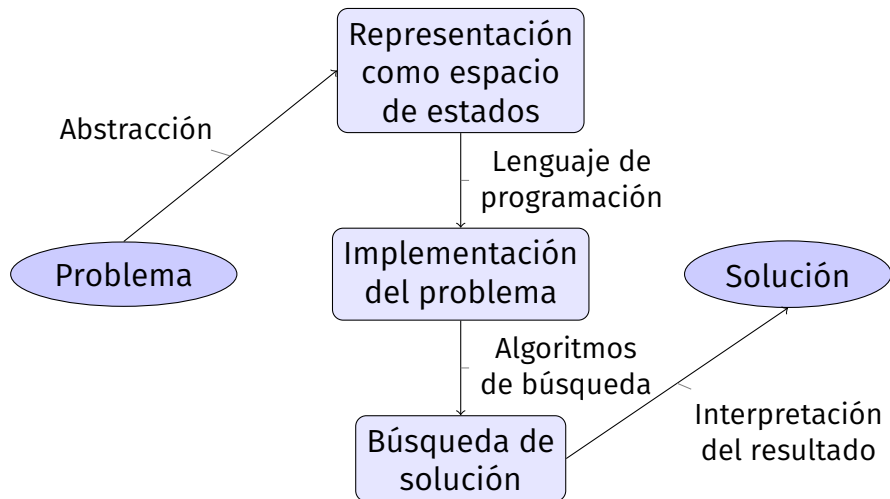


# Búsqueda en espacio de estados

Departamento de Ciencias de la Computación  
e Inteligencia Artificial  
Universidad de Sevilla

Inteligencia Artificial

# Metodología de resolución de problemas



# Rompecabezas de las *Torres de Hanoi*

El rompecabezas de las *Torres de Hanoi* consta de tres varillas verticales y un número de discos, que determinará la complejidad del problema, todos de distinto tamaño y apilados de mayor a menor radio en una de las varillas.



El objetivo del juego es pasar todos los discos de la varilla ocupada a una de las otras varillas vacantes, siguiendo tres simples reglas:

1. Se desplaza un disco cada vez.
2. Solo se pueden desplazar los discos de arriba de las varillas.
3. No se puede colocar un disco sobre otro más pequeño.

# Espacio de estados

Un **estado** es una descripción de una posible situación en que puede encontrarse el problema. Recopila toda la información relevante e ignora la irrelevante.

# Espacio de estados

Un **estado** es una descripción de una posible situación en que puede encontrarse el problema. Recopila toda la información relevante e ignora la irrelevante.

Información irrelevante para el problema de las *Torres de Hanoi*:

- ▶ De qué material están fabricados las varillas y los discos.
- ▶ Cuál es el color de cada disco.
- ▶ Etcétera.

# Espacio de estados

Un **estado** es una descripción de una posible situación en que puede encontrarse el problema. Recopila toda la información relevante e ignora la irrelevante.

Información irrelevante para el problema de las *Torres de Hanoi*:

- ▶ De qué material están fabricados las varillas y los discos.
- ▶ Cuál es el color de cada disco.
- ▶ Etcétera.

Información relevante para el problema de las *Torres de Hanoi*:

- ▶ Qué discos están colocados en cada varilla.
- ▶ En qué orden están colocados esos discos.

# Espacio de estados

Un **estado** es una descripción de una posible situación en que puede encontrarse el problema. Recopila toda la información relevante e ignora la irrelevante.

Información irrelevante para el problema de las *Torres de Hanoi*:

- ▶ De qué material están fabricados las varillas y los discos.
- ▶ Cuál es el color de cada disco.
- ▶ Etcétera.

Información relevante para el problema de las *Torres de Hanoi*:

- ▶ Tamaño (relativo) de los discos colocados en cada varilla.

# Espacio de estados

Un **estado** es una descripción de una posible situación en que puede encontrarse el problema. Recopila toda la información relevante e ignora la irrelevante.

Información irrelevante para el problema de las *Torres de Hanoi*:

- ▶ De qué material están fabricados las varillas y los discos.
- ▶ Cuál es el color de cada disco.
- ▶ Etcétera.

