TÉCNICAS BÁSICAS DE BÚSQUEDA

Abraham Sánchez López
FCC/BUAP
Grupo MOVIS

MOVIS

Facultad de Ciencias de la Computación

Técnica básica de IA

INTELIGENCIA ARTIFICIAL

SOLUCION DE CIERTOS PROBLEMAS

UTILIZANDO LA BÚSQUEDA EN UN ESPACIO DE ESTADOS

Resolución de problemas

Formulación de problemas mediante el espacio de estados

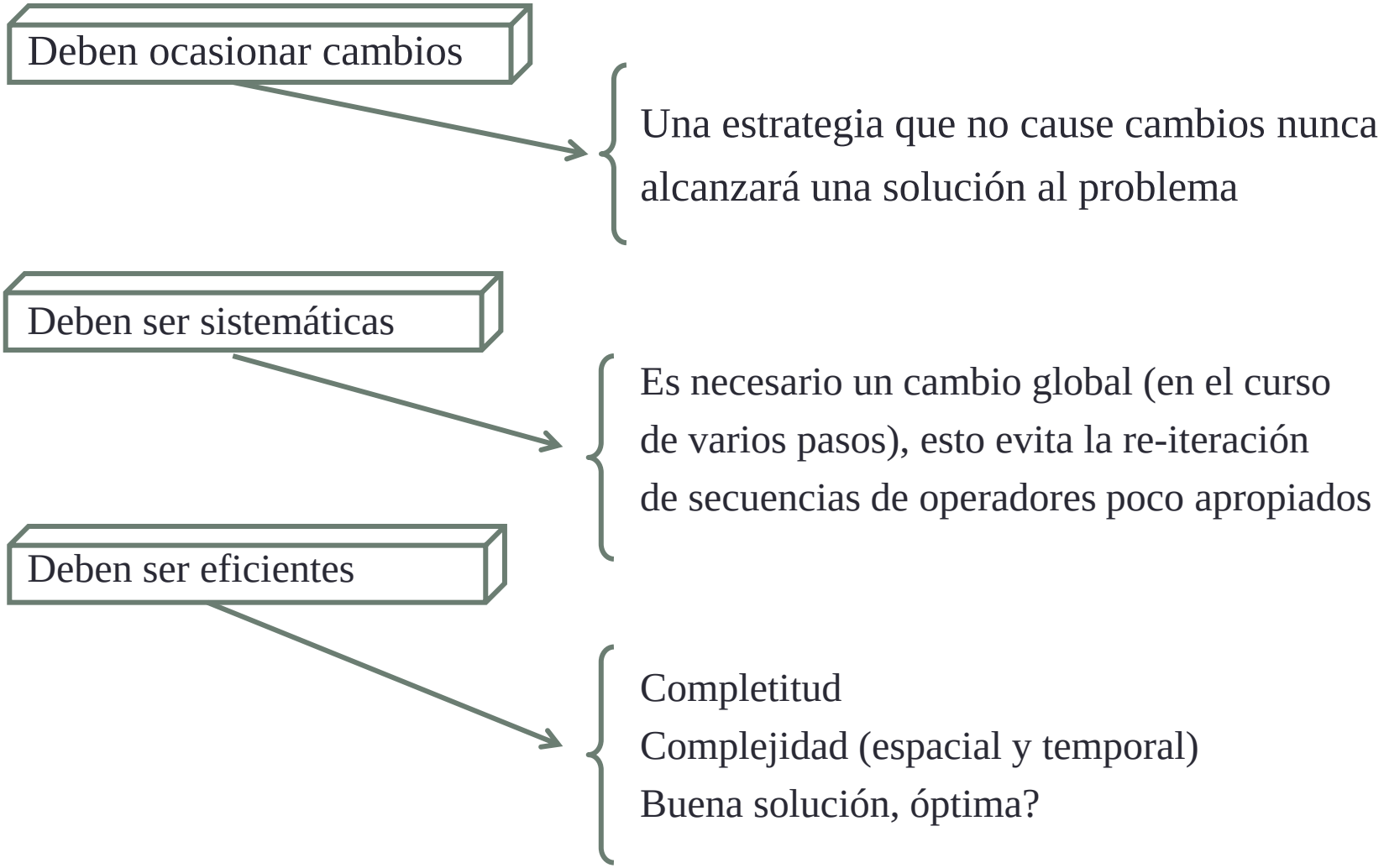
Representación formal

Aplicación de distintas técnicas de búsquedas

Secuencias de operadores

- Búsqueda, es el proceso de evaluar las distintas secuencias de acciones para encontrar las que llevan del estado inicial al estado meta.
- Es muy importante en la solución de problemas de IA, considerar si no existen técnicas más directas.
- Características de las estrategias de búsqueda?

