Búsqueda

Carlos Linares López

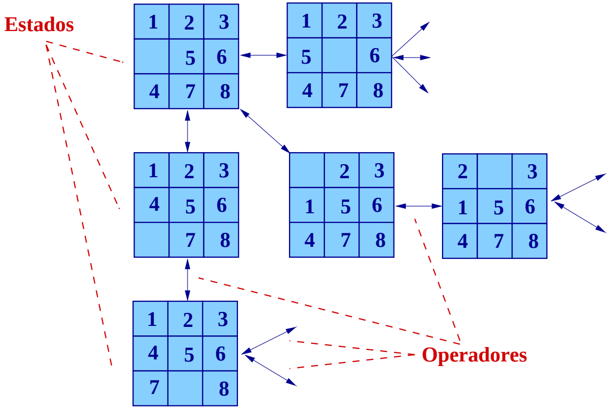
Grupo de Planificación y Aprendizaje (PLG)
Departamento de Informática
Escuela Politécnica Superior
Universidad Carlos III de Madrid

2 de diciembre de 2013

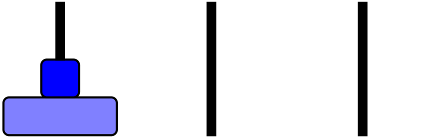
Definiciones

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s) o estado(s) final(es)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación, b
  - Profundidad del árbol de búsqueda, d

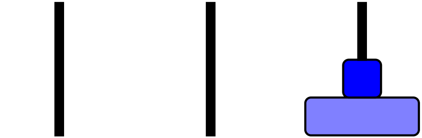
Ejemplo: 8-Puzzle



Ejemplo: Las torres de Hanoi (3,2)

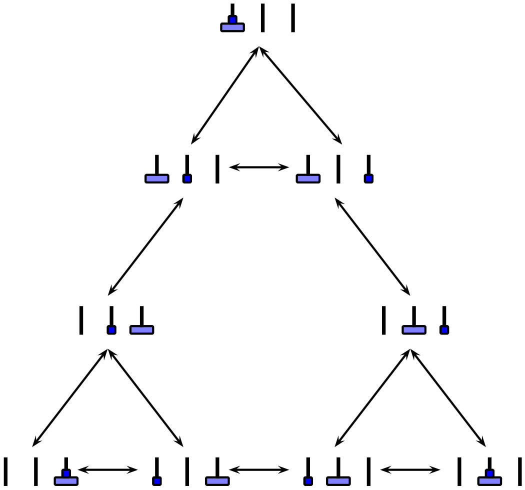


Estado inicial



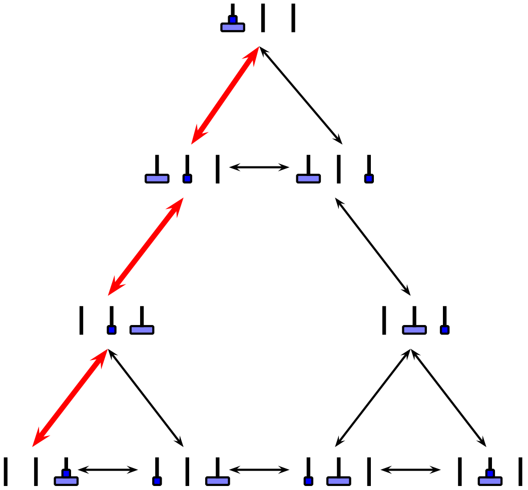
Estado final

Ejemplo: Las torres de Hanoi (3,2)



El *factor de ramificación* ($b = \frac{8}{3}$) es una propiedad del grafo de estados

Ejemplo: Las torres de Hanoi (3,2)



La *profundidad* ($d = 3$) es una propiedad del problema a resolver

Ejemplo: Las garrafas

- Simon dice:
  Se tienen dos garrafas de agua, una de cinco galones de capacidad y otra de tres. Ninguna de ellas tiene marcas de medición. Se tiene una bomba que permite llenar las jarras de agua, vaciarlas, y traspasar contenido de una garrafa a otra. ¿Cómo se puede lograr tener exactamente cuatro galones de agua en la jarra de cinco galones de capacidad?

Ejemplo: Las garrafas

- Espacio de Estados:
  - conjunto de pares ordenados de enteros (x, y), de forma que $x = 0, \ldots, 5$, $y = 0, \ldots, 3$
  - x representa el número de galones de agua que hay en la garrafa de 5 galones de capacidad
  - y representa el número de galones de agua que hay en la garrafa de 3 galones de capacidad
- Estado inicial: $(0, 0)$
- Estado meta:
  - Descripción implícita: $(4, n)$, donde $n = 0, \ldots, 3$
  - Descripción explícita: $(4, 0)$, $(4, 1)$, $(4, 2)$, $(4, 3)$

Ejemplo: Las garrafas

Operadores

Llenar garrafa grande: Si $(x < 5) \rightarrow (5, y)$
Llenar garrafa pequeña: Si $(y < 3) \rightarrow (x, 3)$
Vaciar garrafa grande: Si $(x > 0) \rightarrow (0, y)$
Vaciar garrafa pequeña: Si $(y > 0) \rightarrow (x, 0)$
Verter en grande: Si $(y > 0) \rightarrow (x + \text{mín}\{5 - x, y\}, y - \text{mín}\{5 - x, y\})$
Verter en pequeña: Si $(x > 0) \rightarrow (x - \text{mín}\{x, 3 - y\}, y + \text{mín}\{x, 3 - y\})$