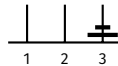
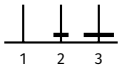
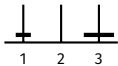
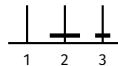
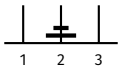
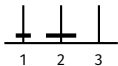
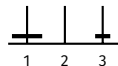
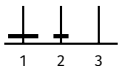
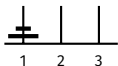
Información relevante para el problema de las *Torres de Hanoi*:

- ▶ Tamaño (relativo) de los discos colocados en cada varilla.

El conjunto de todos los posibles estados en que puede encontrarse un problema es el **espacio de estados** del problema.



# Espacio de estados de las *Torres de Hanoi*



# Grafo de un problema de espacio de estados

Mediante la aplicación de una **acción** u operador, el problema pasa a encontrarse en un nuevo estado.

El conjunto de posibles acciones que se pueden usar en un problema es fijo y finito.

Debe determinarse la **aplicabilidad** de las acciones, ya que en general no será posible aplicar cualquier acción a cualquier estado.

Un problema de espacio de estados se puede representar gráficamente como un grafo dirigido en el que los vértices son los estados y hay un arco de un estado a otro si el problema puede pasar del primero al segundo mediante la aplicación de una acción.

# Acciones para las *Torres de Hanoi*

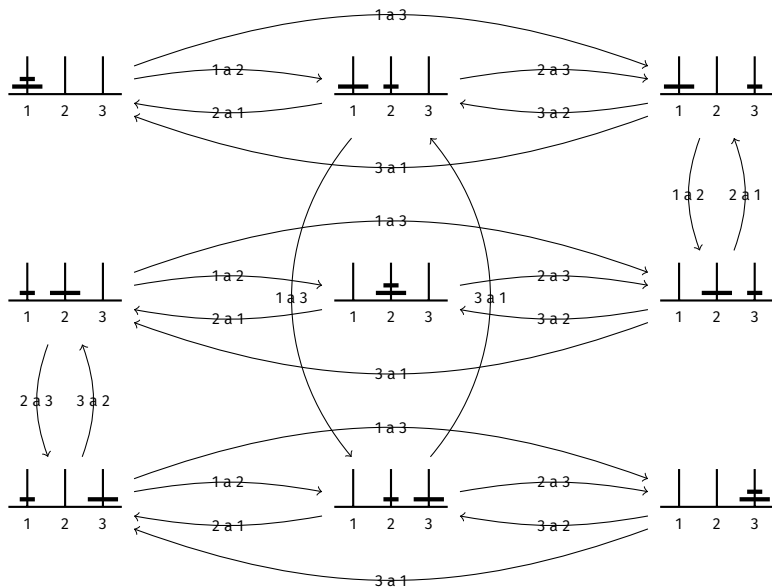
- ▶ Mover el disco de tamaño  $i$  a la varilla  $j$ .
- ▶ Condiciones de aplicabilidad:
  - ▶ el disco de tamaño  $i$  se encuentra en lo alto de una varilla distinta de la  $j$ .
  - ▶ o no hay discos en la varilla  $j$  o el disco que se encuentra en lo alto de ella tiene un tamaño mayor que  $i$ .
- ▶ Número total de acciones:  $3n$  acciones, si el número de discos es  $n$ .

# Acciones para las *Torres de Hanoi*

En general, es conveniente que el número de acciones sea lo menor posible.

- ▶ Mover un disco de la varilla  $i$  a la varilla  $j$ .
- ▶ Condiciones de aplicabilidad:
  - ▶ en la varilla  $i$  hay al menos un disco.
  - ▶ o no hay discos en la varilla  $j$  o el disco que se encuentra en lo alto de ella tiene un tamaño mayor que el que se encuentra en lo alto de la varilla  $i$ .
- ▶ Número total de acciones: 6 acciones, independientemente del número de discos.

# Grafo para las Torres de Hanoi

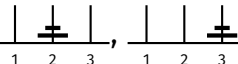


# Solución de un problema de espacio estados

Una **solución** de un problema de espacio de estados es una secuencia de acciones que permita llevar el problema de la situación inicial en la que se encuentra, su **estado inicial**, a un estado deseado, un **estado final**.