Técnicas de búsqueda



Características del problema, I

- La búsqueda es un método muy general que se puede aplicar a una gran clase de problemas. Hay que considerar algunos aspectos de cada problema.

1. Puede descomponerse el problema en partes más simples independientes entre sí?

   Un ejemplo posible es la solución de una integral. Se pasa de un problema más complejo a la resolución de varios más simples.

   En el mundo de los bloques, los distintos objetivos influyen unos en otros y no resulta aplicable.

2. Pueden ignorarse o deshacerse pasos?

   Esto nos lleva a clasificar los problemas en tres tipos:

   Ignorables: En la demostración de teoremas matemáticos es posible. Si un paso no es deseable se ignora y se inicia nuevamente.

   Recuperables: En el 8-puzzle pueden deshacerse los pasos y recuperarse de los errores (hay que implementar una vuelta atrás).

Características del problema, II

No recuperables: En el ajedrez no pueden deshacerse las jugadas. Por lo tanto se necesita planificar con cuidado, lo que implica una estructura de control mucho más compleja

3. Es predecible el universo?

   Esto nos lleva a clasificar los problemas en dos tipos:

   Consecuencia cierta: Se conocen las consecuencias de cada operador con certeza. En el 8-puzzle es así, pueden planearse secuencias completas de movidas sabiendo siempre que ocurrirá.

   Consecuencia incierta: Planificación con incertidumbre (juegos con contrincantes, robots desplazándose, …).

4. Nos interesa una solución o la mejor de ellas?

   Esto nos lleva a clasificar los problemas en dos tipos:

   Problemas de algún camino.

   Problemas del mejor camino.

   En general es más costoso encontrar el mejor camino

Características del problema, III

5. La solución es un estado o una ruta?

   Un estado: clasificación, diagnóstico.

   Un camino: rutas, jarras de agua, etc. (hay que almacenar el camino seguido)

6. Cuál es el papel del conocimiento?

   Ajedrez: mucho conocimiento de control, poco del dominio.

   Interpretación de un artículo: mucho conocimiento del dominio.

7. Es necesaria la interacción con una persona/entorno?

   En ambientes estáticos y disponiendo inicialmente de la información: Se entrega la descripción del problema y el sistema devuelve la solución.

   Ejemplos: Demostración de teoremas, sistema de clasificación, …

   En ambientes dinámicos/procesos conversacionales: Existe comunicación periódica para proporcionar datos o recibir información.

   Ejemplo: robótica

Algoritmos de búsqueda, I

- La resolución de un problema planteado como un espacio de estados pasa por su exploración.
- Debemos partir del estado inicial marcado por el problema, evaluando cada posible paso y avanzando hasta encontrar un estado final.
- En el peor caso deberemos explorar todos los posibles caminos desde el estado inicial hasta poder llegar a un estado que cumpla las condiciones del estado final.
- Para poder implementar un algoritmo capaz de explorar el espacio de estados en busca de una solución primero debemos definir una representación adecuada de éste.
- Las estructuras de datos más adecuadas para representar el espacio de estados son los grafos y los árboles. En estas estructuras, cada nodo representará un estado del problema y los operadores de cambio de estado estarán representados por los arcos.
- La elección de un árbol o un grafo, determinará la posibilidad de que a un estado sólo se pueda llegar a través de un camino (árbol) o que haya diferentes caminos que puedan llevar al mismo estado (grafo).

Algoritmos de búsqueda, II

- Los grafos sobre los que trabajarán los algoritmos de resolución de problemas son diferentes de los algoritmos habituales para grafos.
- La característica principal es que el grafo se construirá a medida que se haga la exploración, dado que el tamaño que puede tener hace imposible que se pueda almacenar en memoria (el grafo puede ser incluso infinito).
- Esto hace que, por ejemplo, los algoritmos habituales de caminos mínimos en grafos no sirvan en este tipo de problemas.
- Nosotros nos vamos a centrar en los algoritmos que encuentran una solución, pero evidentemente extenderlo a todas las soluciones es bastante sencillo.
- Este sería un esquema a alto nivel de un algoritmo que encuentra una solución de un problema en el espacio de estados.
- El siguiente algoritmo general, clarifica estas ideas.