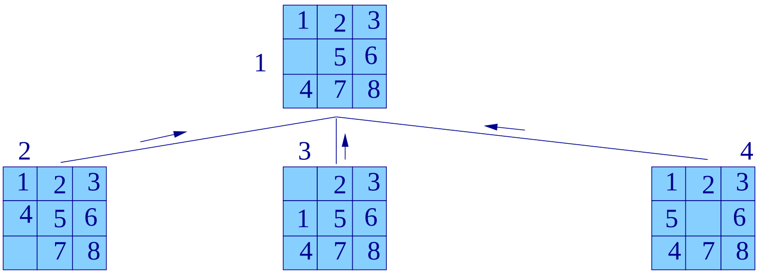
Explosión combinatoria

Dominio	**Número de estados**	**Tiempo (10^7 nodos/s)**
8-puzzle	$\left.\left(\frac{N^2!}{2}\right)\right\|_{N=3} = 181{,}440$	0.01 segundos
15-puzzle	$\left.\left(\frac{N^2!}{2}\right)\right\|_{N=4} = 10^{13}$	11,5 días
24-puzzle	$\left.\left(\frac{N^2!}{2}\right)\right\|_{N=5} = 10^{25}$	$31,7 \times 10^9$ años
Hanoi (3,2)	$(3^n)\|_{n=2} = 9$	9×10^{-7} segundos
Hanoi (3,4)	$(3^n)\|_{n=4} = 81$	$8,1 \times 10^{-6}$ segundos
Hanoi (3,8)	$(3^n)\|_{n=8} = 6561$	$6,5 \times 10^{-4}$ segundos
Hanoi (3,16)	$(3^n)\|_{n=16} = 4,3 \times 10^7$	4,3 segundos
Hanoi (3,24)	$(3^n)\|_{n=24} = 2,824 \times 10^{11}$	0,32 días
Cubo de Rubik $2 \times 2 \times 2$	10^6	0,1 segundos
Cubo de Rubik $3 \times 3 \times 3$	$4,32 \times 10^{19}$	31.000 años

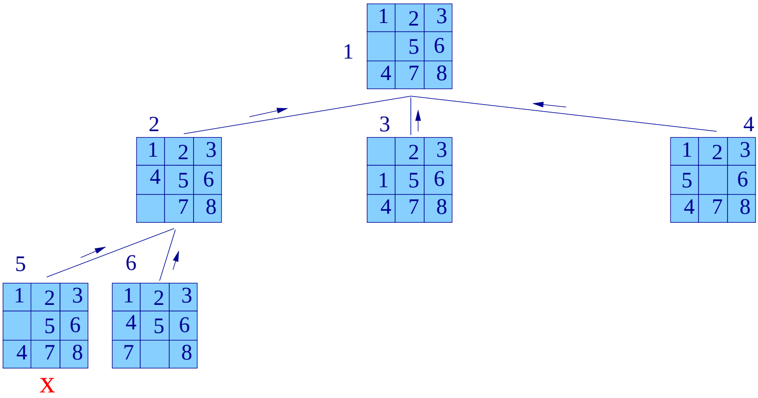
Ejercicio: 8 reinas

- Objetivo: Colocar 8 reinas en un tablero de ajedrez de manera que cada reina no ataque a ninguna otra (una reina ataca a otra si está en su misma fila, columna o diagonal)
- Dos posibles formulaciones del problema:
  - Formulación completa de estados: comienza con las 8 reinas en el tablero y las mueve
  - Formulación incremental: comienza con el tablero vacío, y añade una reina cada vez
- En cualquier caso, no importa el camino a la solución: sólo importa la solución (no hay descripción explícita de la meta)