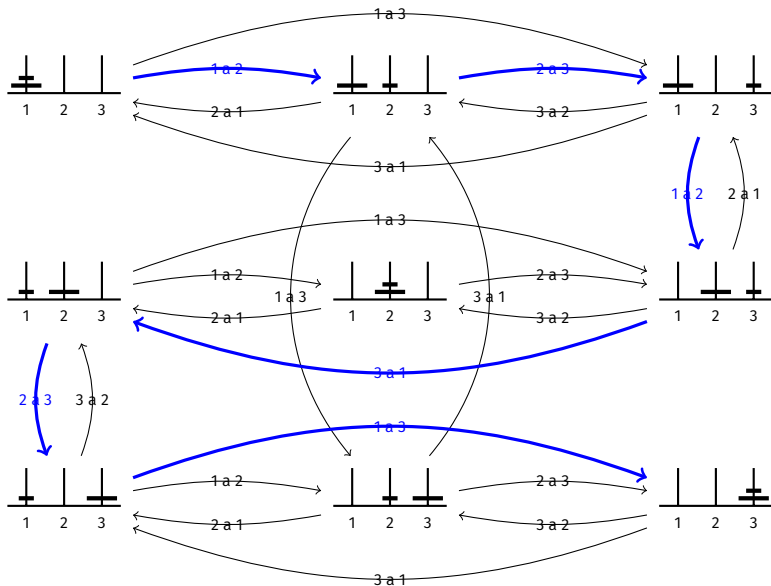
Estado inicial de las *Torres de Hanoi*: 

Posibles estados finales de las *Torres de Hanoi*.

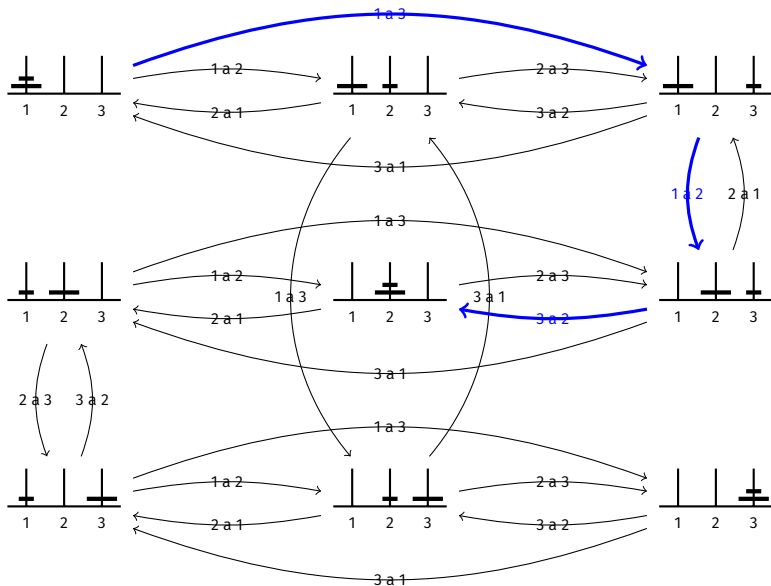
► Descripción enumerativa: 

► Descripción declarativa: todos los discos deben estar en la segunda o en la tercera varilla.

# Posibles soluciones para las Torres Hanoi



# Posibles soluciones para las Torres Hanoi





# Problema de espacio de estados

En resumen, para representar un problema como un problema de espacio de estados es necesario:

- ▶ Describir cuáles son todos los posibles **estados** del problema.
- ▶ Fijar las **acciones** que se pueden aplicar para que el problema pase de un estado a otro, determinando sus condiciones de aplicabilidad (dado un estado  $s$  y una acción  $a$  aplicable a ese estado, notaremos  $a(s)$  el estado resultante de aplicar  $a$  a  $s$ ).
- ▶ Indicar el **estado inicial** en el que se encuentra el problema.
- ▶ Enumerar o declarar las propiedades que deben cumplir los **estados finales** en los que deseamos que se encuentre el problema.

Se puede entonces aplicar **procedimientos generales** de búsqueda de soluciones que son **independientes** del problema.