Algoritmo

funcion Busqueda_en_espacio_de_estados **retorna** solucion
    Seleccionar el primer estado como el estado actual
  mientras el estado actual **no** es el estado final **hacer**
    Generar y guardar sucesores del estado actual (expansion)
    Escoger el siguiente estado entre los pendientes (seleccion)
  fmientras
  retorna solucion
Ffuncion

- En este algoritmo tenemos varias decisiones que nos permitirán obtener diferentes variantes que se comportarán de distinta manera y que tendrán propiedades específicas.
- La primera decisión es el **orden de expansión**, o lo que es lo mismo, el orden en el que generamos los sucesores de un nodo. La segunda decisión es el **orden de selección**, o sea, el orden en el que decidimos explorar los nodos que aún no hemos visitado.

Algoritmos de búsqueda, III

- Entrando un poco más en los detalles de funcionamiento del algoritmo, podremos distinguir dos tipos de nodos, los nodos abiertos, que representarán a los estados generados pero aún no visitados y los nodos cerrados, que corresponderán a los estados visitados y que ya se han expandido.
- Nuestro algoritmo siempre tendrá una estructura que almacenará los nodos abiertos.
- Las diferentes políticas de inserción de esta estructura serán las que tengan una influencia determinante sobre el tipo de búsqueda que hará el algoritmo.
- Dado que estaremos explorando grafos, y que por lo tanto durante la exploración nos encontraremos con estados ya visitados, puede ser necesario tener una estructura para almacenar los nodos cerrados.
- Valdrá la pena si el número de nodos diferentes es pequeño respecto al número de caminos, pero puede ser prohibitivo para espacios de búsqueda muy grandes.

Algoritmo general de búsqueda, I

- A continuación tenemos un algoritmo algo más detallado que nos va a servir como esquema general para la mayoría de los algoritmos que iremos analizando en el curso:

```
Algoritmo Busqueda General
  Est_abiertos.insertar (Estado inicial)
  Actual= Est_abiertos. primero( )
    mientras no es_final?(Actual) y no Est_abiertos.vacia?( ) hacer
      Est_abiertos.borrar_primero( )
      Est_cerrados.insertar(Actual)
      Hijos=generar_sucesores(Actual)
      Hijos=tratar_repetidos(Hijos, Est_cerrados, Est_abiertos)
      Est_abiertos.insertar(Hijos)
      Actual=Est_abiertos.primero( )
  fmientras
fAlgoritmo
```

Algoritmo general de búsqueda, II

- Variando la estructura de abiertos variamos el comportamiento del algoritmo (orden de visita de los nodos).
- La función generar_sucesores seguirá el orden de generación de sucesores definido en el problema.
- El tratamiento de repetidos dependerá de cómo se visiten los nodos, en función de la estrategia de visitas puede ser innecesario.
- Para poder clasificar y analizar los algoritmos que vamos a tratar, escogeremos unas propiedades que nos permitirán caracterizarlos.
- Estas propiedades serán:

  Completitud: Si un algoritmo es completo tenemos la garantía de que hallará la solución, si no lo es, es posible que algoritmo no termineo que sea incapaz de encontrarla.

  Complejidad temporal: Nos indicará el costo temporal de la búsqueda en función del tamaño del problema, generalmente a partir del factor de ramificación y la profundidad a la que se encuentra la solución.

Algoritmo general de búsqueda, III

Complejidad espacial: Espacio requerido para almacenar los nodos pendientes de explorar. También se puede tener en cuenta el espacio para almacenar los nodos ya explorados si es necesario para el algoritmo.

Optimalidad: Si es capaz de encontrar la mejor solución según algún criterio de preferencia o costo.

