Agentes Inteligentes

Javier García

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

October 13, 2021

Part IV

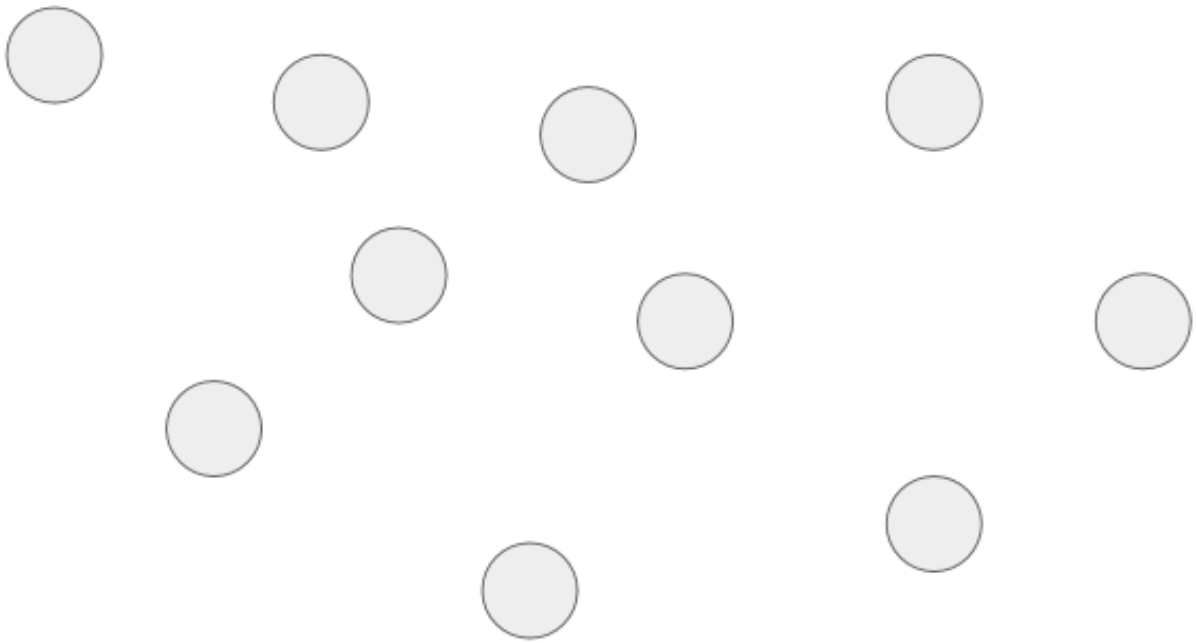
Búsqueda

Contenidos

Espacios de búsqueda

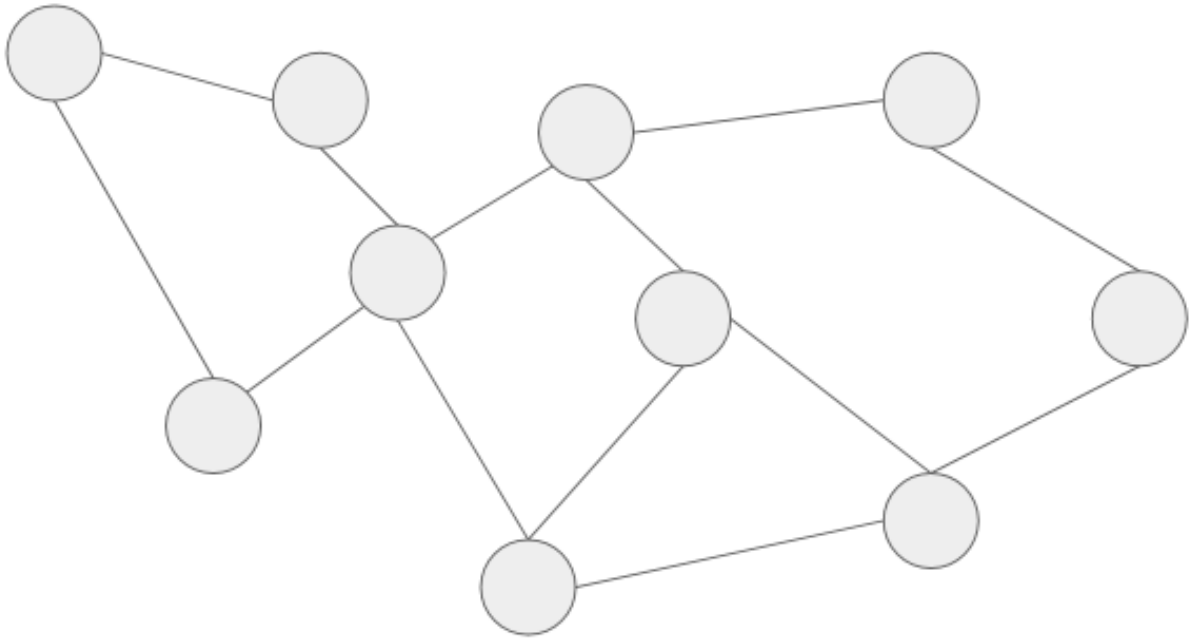
Definiciones

Espacio de búsqueda:



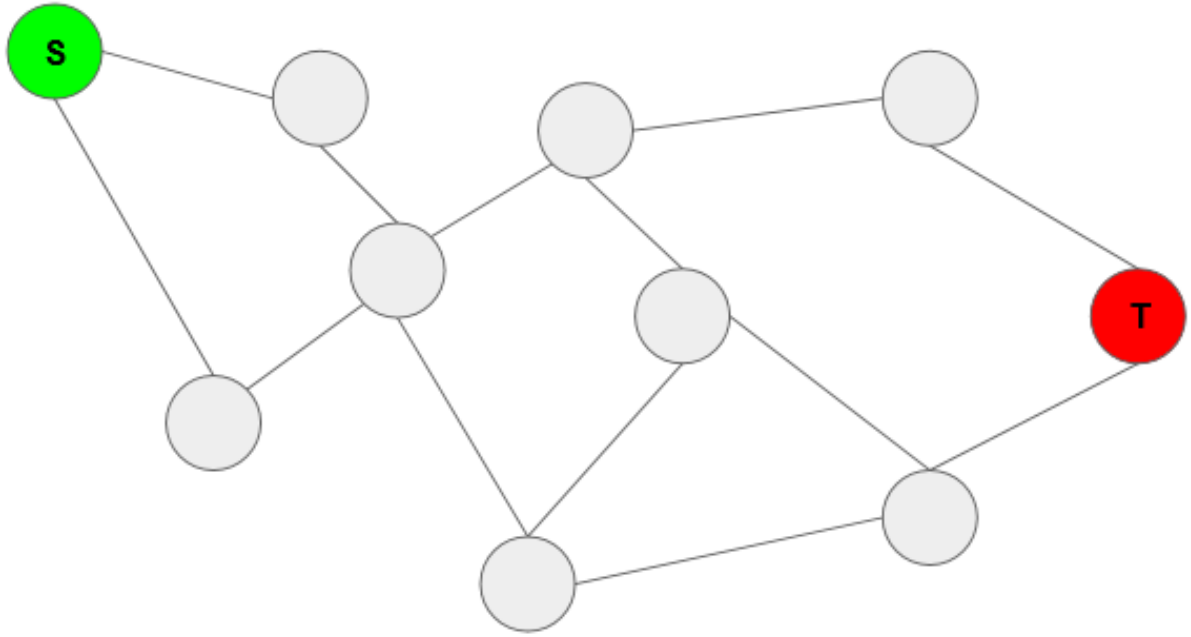
Definiciones

Espacio de búsqueda:



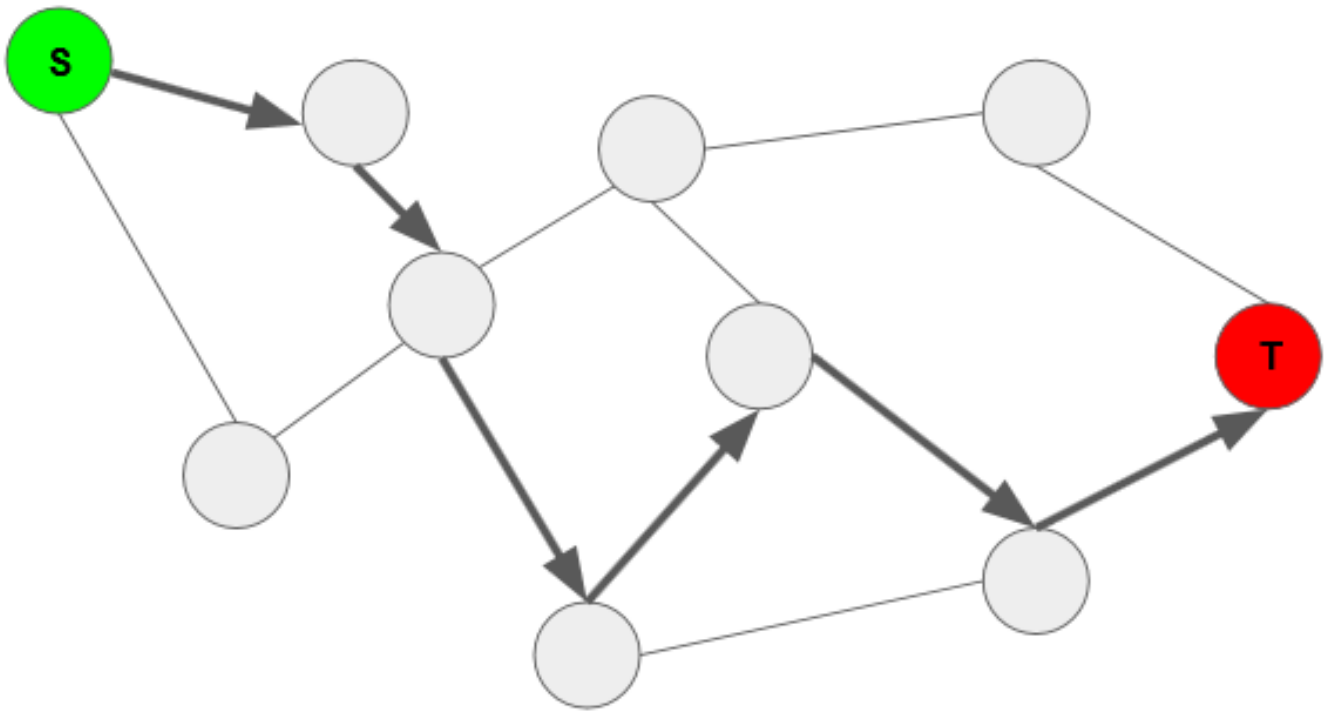
Definiciones

Espacio de búsqueda:



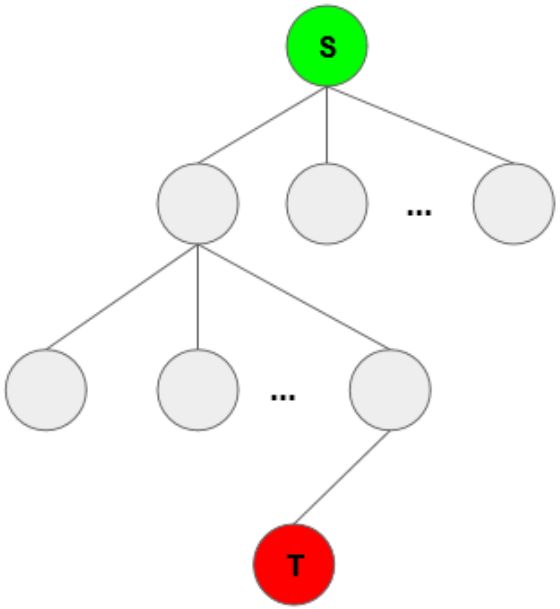
Definiciones

Espacio de búsqueda:



Definiciones

Espacio de búsqueda:



Los GRAFOS DE BÚSQUEDA se recorren con ÁRBOLES DE BÚSQUEDA

Definiciones

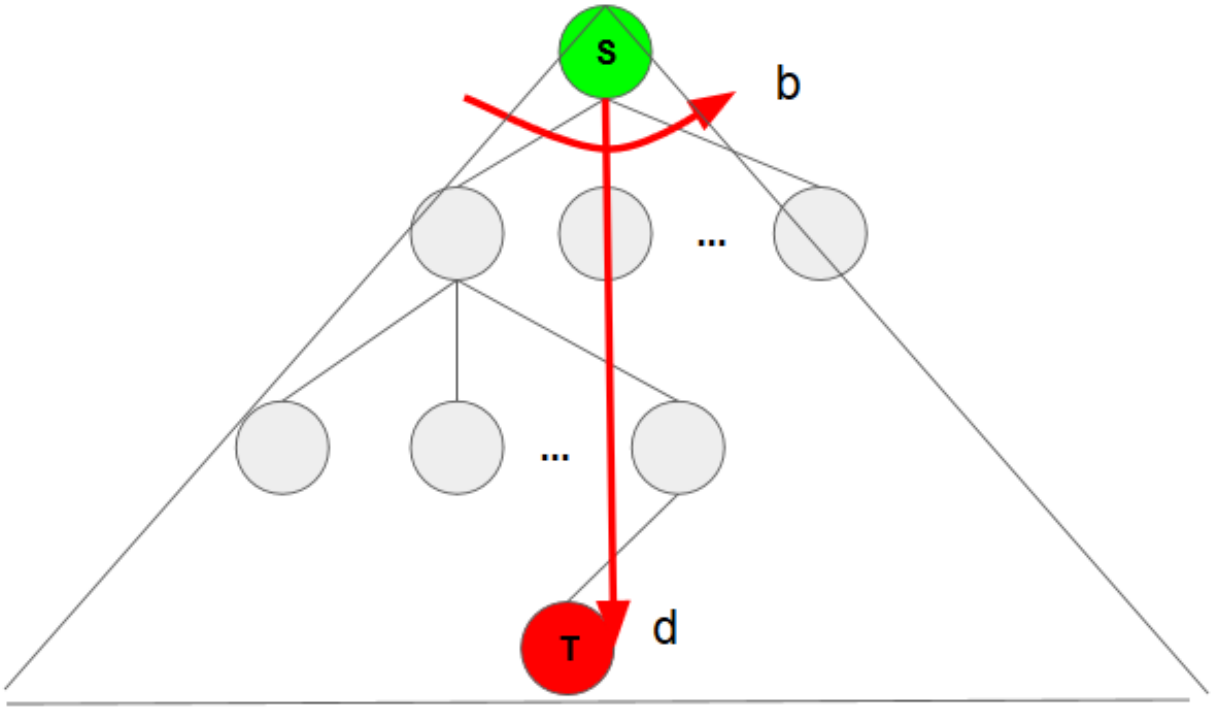
- Espacio de búsqueda
  - Conjunto de estados: cada estado contiene información de caracter **estático** y se representan típicamente con **estructuras de datos** (e.g., una clase, un objeto, struct)
  - Conjunto de operadores: implementan el conocimiento de caracter **dinámico** y se representan típicamente como funciones o métodos

    if <preconds> **then**
    <effects>

  - 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

Definiciones

- Parámetros importantes
  - Factor de ramificación, b: número medio de sucesores de cada nodo
  - Profundidad del árbol de búsqueda, d: número mínimo de niveles hasta alguna solución



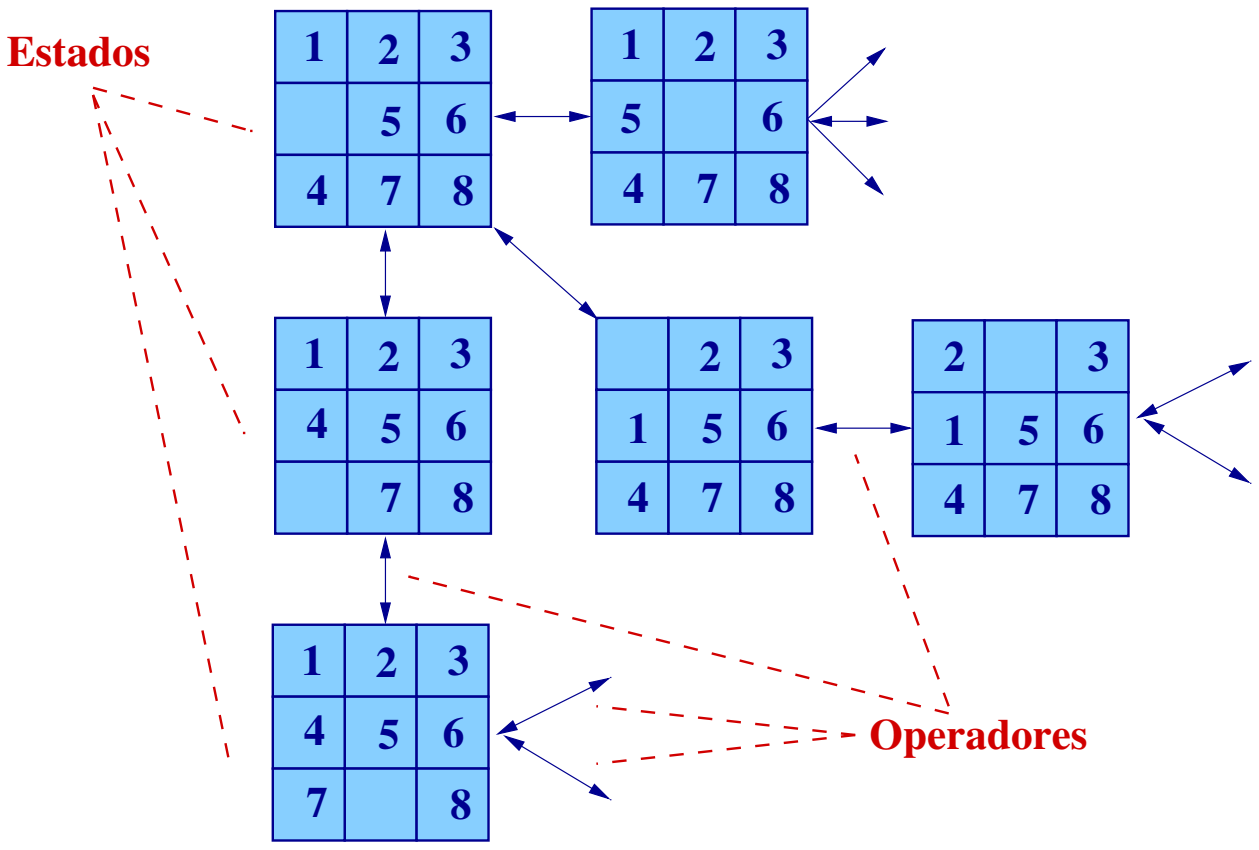
Definiciones

- **Generación** de un nodo: es el proceso de creación de un nuevo estado en memoria
- **Expansión** de un nodo: es el proceso de *generar* todos los sucesores de un nodo
- **Completitud**: un algoritmo es completo si garantiza que encontrará una solución considerando una cantidad infinita de recursos computacionales
- **Admisibilidad**: si se garantiza que se encuentra la solución óptima
- En general, se asume que no hay preferencia en la aplicación de los operadores, i.e., tenemos costes unitarios

Tipos de problemas

- **Estático**: Formulación y búsqueda se realizan sin prestar atención a los cambios en el entorno
- **Observable**: El agente tiene acceso completo a la información del estado
- **Discreto**: Número finito de estados y acciones
- **Determinista**: Dado una acción a en un estado s, transita con probabilidad 1 al estado s'
- **No episódico**: Las acciones influyen en las futuras acciones
- y además...
- Mantienen un modelo del entorno...
- ...y desean modificar el estado del entorno de acuerdo a sus objetivos

Ejemplo: 8-Puzzle



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 garrafas de agua, vaciarlas, y traspasar contenido de una garrafa a otra. ¿Cómo se puede lograr tener exactamente cuatro galones de agua en la garrafa 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 $(y > 0) \rightarrow (x + \min\{5 - x, y\}, y - \min\{5 - x, y\})$
Verter en pequeña :Si $(x > 0) \rightarrow (x - \min\{x, 3 - y\}, y + \min\{x, 3 - y\})$