# Búsqueda de una solución de un problema de espacio de estados

La búsqueda de una solución de un problema de espacio de estados se reduce a una exploración del grafo del espacio de estados:

- ▶ La búsqueda comienza a partir de un único estado inicial.
- ▶ Las acciones se aplican de manera sistemática a estados ya considerados para analizar nuevos estados.
- ▶ La búsqueda termina al analizar un estado que satisface el objetivo (solución encontrada) o al explorar el grafo al completo (solución no encontrada).

Tipos de soluciones a buscar: una solución cualquiera, todas las soluciones, la solución más corta, la solución menos costosa, etc.

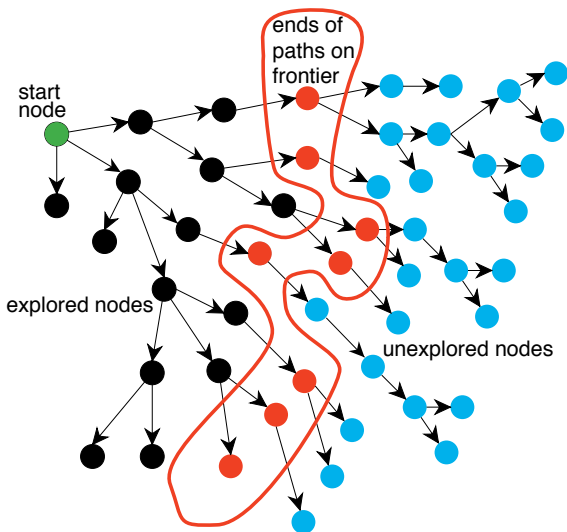
# Árbol de búsqueda

La inviabilidad de construir el grafo del espacio de estados para problemas de cierta complejidad (con un gran número de posibles estados), hace necesario llevar a cabo de manera dinámica la búsqueda de una solución del problema, mediante un **árbol de búsqueda**.

Un árbol de búsqueda:

- ▶ Se construye de manera incremental.
- ▶ Dispone de un conjunto de nodos analizados (**explorados** o cerrados) y de nodos por analizar (**frontera** o abiertos).
- ▶ Controla la repetición de nodos, para evitar bucles infinitos.

# Descripción gráfica de un árbol de búsqueda



# Nodos de un árbol de búsqueda

Información necesaria para un **nodo**:

- ▶ Estado.
- ▶ Nodo padre (ninguno para el nodo inicial).
- ▶ Acción (ninguna para el nodo inicial).

La información guardada para cada nodo permite deducir la secuencia de acciones que llevan del estado inicial al estado de ese nodo.

**Expansión** de un nodo  $n$  con estado  $s$ :

- ▶ Colección de nodos hijos (**sucesores** de  $n$ ).
- ▶ **Nodos hijos**: para cada acción  $a$  aplicable a  $s$ , nodo con estado  $a(s)$ , nodo padre  $n$  y acción  $a$ .

# Esquema general de búsqueda I

**función** búsqueda-general (*problema*)

**devuelve** *solución* o *fallo*

**inicio función**

inicializar la *frontera* con el nodo inicial

inicializar los nodos *explorados* al conjunto vacío

**repetir**

**si** la *frontera* está vacía **entonces**

**devolver** *fallo*

**fin si**

seleccionar y sacar un *nodo* de la *frontera*

**si** *nodo* contiene un estado final **entonces**

**devolver** secuencia de acciones desde el nodo inicial hasta *nodo*

**fin si**

# Esquema general de búsqueda II

```
añadir nodo al conjunto de explorados  
para cada acción aplicable al estado de nodo hacer  
    si el nodo hijo correspondiente a acción no está  
        ni en frontera ni en explorados  
        entonces  
            añadir el nodo hijo a frontera  
        fin si  
    fin para  
fin repetir  
fin función
```