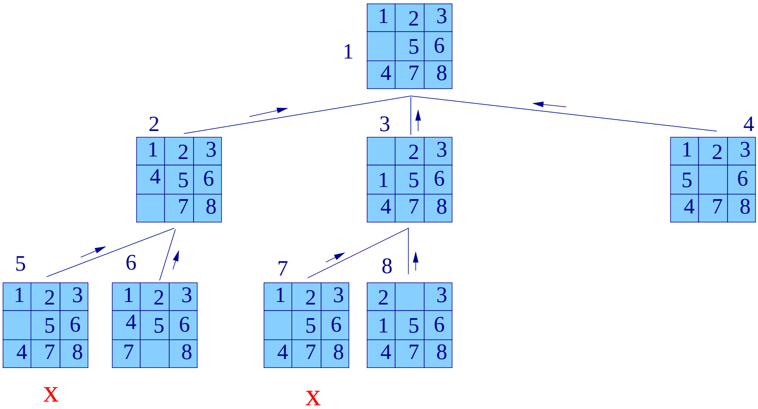
8-Puzle – Amplitud



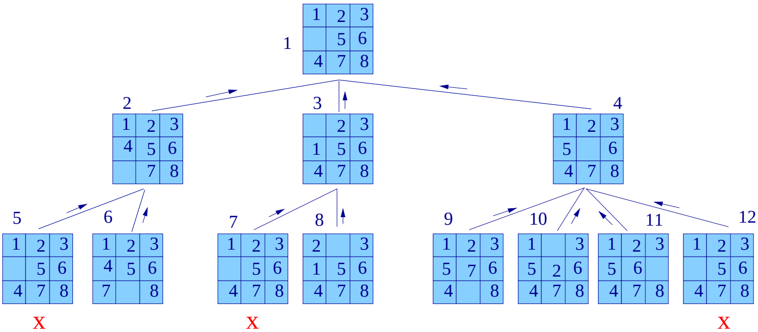
8-Puzle – Amplitud



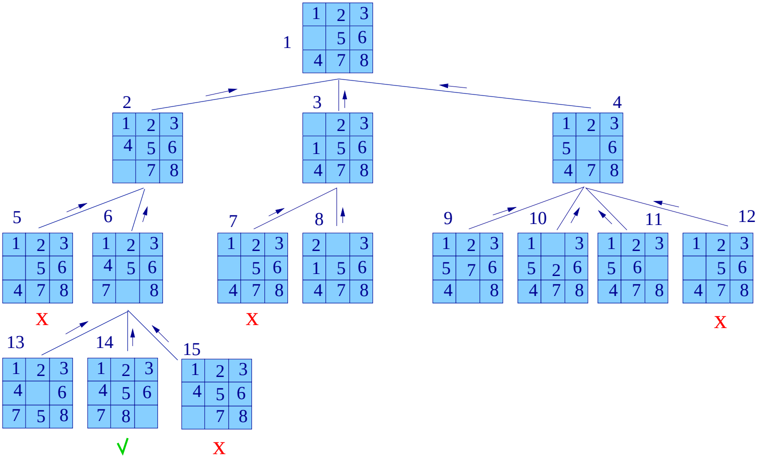
8-Puzle – Amplitud



8-Puzle – Amplitud



8-Puzle – Amplitud



Búsqueda en amplitud

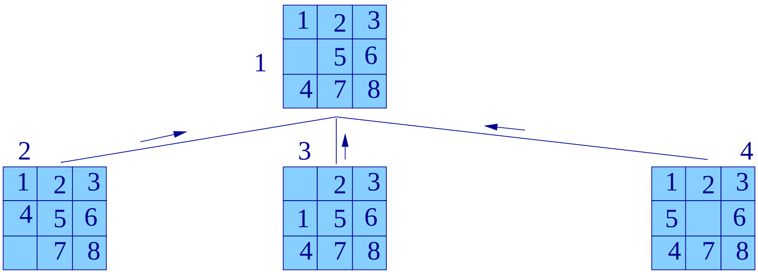
Procedimiento Amplitud (Estado-inicial, Estado-Final)

1. Crear lista ABIERTA con el nodo inicial, I, (estado-inicial)
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO
   - Quitar de ABIERTA el primer nodo, N
   - Si N tiene sucesores
   - Entonces Generar los sucesores de N
     - Crear punteros desde los sucesores hacia N
     - Si algún sucesor es nodo meta
     - Entonces EXITO=Verdadero
     - Si no Añadir los sucesores al final de ABIERTA
4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

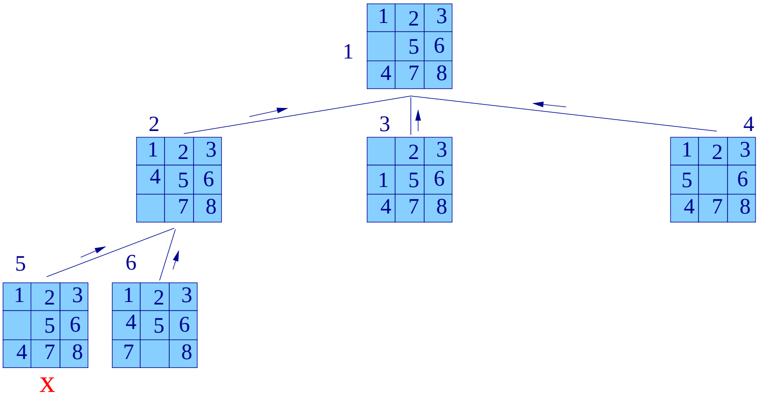
Características de la búsqueda en amplitud

- **Completitud**: encuentra solución si existe y el factor de ramificación es finito en cada nodo
- **Optimalidad**: si todos los operadores tienen el mismo coste, encontrará la solución óptima
- **Eficiencia**: buena si las metas están cercanas
- Problema: consume memoria exponencial

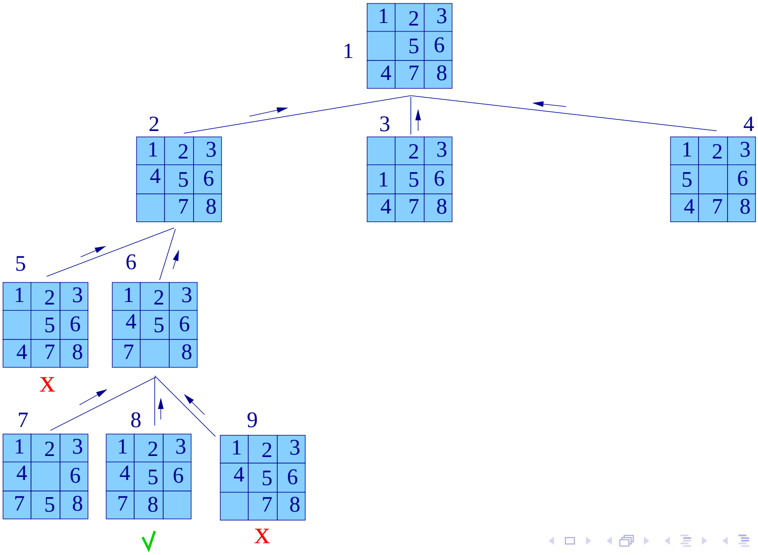
8-Puzle – Profundidad



8-Puzle – Profundidad



8-Puzle – Profundidad



Búsqueda en profundidad

Procedimiento Profundidad (Estado-inicial, Estado-Final Profundidad-máxima)

1. Crear lista ABIERTA con el nodo inicial, I, y su profundidad=0
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO

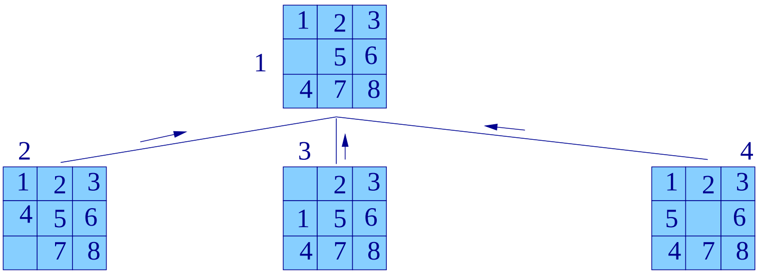
    Quitar de ABIERTA el primer nodo
    Lo llamaremos N y a su profundidad P
    Si P < Profundidad-máxima Y N tiene sucesores
    Entonces Generar los sucesores de N
        Crear punteros desde los sucesores hacia N
        Si algún sucesor es el Estado-Final
        Entonces EXITO=Verdadero
        Si no, Añadir los sucesores al principio de ABIERTA
            Asignarles profundidad P+1