Explosión combinatoria

Dominio	**Número de estados**	**Tiempo (10^7 nodos/s)**
8-puzzle	$\left(\frac{N^2!}{2}\right)\Big\|_{N=3} = 181.440$	0.01 segundos
15-puzzle	$\left(\frac{N^2!}{2}\right)\Big\|_{N=4} = 10^{13}$	11,5 días
24-puzzle	$\left(\frac{N^2!}{2}\right)\Big\|_{N=5} = 10^{25}$	$31,7 \times 10^9$ 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

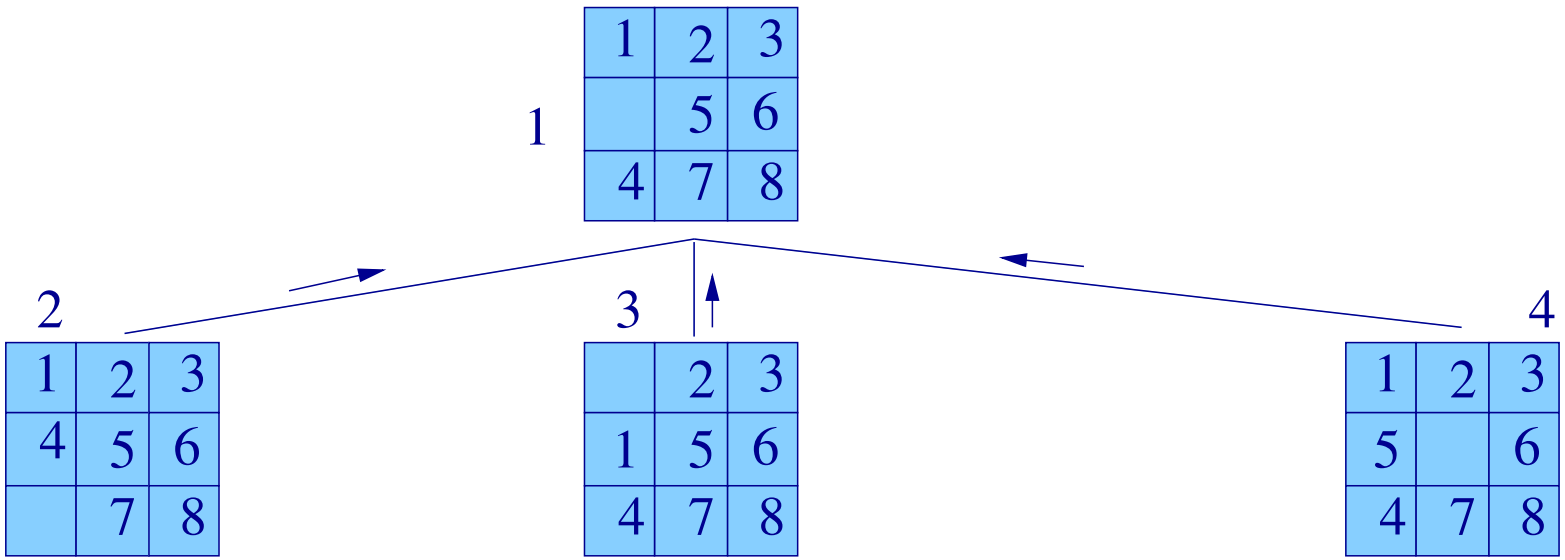
Ejercicio: 8 reinas

- 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 y podría haber más de una