- Atendiendo a estos criterios podemos clasificar los algoritmos principales que estudiaremos en:

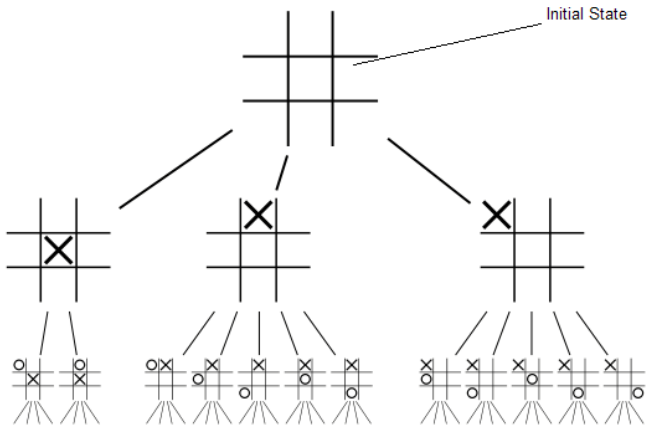
Algoritmos de búsqueda no informada (ciegos): Estos algoritmos no tienen en cuenta el costo de la solución durante la búsqueda. Su funcionamiento es sistemático, siguen un orden de visitas de nodos fijo, establecido por la estructura del espacio de búsqueda.

Los principales ejemplos de estos algoritmos son el de anchura, el de profundidad y el de profundidad iterativa y profundidad acotada.

Algoritmos de búsqueda heurística y búsqueda local: Estos algoritmos utilizan una estimación del costo o de la calidad de la solución para guiar la búsqueda, de manera que se basan en algún criterio heurístico dependiente del problema.

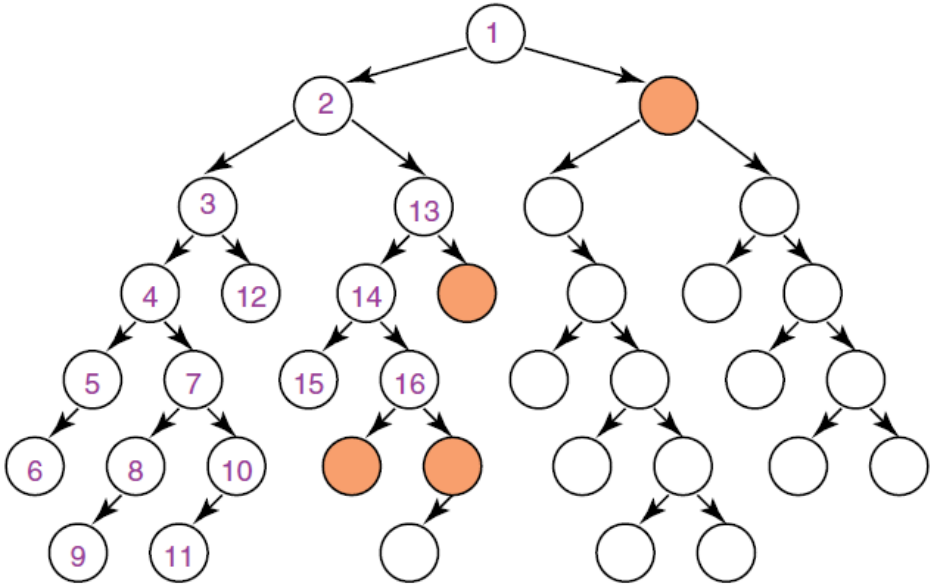
Algoritmo general de búsqueda, IV

- Estos algoritmos no son sistemáticos, de manera que el orden de exploración no lo determina la estructura del espacio de búsqueda sino el criterio heurístico.
- Algunos de ellos tampoco son exhaustivos y basan su menor complejidad computacional en ignorar parte del espacio de búsqueda.
- Existen bastantes algoritmos de este tipo con funcionamientos muy diferentes, no siempre garantizan el encontrar la solución óptima, ni tan siquiera el hallar una solución.
- Los principales ejemplos de estos algoritmos son A*, primero el mejor, IDA*, Branch & Bound, Hill-climbing, etc.



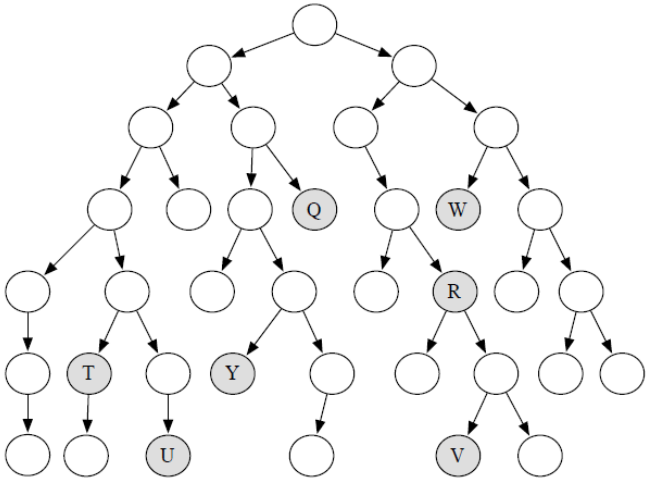
Búsqueda en profundidad, I

- La primera búsqueda en profundidad trata la frontera como una pila.
- Siempre selecciona uno de los últimos elementos agregados a la frontera.
- Si la lista de caminos en la frontera es [p_1, p_2, ...]
  - p_1 está seleccionado. Los caminos que se extienden de p_1 se agregan al frente de la pila (frente a p_2).
  - p_2 solo se selecciona cuando se han explorado todos los caminos de p_1.