4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

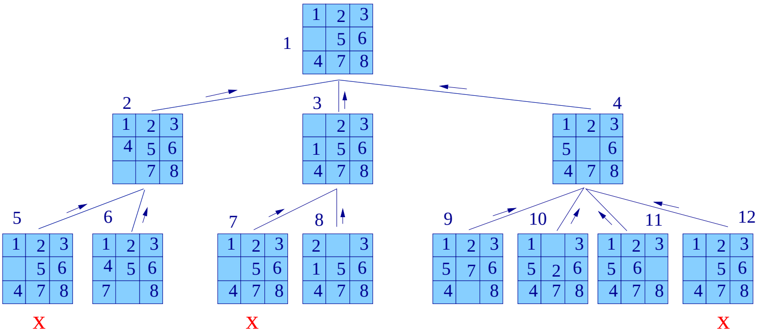
Características de la búsqueda en profundidad

- Requiere técnica de retroceso ("backtracking")
- Razones para retroceso:
  - Se ha llegado al límite de profundidad
  - Se han estudiado todos los sucesores de un nodo y no se ha llegado a la solución
  - Se sabe que el estado no conduce a la solución
  - Se genera un estado repetido
- **Completitud**: no asegura encontrar la solución
- **Optimalidad**: no asegura encontrar la solución óptima
- **Eficiencia**: bueno cuando metas alejadas de estado inicial, o problemas de memoria
- No es bueno, especialmente cuando hay ciclos

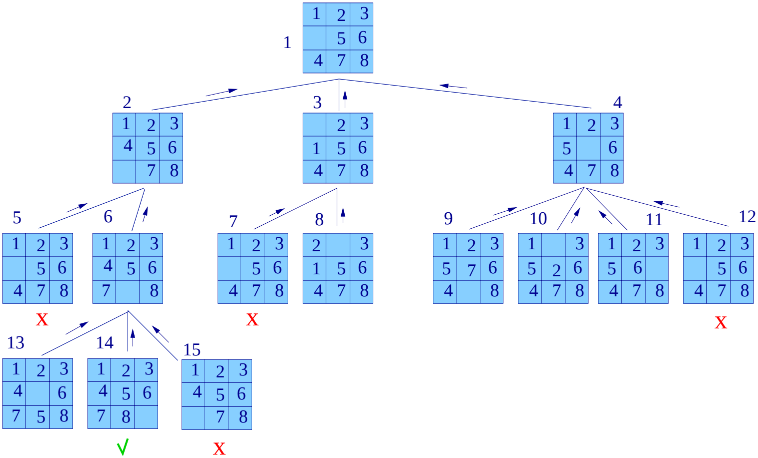
8-Puzle – Profundidad Iterativa



8-Puzle – Profundidad Iterativa



8-Puzle – Profundidad Iterativa



Búsqueda en profundidad iterativa

Procedimiento Profundidad-Iterativa (Estado-inicial, Estado-Final, Incremento)

1. Profundidad-máxima = Incremento
2. EXITO=Falso
3. Mientras que EXITO=Falso

   EXITO = Profundidad (Estado-inicial, Estado-Final,
   Profundidad-máxima)
   Profundidad-máxima += Incremento

4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

Características de la búsqueda en profundidad iterativa

- **Completitud**: encuentra la solución, si ésta existe
- **Optimalidad**: encuentra la solución óptima, si `Incremento=1`
- Eficiencia:

$$\frac{\mathrm{Tiempo(Profundidad - Iterativa)}}{\mathrm{Tiempo(Primero - Amplitud)}} = \frac{b}{b-1}$$

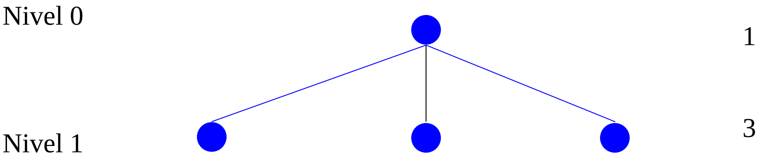
- Problema: puede generar muchos nodos duplicados

Análisis de complejidad: Problema

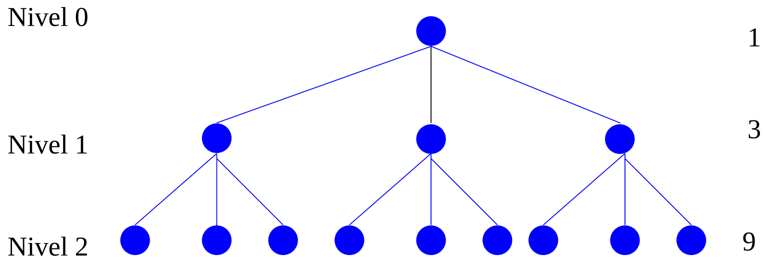
- Si se dispone de:
  - factor de ramificación medio, b
  - profundidad del árbol de búsqueda, d
- ¿Cuál sería, en el peor de los casos, el número de nodos que examinaría cada técnica?
  - Primero en Amplitud (5)
  - Primero en Profundidad (3)
  - Primero en Profundidad Iterativo (3)
- Pista:
  - supóngase que $b = 3$, e ir incrementando d
  - calcular de forma inductiva el número de nodos

Ejemplo

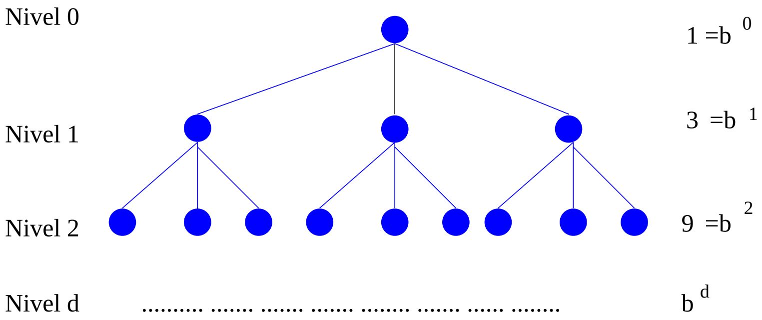
Ejemplo



Ejemplo



Ejemplo



Número máximo de nodos

Técnica de Búsqueda	Número máximo de nodos
Primero en amplitud	$\sum_{i=0}^{d} b^i$
Primero en profundidad	$\sum_{i=0}^{d} b^i$
Primero en profundidad iterativo	$\sum_{i=0}^{d}(d-i+1)b^i$