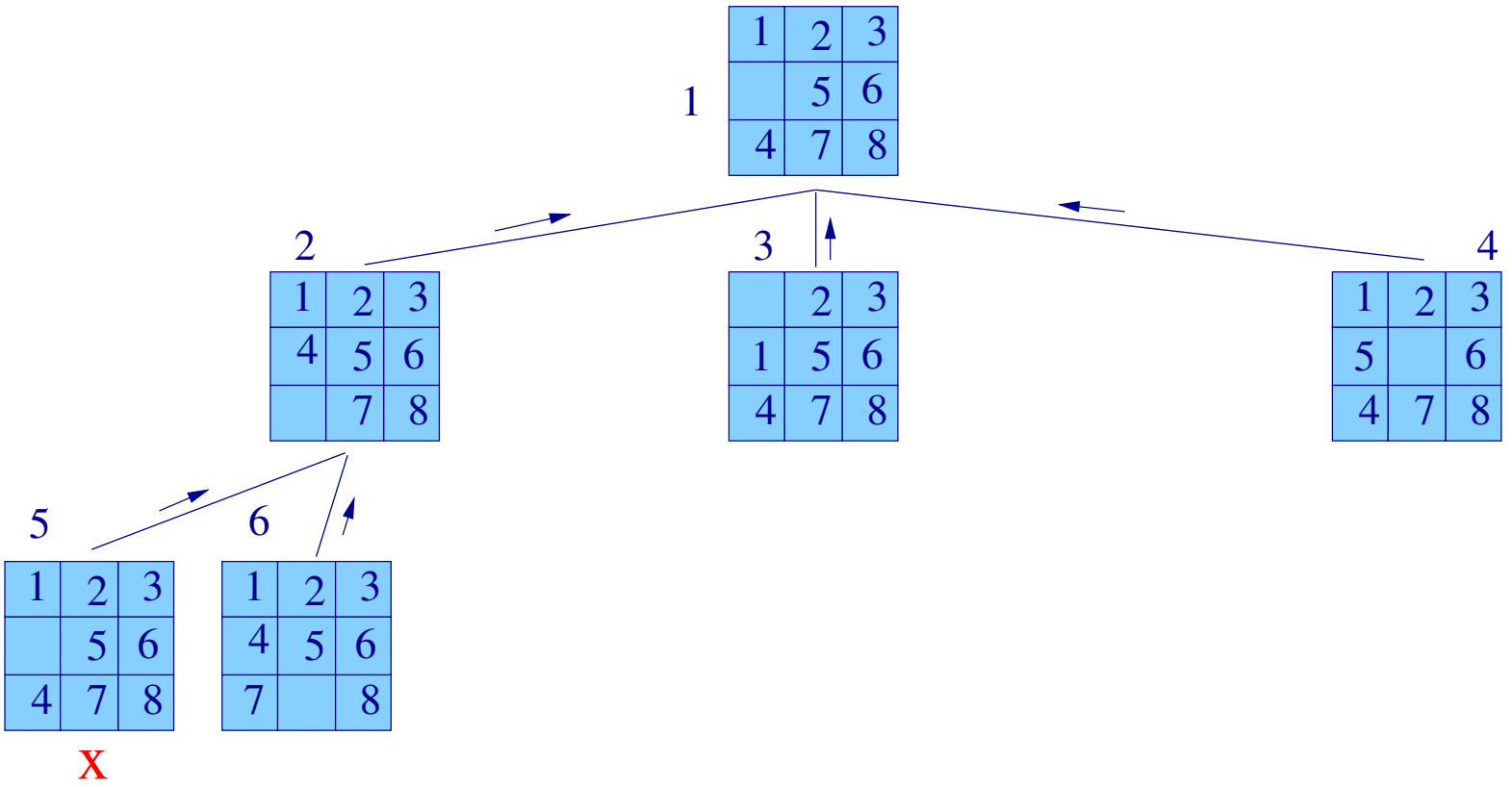
Búsqueda no informada

8-Puzle – Amplitud

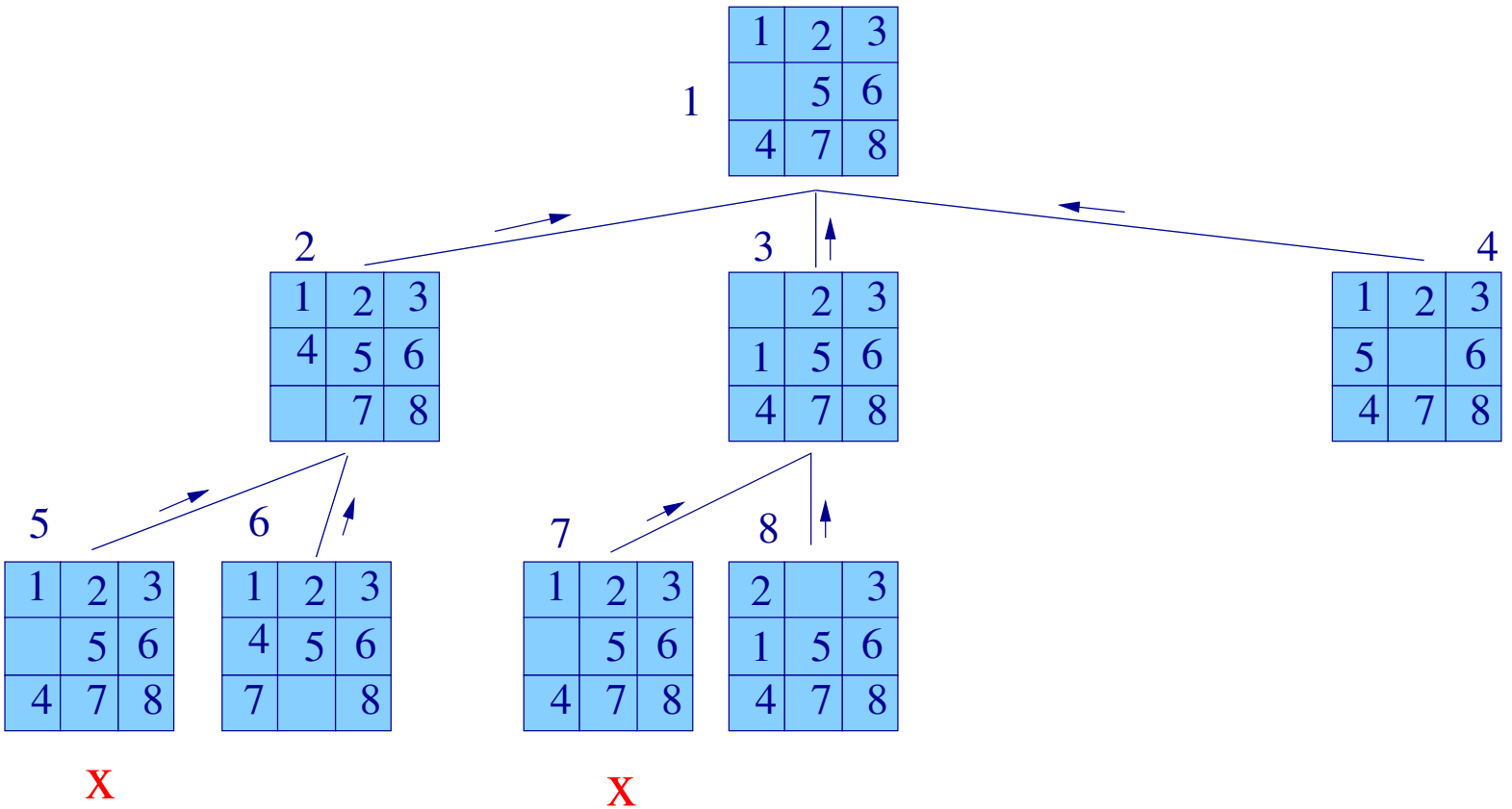
"Nunca expande un nodo a profundidad *d* si no ha expandido todos los nodos a profundidad *d-1*"