Búsqueda en profundidad, II

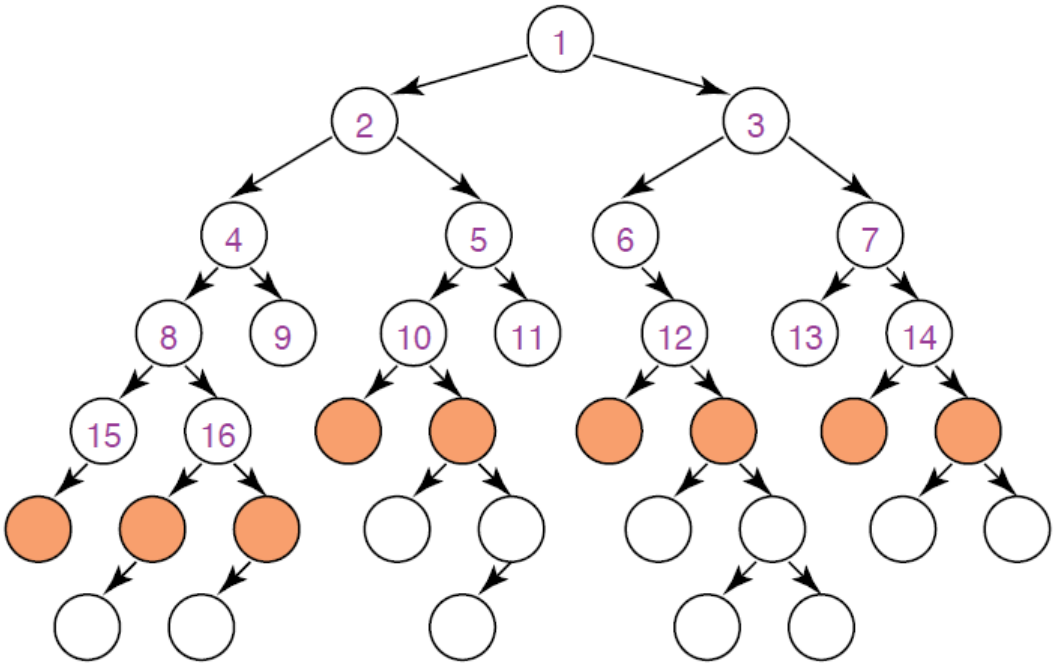
- ¿Qué objetivo sombreado buscará primero la búsqueda en profundidad?



- Complejidad
  - ¿La búsqueda en profundidad garantiza encontrar el camino con la menor cantidad de arcos?
  - ¿Qué sucede en los grafos infinitos o en los grafos con ciclos si hay una solución?
  - ¿Cuál es la complejidad del tiempo en función de la longitud del camino seleccionado?
  - ¿Cuál es la complejidad del espacio en función de la longitud del camino seleccionado?
  - ¿Cómo afecta el objetivo a la búsqueda?

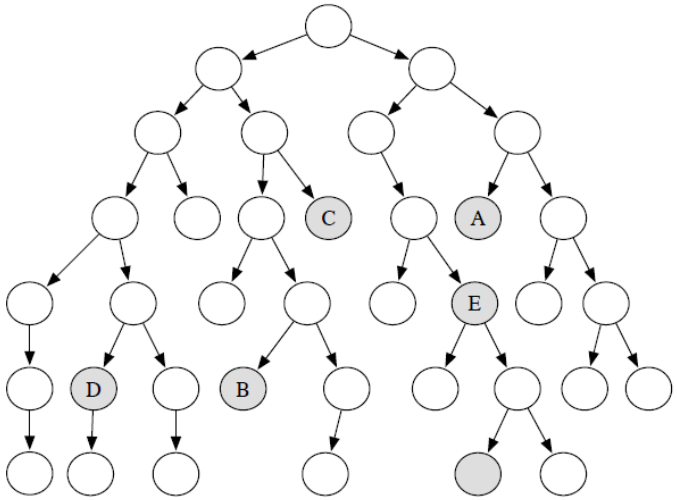
Búsqueda en anchura, I

- La búsqueda en anchura trata la frontera como una cola.
- Siempre selecciona uno de los primeros elementos agregados a la frontera.
- Si la lista de caminos en la frontera es [p_1, p_2, …, p_r]:
  - p_1 está seleccionado. Sus vecinos se agregan al final de la cola, después de p_r.
  - p_2 se selecciona a continuación.