Complejidad temporal y espacial

Técnica de Búsqueda	Complejidad temporal	Complejidad espacial
Amplitud	$O(b^d)$	$O(b^d)$
Profundidad	$O(b^d)$	$O(d)$
Profundidad iterativa	$O(b^d)$	$O(d)$

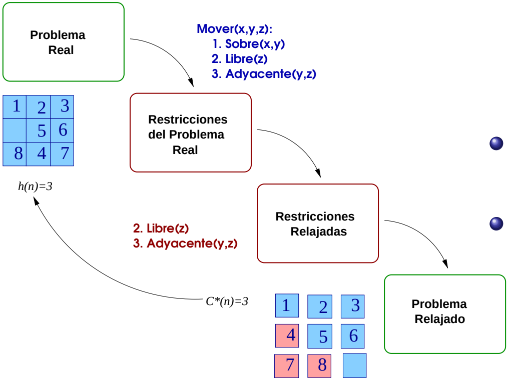
- Cuando complejidad en tiempo es igual a la de espacio, se agota la memoria antes que el tiempo

Heurísticas

- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - Metarreglas
  - Funciones $h(n, t)$
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Relajación de restricciones

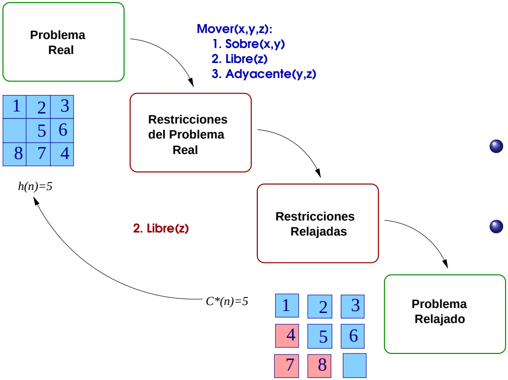
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



- $h(\cdot)$ es la solución **óptima** del problema relajado
- $h_1(\cdot)$: heurística del número de posiciones mal dispuestas

Relajación de restricciones

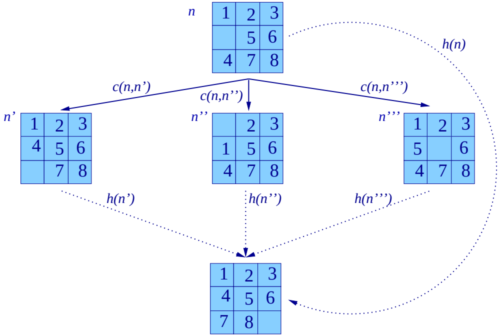
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



- $h(\cdot)$ es la solución **óptima** del problema relajado
- $h_2(\cdot)$: heurística de Manhattan

Admisibilidad

- Las funciones $h(\cdot)$ obtenidas por relajación deben ser necesariamente *monótonas*:



$$h(n) \leq c(n,n') + h(n')$$

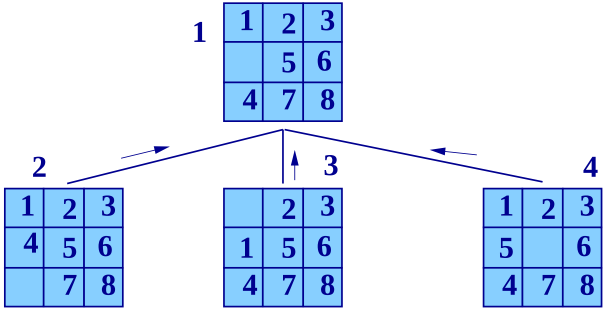
  - Y, por lo tanto, *admisibles*:

$$h(n) \leq h^*(n)$$

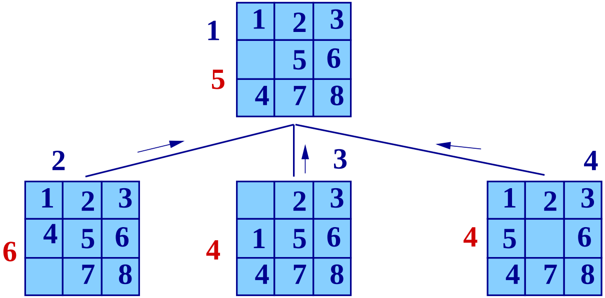
Otras relajaciones

- La *relajación* no es la única forma de simplificar problemas
- ¡Tampoco es cierto que cualquier *relajación* sea más fácil de resolver que el problema original!
- Otras alternativas son:
  - **Añadir restricciones** al problema original $\longrightarrow$ heurísticas no admisibles. Se usa para eliminar alternativas anticipadas que exceden el coste $h(\cdot)$
  - Por **estimación probabilística** de los descendientes más prometedores
  - Por **razonamiento por analogía** o **metafórico**

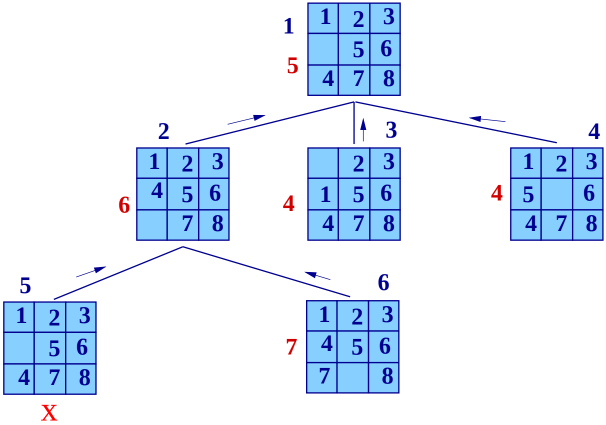
8-Puzle – Búsqueda en escalada



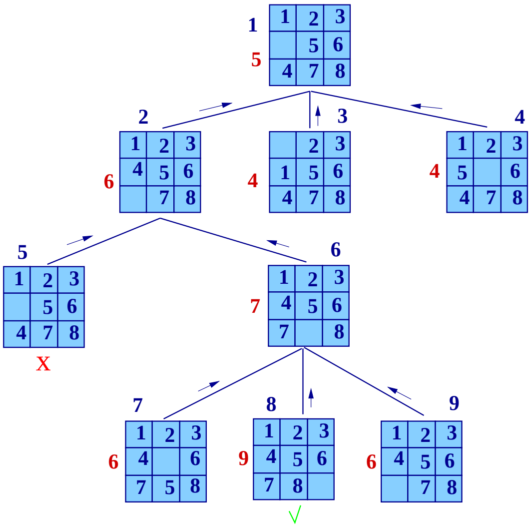
8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



