) = (d)\, b + (d-1)\, b^2 + \ldots\ldots + (1)\, b^d.$$

En general la profundidad iterativa es el método de búsqueda no informada preferido cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución.

Preguntas?