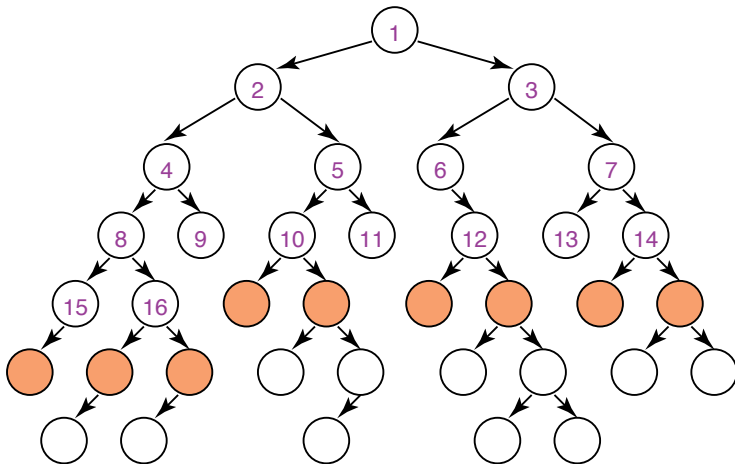
# Estrategias de búsquedas

Cada posible forma de seleccionar los nodos de la frontera determina una **estrategia de búsqueda**.

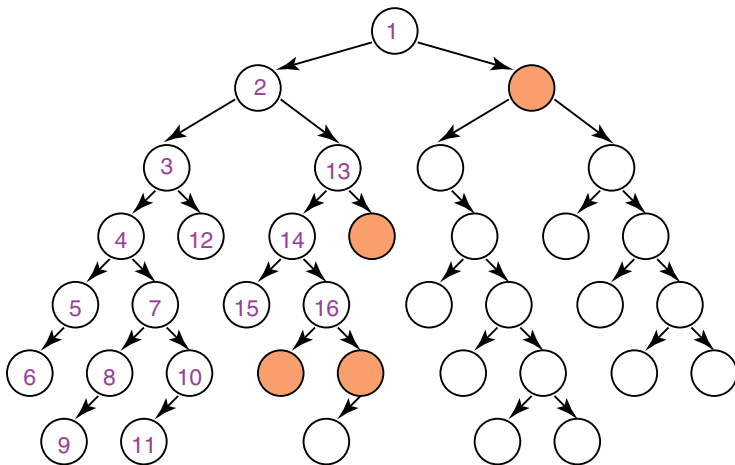
- ▶ Búsqueda no informada: independiente del problema.
  - ▶ Búsqueda en anchura: *frontera* es una cola FIFO (el primero que entra es el primero que sale).
  - ▶ Búsqueda en profundidad: *frontera* es una pila LIFO (el último que entra es el primero que sale).
  - ▶ Búsqueda en profundidad iterativa: búsqueda en profundidad reiterada.
- ▶ Búsqueda informada: usa información específica del problema.
  - ▶ Búsqueda primero el mejor: se selecciona el nodo que se *estima* mejor candidato.
  - ▶ Búsqueda  $A^*$ : se selecciona el nodo que se *estima* más cerca de una solución óptima.



## Descripción gráfica de la búsqueda en anchura



# Descripción gráfica de la búsqueda en profundidad



# Complejidad de los algoritmos de búsqueda

Complejidad en tiempo: número de nodos generados.  
Complejidad en espacio: tamaño máximo de *frontera* y *explorados*.

# Complejidad de los algoritmos de búsqueda

Complejidad en tiempo: número de nodos generados.  
Complejidad en espacio: tamaño máximo de *frontera* y *explorados*.

Parámetros de complejidad:

- ▶  $r$ : factor de ramificación.
- ▶  $p$ : profundidad mínima de una solución.
- ▶  $m$ : profundidad del árbol de búsqueda.

# Complejidad de los algoritmos de búsqueda

Complejidad en tiempo: número de nodos generados.  
Complejidad en espacio: tamaño máximo de *frontera* y *explorados*.

Parámetros de complejidad:

- ▶  $r$ : factor de ramificación.
- ▶  $p$ : profundidad mínima de una solución.
- ▶  $m$ : profundidad del árbol de búsqueda.

Notación  $O$ : complejidad del peor caso.