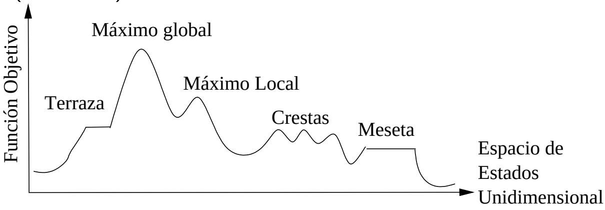
Búsqueda en escalada

Procedimiento escalada (Estado-inicial Estado-final)

N=Estado-inicial; EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    Generar los sucesores de N
    SI algún sucesor es Estado-final
    ENTONCES EXITO=Verdadero
    SI NO, Evaluar cada nodo con la función de evaluación, $f(n)$
        N=mejor sucesor
Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
    al nodo N por los punteros
Si no, Solución=fracaso

Características

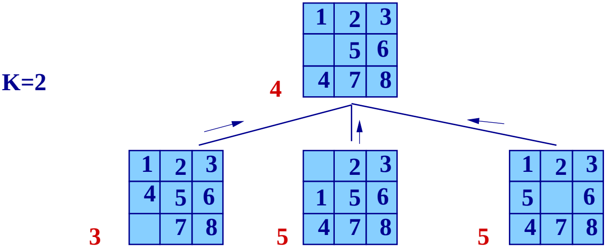
- Problemas de los métodos *avariciosos*
  - **Máximos (o mínimos) locales**: pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo global
  - **Mesetas**: zona del espacio de estados con función de evaluación plana
  - **Crestas**: zona del espacio de estados con varios máximos (mínimos) locales



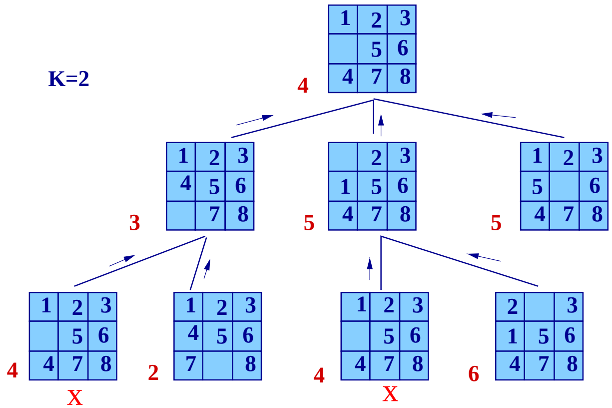
Características

- Soluciones
  - Retroceso
  - Dar más de un paso
  - Reinicio aleatorio
- Método local
  - Completitud: no tiene porqué encontrar la solución
  - Admisibilidad: no siendo completo, aún menos será admisible
  - Eficiencia: rápido y útil si la función es monótona (de)creciente

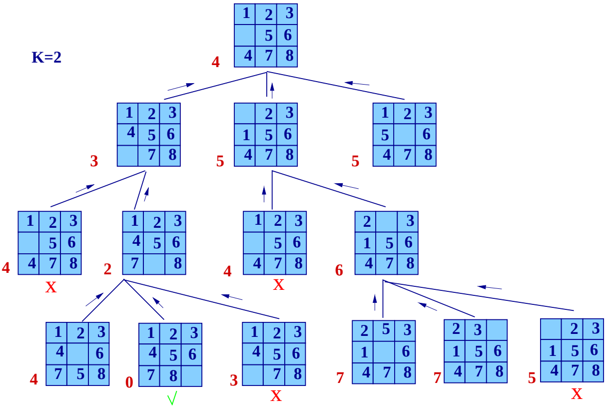
8 Puzle – Búsqueda en haz



8 Puzle – Búsqueda en haz



8 Puzle – Búsqueda en haz



Búsqueda en haz

Procedimiento "Beam Search" (Estado-inicial Estado-final K)

ABIERTA=(Estado-inicial); EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    ABIERTA=Todos los sucesores de los nodos de ABIERTA
    SI algún nodo de ABIERTA es Estado-final
    ENTONCES EXITO=Verdadero
    SI NO, Evaluar cada nodo con la función de evaluación $f(n)$
        ABIERTA=K mejores nodos de ABIERTA
Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
    al nodo N por los punteros
Si no, Solución=fracaso

Características

- La búsqueda en haz es una generalización de la escalada: $HC = BS(k = 1)$
- Abriendo la ventana de sucesores eligibles, mejoran las posibilidades de:
  - Escapar de los *plateaus* o mesetas formadas por la función heurística
  - Encontrar caminos más cortos hasta alguna meta
- Sin embargo, no es cierto que el algoritmo encuentra soluciones mejores con valores de k mayores, aunque lo normal es que sea así

Búsqueda de el mejor primero

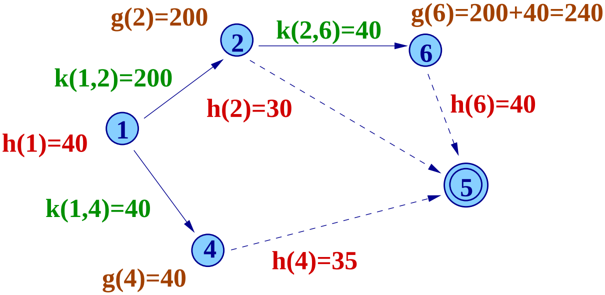
Procedimiento Mejor-primero (Estado-inicial Estado-final)