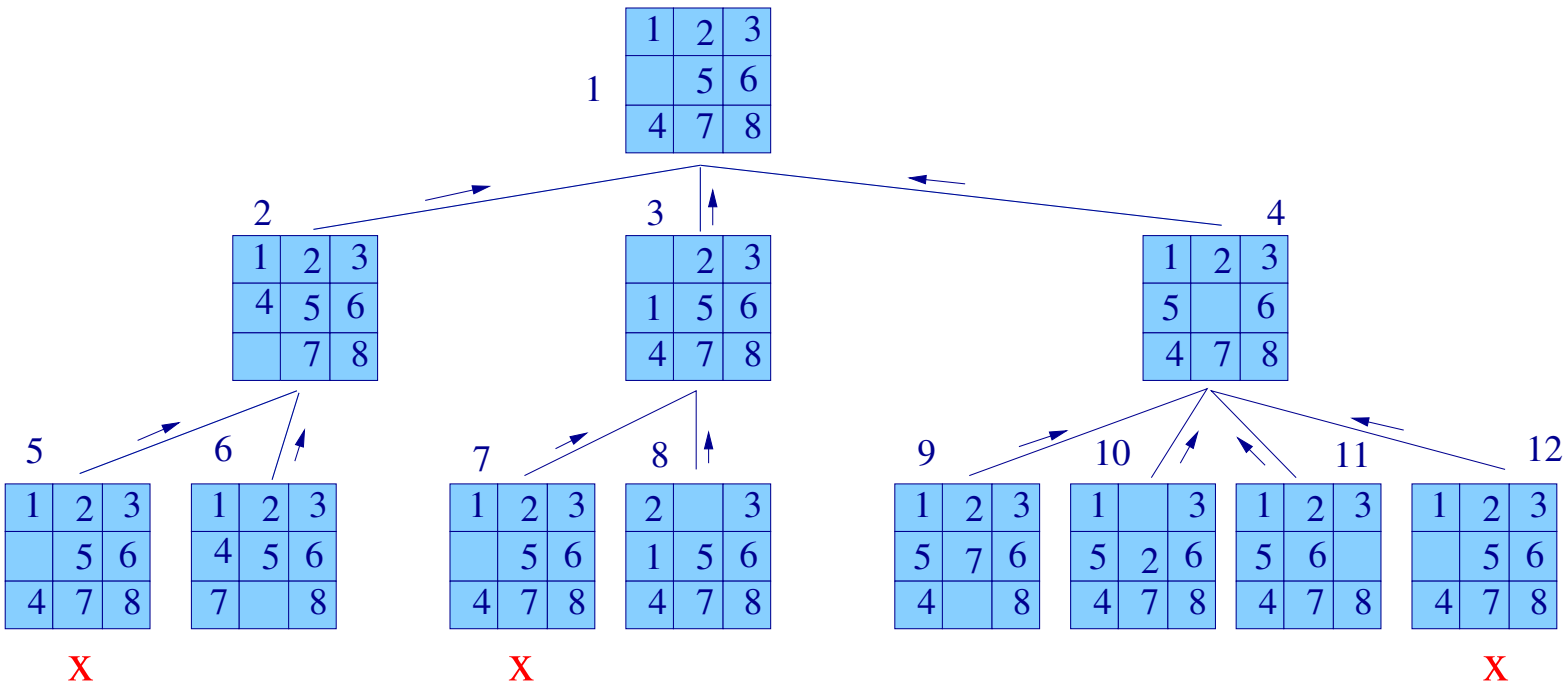
8-Puzle – Amplitud



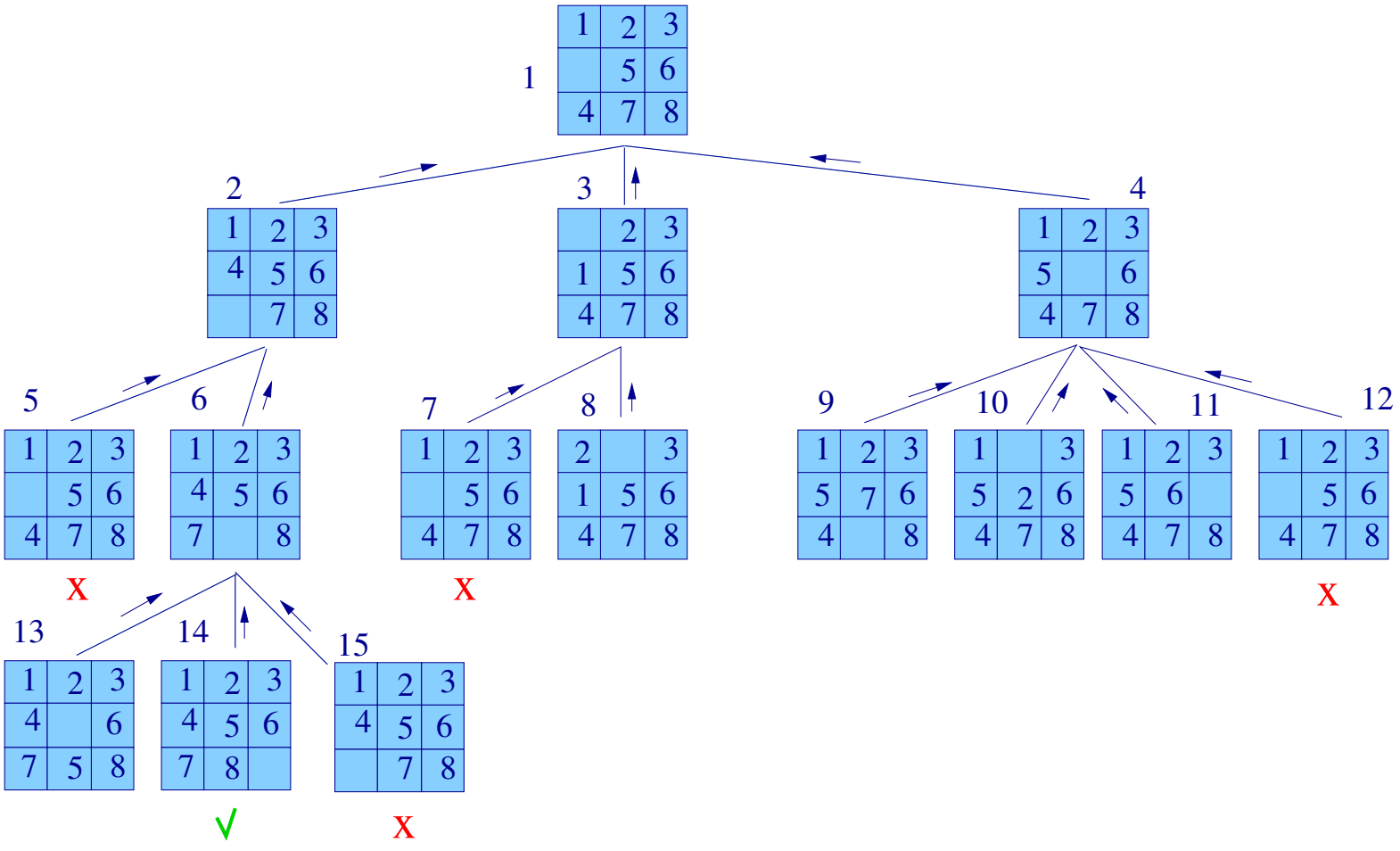
8-Puzle – Amplitud



8-Puzle – Amplitud



8-Puzle – Amplitud



Búsqueda en amplitud

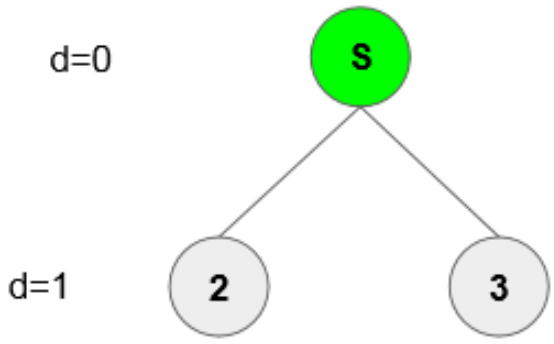
Procedimiento Amplitud (Estado-inicial, Estado-Final)

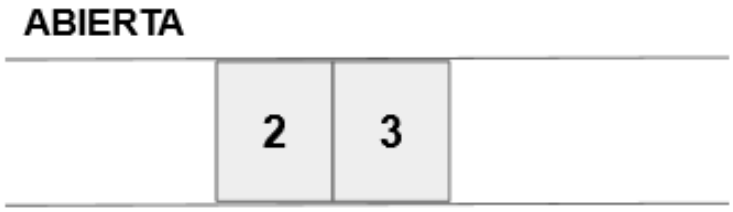
1. Crear lista ABIERTA con el nodo inicial, I, (estado-inicial)
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO
   - Quitar de ABIERTA el primer nodo, N
   - Si N tiene sucesores
   - Entonces Generar los sucesores de N
     - Crear punteros desde los sucesores hacia N
     - Si algún sucesor 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