# Complejidad de la búsqueda no informada

**Complejidad:** siempre se encuentra la solución, si existe.

**Minimalidad:** solución con número mínimo de acciones.

	Anchura	Profundidad
Completa	Sí	No en general Sí con número finito de estados
Minimal	Sí	No
Tiempo	$O(r^{p+1})$	$O(r^{m+1})$
Espacio	$O(r^{p+1})$	$O(r^{m+1})$

# Complejidad de la búsqueda no informada

**Complejidad:** siempre se encuentra la solución, si existe.

**Minimalidad:** solución con número mínimo de acciones.

	Anchura	Profundidad
Completa	Sí	No en general Sí con número finito de estados
Minimal	Sí	No
Tiempo	$O(r^{p+1})$	$O(r^{m+1})$
Espacio	$O(r^{p+1})$	$O(r^{m+1})$

Suponiendo ramificación 10,  $10^6$  nodos/seg y 1000 bytes/nodo

Prof.	Tiempo	Espacio	Prof.	Tiempo	Espacio
6	1 seg	1 GB	12	13 días	1 PB
8	2 min	103 GB	14	3.5 años	99 PB
10	3 horas	10 TB	16	350 años	10 EB

# Complejidad de la búsqueda no informada

**Complejidad:** siempre se encuentra la solución, si existe.

**Minimalidad:** solución con número mínimo de acciones.

	Anchura	Profundidad
Completa	Sí	No en general Sí con número finito de estados
Minimal	Sí	No
Tiempo	$O(r^{p+1})$	$O(r^{m+1})$
Espacio	$O(r^{p+1})$	$O(rm)$

Búsqueda en profundidad: para evitar bucles infinitos solo es necesario guardar los nodos de la rama que se está explorando.



# Búsqueda en profundidad iterativa I

**función** búsqueda-p-acotada (*problema, cota*)

**devuelve** *solución* o *fallo*

**inicio función**

inicializar la *frontera* como una pila con el nodo inicial

inicializar los nodos *explorados* al conjunto vacío

**repetir**

**si** la *frontera* está vacía **entonces**

**devolver** *fallo*

**fin si**

seleccionar y sacar el primer *nodo* de la *frontera*

**si** *nodo* contiene un estado final **entonces**

**devolver** secuencia de acciones desde el nodo inicial hasta *nodo*

**fin si**

# Búsqueda en profundidad iterativa II

```
añadir nodo al conjunto de explorados  
si profundidad de nodo es menor que cota entonces  
  para cada acción aplicable al estado de nodo hacer  
    si el nodo hijo correspondiente a acción no  
      está ni en frontera ni en explorados  
    entonces  
      añadir el nodo hijo a frontera  
    fin si  
  fin para  
fin si  
fin repetir  
fin función
```

# Búsqueda en profundidad iterativa III

```
función búsqueda-p-iterativa (problema)  
    devuelve solución o fallo  
inicio función  
    para cota desde 0 hasta  $\infty$  hacer  
        resultado  $\leftarrow$  búsqueda-p-acotada(problema, cota)  
        si resultado no es fallo entonces  
            devolver resultado  
        fin si  
    fin para  
fin función
```

# Evaluación heurística de un nodo

Estimación del nodo  $n$  de la frontera mejor candidato para ser analizado a continuación por un algoritmo de búsqueda:

**función de evaluación heurística**,  $f(n)$ .

Heurística de un nodo: valor numérico que puede depender de

- ▶ el nodo.
- ▶ el objetivo.
- ▶ la información recogida por la búsqueda hasta ahora.
- ▶ cualquier conocimiento extra acerca del dominio del problema.

Algoritmo de búsqueda primero el mejor:

- ▶ la frontera es una cola de prioridades ordenada por la heurística.
- ▶ un nodo a explorar solo se descarta si existe algún nodo en la frontera o en explorados con el mismo estado y con un valor heurístico menor o igual.

# Búsqueda primero el mejor I

**función** búsqueda-primero-mejor (*problema*)

**devuelve** *solución* o *fallo*

**inicio función**

inicializar la *frontera* como una cola de prioridades ordenada por heurística con el nodo inicial

inicializar los nodos *explorados* al conjunto vacío

**repetir**

**si** la *frontera* está vacía **entonces**

**devolver** *fallo*

**fin si**

seleccionar y sacar un *nodo* de la *frontera*

**si** *nodo* contiene un estado final **entonces**

**devolver** secuencia de acciones desde el nodo inicial hasta *nodo*

**fin si**

# Búsqueda primero el mejor II

añadir *nodo* al conjunto de *explorados*

**para cada** *acción* aplicable al estado de *nodo* **hacer**

**si** el nodo hijo correspondiente a *acción* no está en  
*frontera* ni en *explorados* con un valor heurístico  
menor o igual

**entonces**

añadir el nodo hijo a *frontera*

**fin si**

**fin para**

**fin repetir**

**fin función**

# Soluciones óptimas

Solución óptima: solución con el menor coste posible.