Crear grafo de búsqueda G, con el nodo inicial, I (Estado-inicial)
ABIERTA=I, CERRADA=Vacío, EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    Quitar el primer nodo de ABIERTA, N y meterlo en CERRADA
    SI N es Estado-final ENTONCES EXITO=Verdadero
    SI NO Expandir N, generando el conjunto S de sucesores de N,
            que no son antecesores de N en el grafo
        Generar un nodo en G por cada s de S
        Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G
        Añadirlos a ABIERTA
        Para cada s de S que estuviera ya en ABIERTA o CERRADA
            decidir si redirigir o no sus punteros hacia N
        Para cada s de S que estuviera ya en CERRADA
            decidir si redirigir o no los punteros de los nodos en sus subárboles
        Reordenar ABIERTA según $f(n)$
Si EXITO Entonces Solución=camino desde I a N a través de los punteros de G
Si no Solución=Fracaso

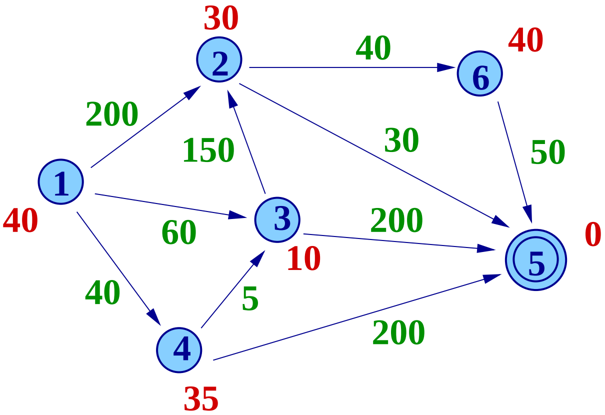
A* (Hart, Nilsson y Raphael, 1968)

- Función de ordenación de nodos: $f(n) = g(n) + h(n)$
  - $f(n)$: función de evaluación
  - $g(n)$: función de coste de ir desde el nodo inicial al nodo n
  - $h(n)$: función heurística que mide la distancia estimada desde n a algún nodo meta
- $g(n)$ se calcula como la suma de los costes de los arcos recorridos, $k(n_i, n_j)$
- Los valores reales sólo se pueden conocer al final de la búsqueda
  - $f^*(n)$: coste real para ir desde el nodo inicial a algún nodo meta a través de n
  - $g^*(n)$: coste real para ir desde el nodo inicial al nodo n
  - $h^*(n)$: coste real para ir desde el nodo n a algún nodo meta

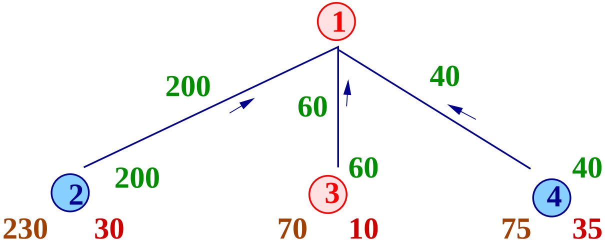
Grafo no dirigido – A* (definiciones)