Búsqueda en anchura, II

- ¿Qué objetivo sombreado buscará primero la búsqueda en anchura?



- Complejidad:
  - ¿La búsqueda en anchura garantiza encontrar el camino con la menor cantidad de arcos?
  - ¿Qué sucede en los grafos infinitos o en los grafos con ciclos si hay una solución?
  - ¿Cuál es la complejidad del tiempo en función de la longitud del camino seleccionado?
  - ¿Cuál es la complejidad del espacio en función de la longitud del camino seleccionado?
  - ¿Cómo afecta el objetivo a la búsqueda?

Ejemplo de búsqueda en profundidad

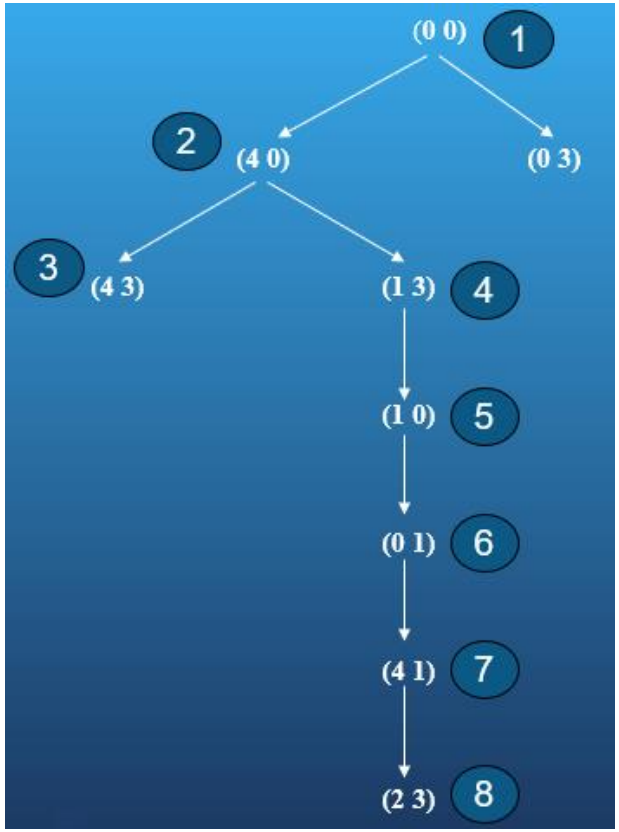


Tabla de búsqueda en profundidad

Nodo	Actual	Sucesores	Abiertos
			((0 0))
1	(0 0)	((4 0) (0 3))	((4 0) (0 3))
2	(4 0)	((4 3) (1 3))	((4 3) (1 3) (0 3))
3	(4 3)	( )	((1 3) (0 3))
4	(1 3)	((1 0))	((1 0) (0 3))
5	(1 0)	((0 1))	((0 1) (0 3))
6	(0 1)	((4 1))	((4 1) (0 3))
7	(4 1)	((2 3))	((2 3) (0 3))
8	(2 3)		