- ▶ para cada estado  $s$  y cada acción  $a$  aplicable a  $s$ , se necesita conocer el coste de obtener  $a(s)$ .
- ▶ coste de una secuencia de acciones: suma de los costes de las acciones de la secuencia.
- ▶ si el coste siempre es 1, entonces las soluciones óptimas son las soluciones minimales.

Búsqueda de una solución óptima:

- ▶ la heurística asocia a cada nodo el coste de la secuencia de acciones usada para llegar desde el estado inicial hasta el estado de ese nodo.
- ▶ completa, siempre y cuando el coste de cada acción exceda una cantidad positiva fija  $\epsilon$ .
- ▶ complejidad exponencial en tiempo y espacio.

# Búsqueda $A^*$

El valor heurístico de cada nodo se calcula como

$$f(n) = g(n) + h(n)$$

donde

- ▶  $g(n)$  es el coste de la secuencia de acciones usada para llegar desde el estado inicial hasta el estado de ese nodo.
- ▶  $h(n)$  es una estimación del coste de una secuencia de acciones óptima desde el estado de ese nodo hasta un estado final (por tanto 0 si el estado del nodo es final e  $\infty$  si no existe ninguna secuencia de acciones desde el estado del nodo hasta un estado final).

Por tanto,  $f(n)$  *estima* el coste de una solución óptima que pasa por el nodo  $n$ .



# Heurísticas admisibles

$h^*(n)$  denota el coste *exacto* de una solución óptima desde el estado de  $n$  hasta un estado final.

Una función heurística  $h$  se dice que es **admisible** si

$$h(n) \leq h^*(n), \text{ para todo nodo } n$$

Una heurística admisible proporciona una cota inferior del coste de una solución óptima desde cada nodo.

## Teorema

1. La búsqueda  $A^*$  es completa.
2. Si  $h$  es una heurística admisible, entonces la solución devuelta por  $A^*$  es óptima.

# Poda inducida por una heurística

Eficacia de una heurística: cantidad de ramas del árbol de búsqueda que evita explorar (poda del árbol de búsqueda).

Casos extremos:

- ▶  $h(n) = h^*(n)$ : se produce la máxima poda posible.
- ▶  $h(n) = 0$ : se produce la mínima poda posible (es una búsqueda óptima).

Una función heurística  $h_2$  se dice que es **más informada** que  $h_1$  si ambas son admisibles y

$$h_2(n) \geq h_1(n), \text{ para todo nodo } n$$

## Teorema

*Si  $h_2$  es mas informada que  $h_1$ , entonces todo nodo explorado por  $A^*$  usando  $h_2$  también es explorado por  $A^*$  usando  $h_1$ .*

# Complejidad de $A^*$

Complejidad en tiempo y espacio de la búsqueda  $A^*$ :

- ▶ En el peor de los casos es exponencial.
- ▶ En la práctica, depende en gran medida del problema particular y de la heurística empleada.

# Técnicas de construcción de heurísticas

Problema **relajado**: simplificación del problema obtenida eliminando restricciones que prohíben o penalizan ciertos movimientos en el problema original.

Para cada nodo, el coste exacto de una solución óptima en el problema relajado proporciona un valor heurístico en el problema original.

Otras técnicas:

- ▶ Combinación de heurísticas admisibles.
- ▶ Uso de información estadística.

# Heurística para las *Torres de Hanoi*

Asumimos que el objetivo es llevar todos los discos a la varilla 3.

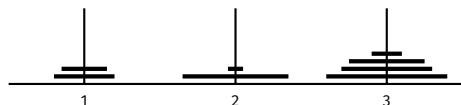
Relajación del problema: los discos se pueden colocar en cualquier posición de la varilla, no únicamente en lo alto.

El coste exacto en el problema relajado es el número de discos que no están en la varilla 3. Es una heurística admisible para el problema original.

Ejemplos:



Valor heurístico: 8



Valor heurístico: 4