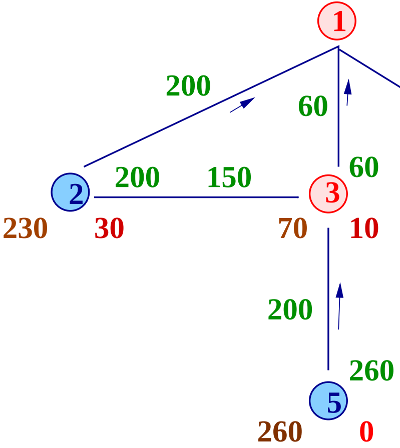
Grafo no dirigido – A*



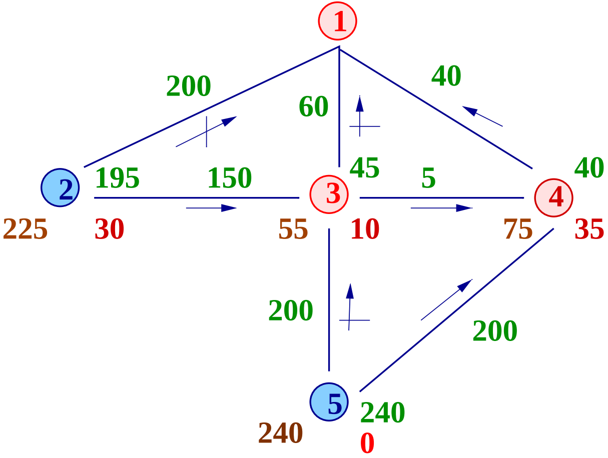
Grafo no dirigido – A*



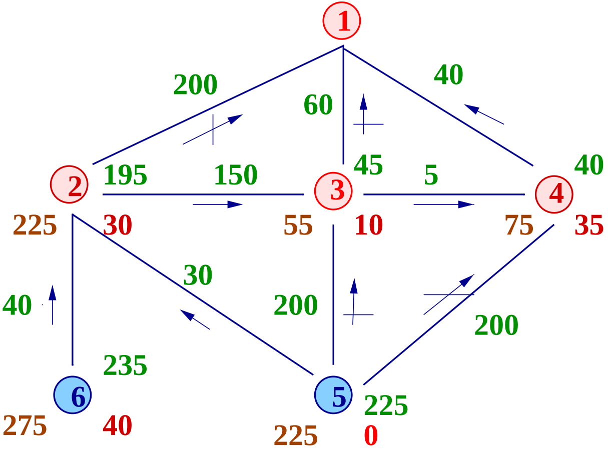
Grafo no dirigido – A*



Grafo no dirigido – A*



Grafo no dirigido – A^*



Características

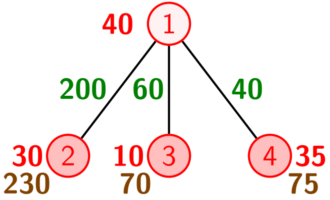
- Completitud: si existe solución, la encuentra
- Admisibilidad: si hay una solución, encuentra la óptima si:
  - el número de sucesores es finito para cada nodo,
  - $k(n_i, n_j) \geq \epsilon > 0$ en cada arco, y
  - La función heurística $h(\cdot)$ es admisible, $h(n) \leq h^*(n) \quad \forall n$
- Si $h_1(n) <= h_2(n) \forall n$, $h_2(n)$ está más informada que $h_1(n)$ y servirá para expandir menos nodos
  - Ejemplo: distancia de Manhattan está más informada que número de casillas mal colocadas (problema de Manhattan es menos relajado que el del número de casillas)
- Extremos:
  - $h(n) = 0$ para cada nodo: no se tiene información (Dijkstra)
  - $h(n) = h^*(n)$ para cada nodo: se tiene información perfecta
- No tiene sentido dedicar más coste computacional a calcular una buena $h(n)$ que a realizar la búsqueda equivalente: equilibrio

Resumen de técnicas de el mejor primero

- No informadas:
  - Búsqueda en amplitud: $f(n) = \text{profundidad}(n)$
  - Dijkstra: $f(n) = g(n)$
- Informadas (heurísticas):
  - Escalada y búsqueda en haz: $f(n) = h(n)$
  - A*, IDA*: $f(n) = g(n) + h(n)$
  - Ponderadas: $f(n) = g(n) + \omega h(n), \omega > 1$
    - Es completo, pero no es admisible
    - La solución óptima tiene un coste menor o igual que $(1 + \omega)$ veces la generada

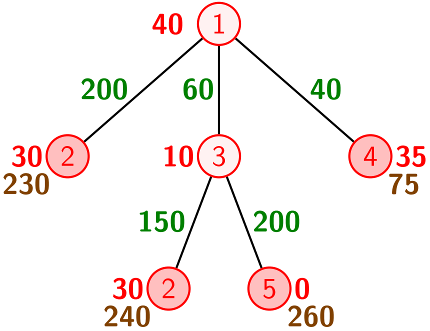
Grafo no dirigido – IDA* (Korf, 1985)

Iteración 1, $\eta_1 = h(1) = 40$



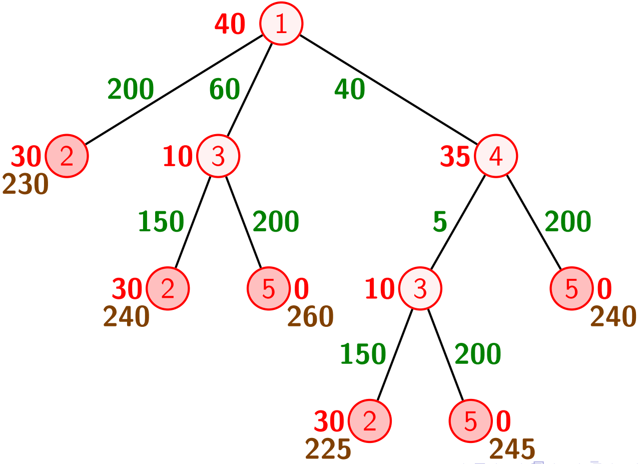
Grafo no dirigido – IDA*

Iteración 2, $\eta_2 = f(3) = 70$



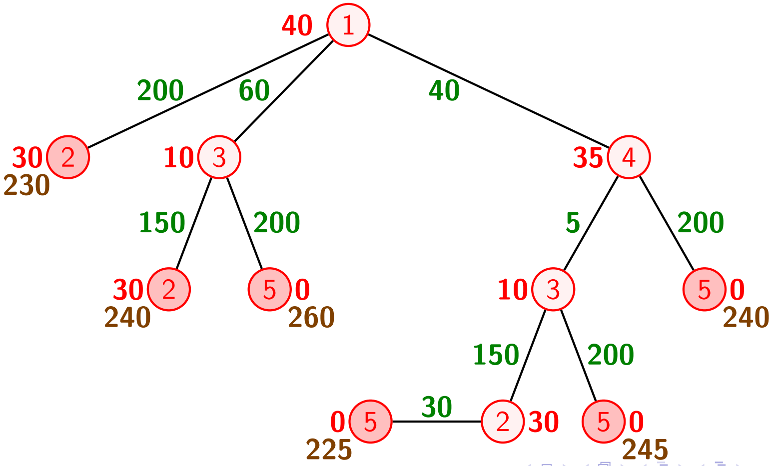
Grafo no dirigido – IDA*

Iteración 3, $\eta_3 = f(4) = 75$



Grafo no dirigido – IDA*

Iteración 4, $\eta_4 = f(2) = 225$



IDA*

Procedimiento IDA* (Estado-inicial Estado-final)

EXITO=Falso
$\eta = h(s)$
Mientras que EXITO=Falso
  EXITO=Profundidad (Estado-inicial,η)
  $\eta = \min_{i=1,n}\{f(i)\} = \min_{i=1,n}\{g(i)+h(i)\}$
Solución=camino desde nodo del Estado-inicial
  al Estado-final por los punteros

Profundidad (Estado-inicial,η)

Expande todos los nodos cuyo coste $f(n)$ no excede el valor de η

Características

- **Completitud**: El algoritmo IDA* es completo, esto es, encuenta una solución si existe alguna
- **Admisibilidad**: Además, el algoritmo IDA* es admisible y, por lo tanto, encontrará la solución óptima
- Mientras su complejidad de tiempo es también exponencial, su complejidad de espacio es lineal en la profundidad del árbol de búsqueda
- Aunque pudiera parecer lo contrario, el número de *re-expansiones* es sólo mayor en un pequeño factor que el número de expansiones de los algoritmos de el mejor primero
- Fue el primer algoritmo que resolvió óptimamente 100 casos generados aleatoriamente en el 15–Puzle