Amplitud

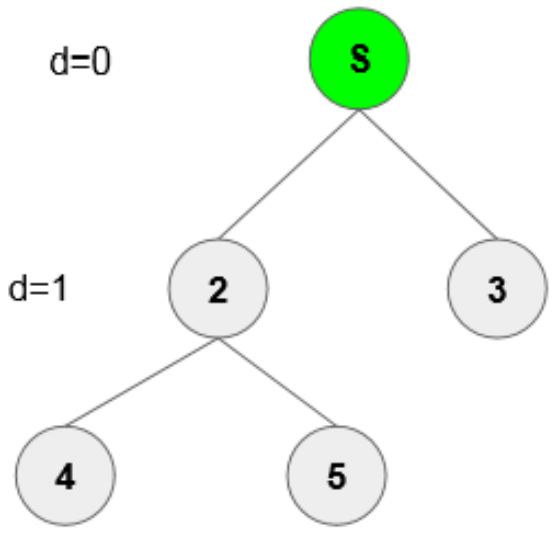
d=0

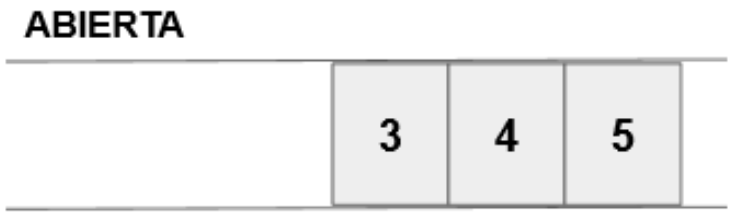
Amplitud



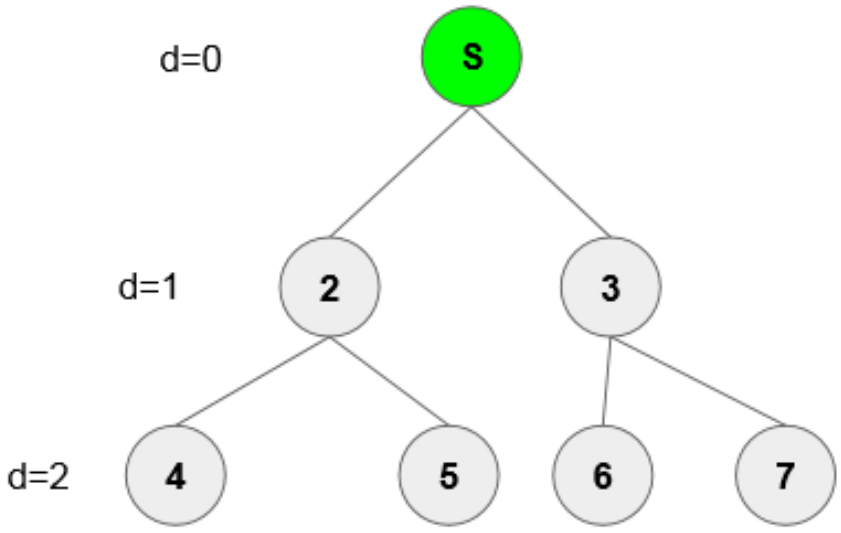


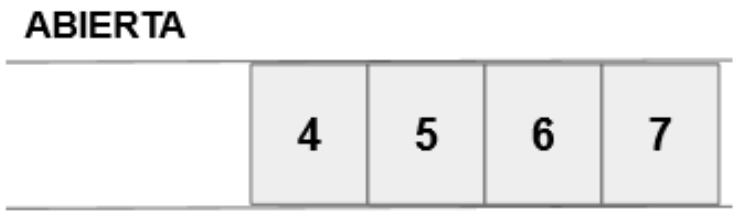
Amplitud



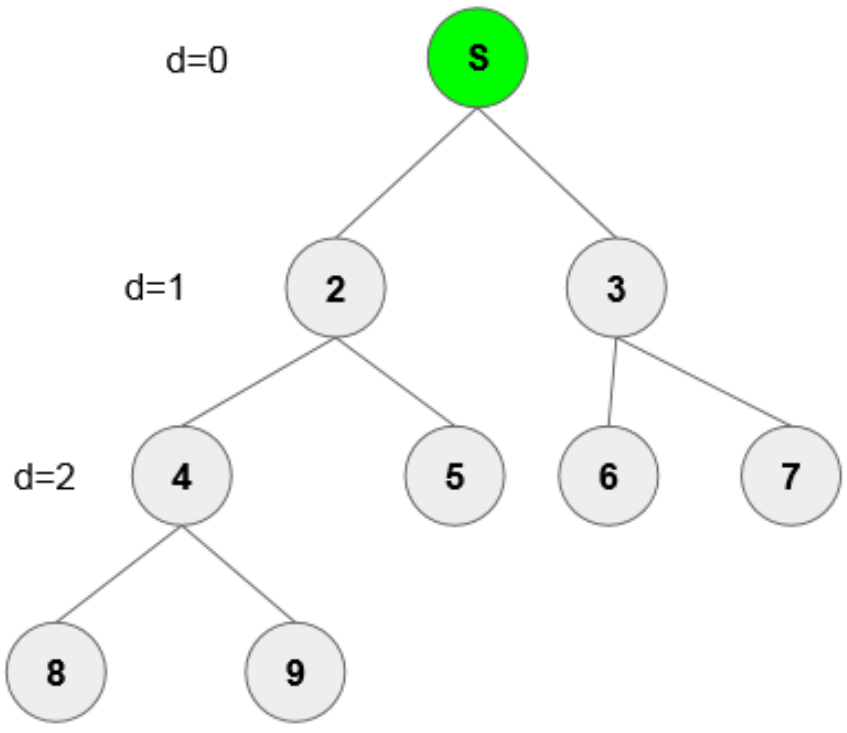


Amplitud

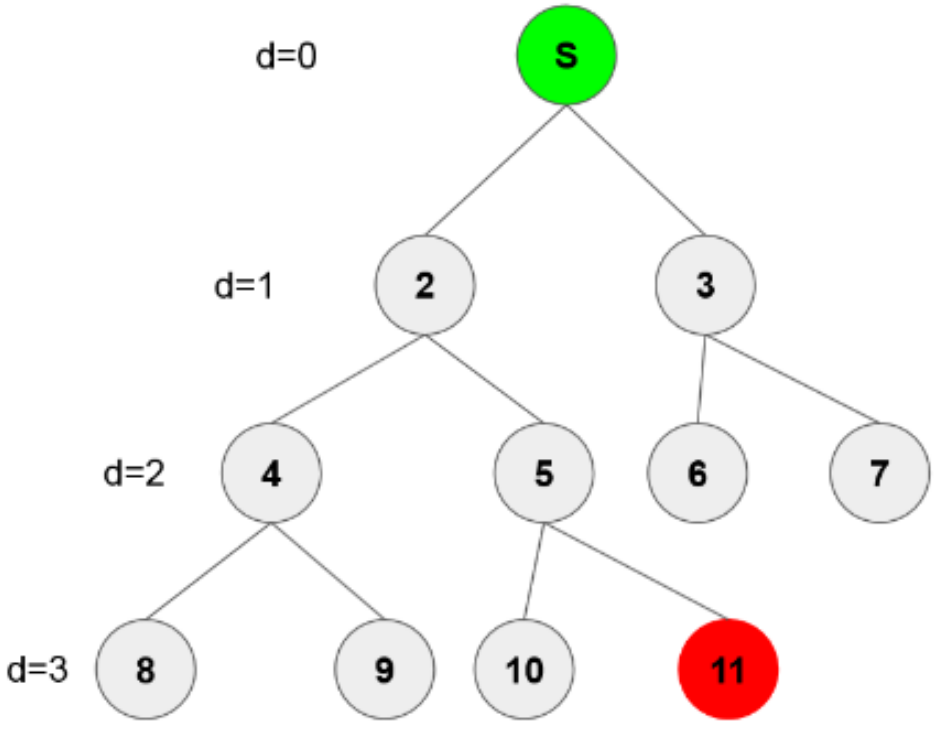




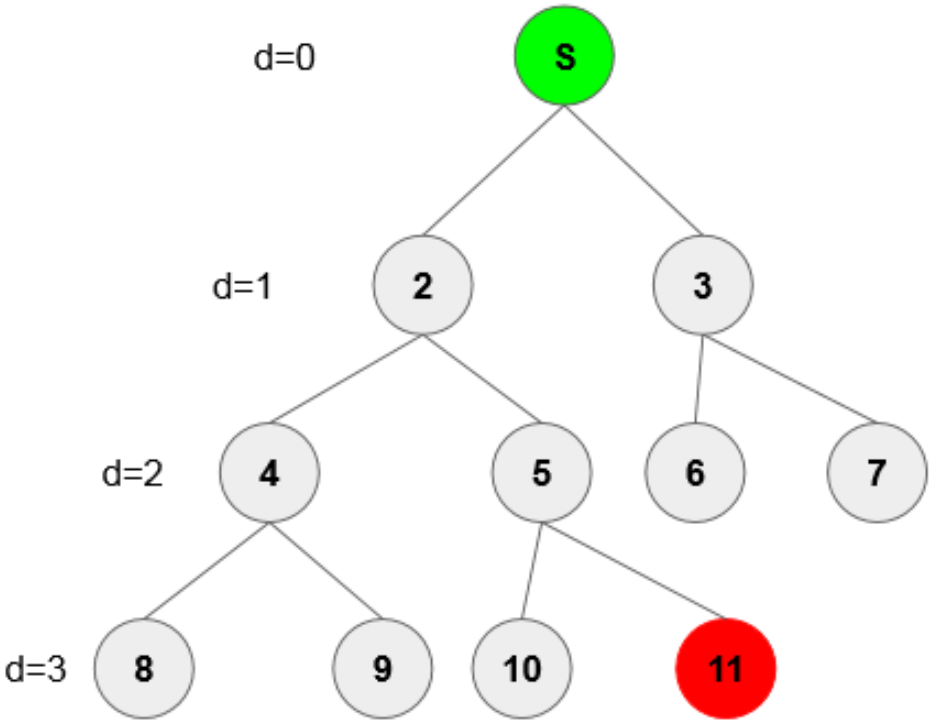
Amplitud



Amplitud



Amplitud



ABIERTA

	6	7	8	9