Ejemplo búsqueda en anchura

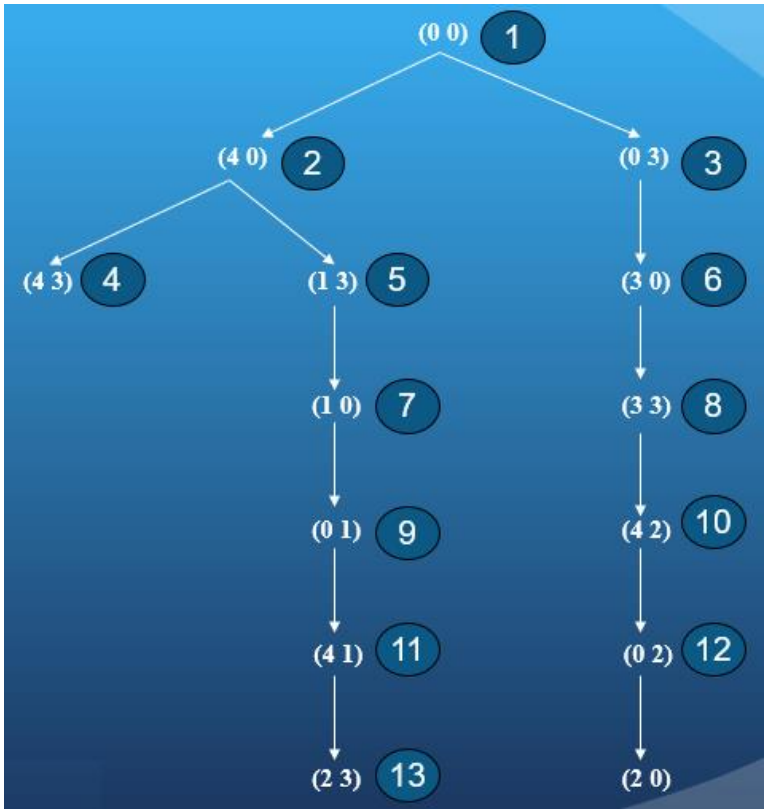


Tabla de búsqueda en anchura

Nodo	Actual	Sucesores	Abiertos
			((0 0))
1	(0 0)	((4 0) (0 3))	((4 0) (0 3))
2	(4 0)	((4 3) (1 3))	((0 3) (4 3) (1 3))
3	(0 3)	((3 0))	((4 3) (1 3) (3 0))
4	(4 3)	( )	((1 3) (3 0))
5	(1 3)	((1 0))	((3 0) (1 0))
6	(3 0)	((3 3))	((1 0) (3 3))
7	(1 0)	((0 1))	((3 3) (0 1))
8	(3 3)	((4 2))	((0 1) (4 2))
9	(0 1)	((4 1))	((4 2) (4 1))
10	(4 2)	((0 2))	((4 1) (0 2))
11	(4 1)	((2 3))	((0 2) (2 3))
12	(0 2)	((2 0))	((2 3) (2 0))
13	(2 3)		

Implementación en profundidad

FUNCION BUSQUEDA-EN-PROFUNDIDAD()

1. Hacer ABIERTOS la pila formada por el nodo inicial (es decir, el nodo cuyo estado es *ESTADO-INICIAL* y cuyo camino está vacío);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
      2.4.1 devolver el nodo ACTUAL y terminar.
      2.4.2 en caso contrario,
         2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
         2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al principio de ABIERTOS
3. Devolver FALLO.

Implementación en anchura

Función BUSQUEDA-EN-ANCHURA()

1. Hacer ABIERTOS la cola formada por el nodo inicial (es decir, el nodo cuyo estado es *ESTADO-INICIAL* y cuyo camino es vacío);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía.
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL EN CERRADOS
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
      2.4.1 devolver el nodo ACTUAL y terminar.
      2.4.2 en caso contrario,
         2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no está ni en ABIERTOS ni en CERRADOS
         2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al final de ABIERTOS
3. Devolver FALLO.

Implementación profundidad acotada

FUNCION BUSQUEDA-EN-PROFUNDIDAD-ACOTADA(COTA)

1. Hacer ABIERTOS la pila formada por el nodo inicial (es decir, el nodo cuyo estado es *ESTADO-INICIAL* y cuyo camino es vacío);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no este vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
      2.4.1 devolver el nodo ACTUAL y terminar.
      2.4.2 en caso contrario, si la profundidad del ACTUAL es menor que la cota,
         2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) cuyo estado no esta ni en ABIERTOS ni en CERRADOS
            2.4.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES al principio de ABIERTOS
3. Devolver FALLO.

Comparación de procedimientos

	Anchura	Profundidad	Profundidad acotada	Profundidad iterativa
Tiempo	$O(r^p)$	$O(r^m)$	$O(r^c)$	$O(r^p)$
Espacio	$O(r^p)$	$O(rm)$	$O(rc)$	$O(rp)$
Completa	*Sí*	*No*	*Sí*, $c \geq p$	*Sí*
Minimal	*Sí*	*No*	*No*	*Si*

- ***r***: factor de ramificación
- ***p***: profundidad de la solución
- ***m***: máxima profundidad de la búsqueda
- ***c***: cota de la profundidad