ABIERTA se implementa con una **COLA**

Características de la búsqueda en amplitud

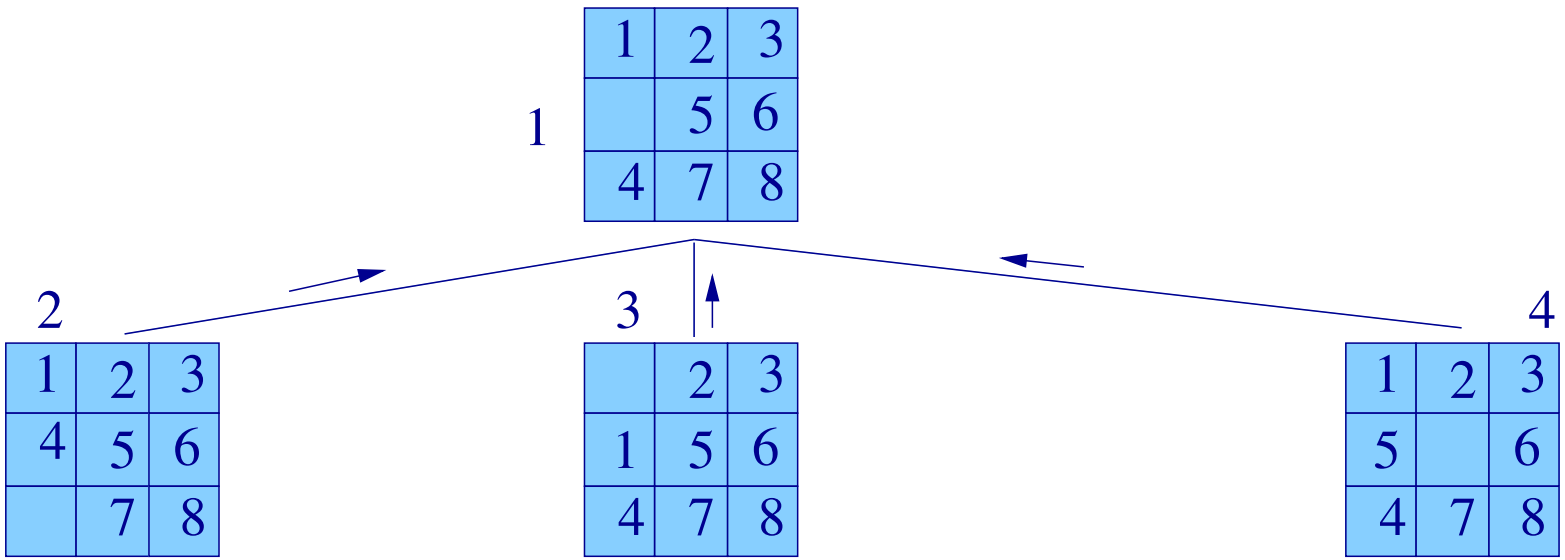
- **Completitud**: encuentra solución si existe y el factor de ramificación es finito en cada nodo
- **Optimalidad**: si todos los operadores tienen el mismo coste, encontrará la solución óptima
- **Eficiencia**: buena si las metas están cercanas
- Problema: consume memoria exponencial

Profundidad

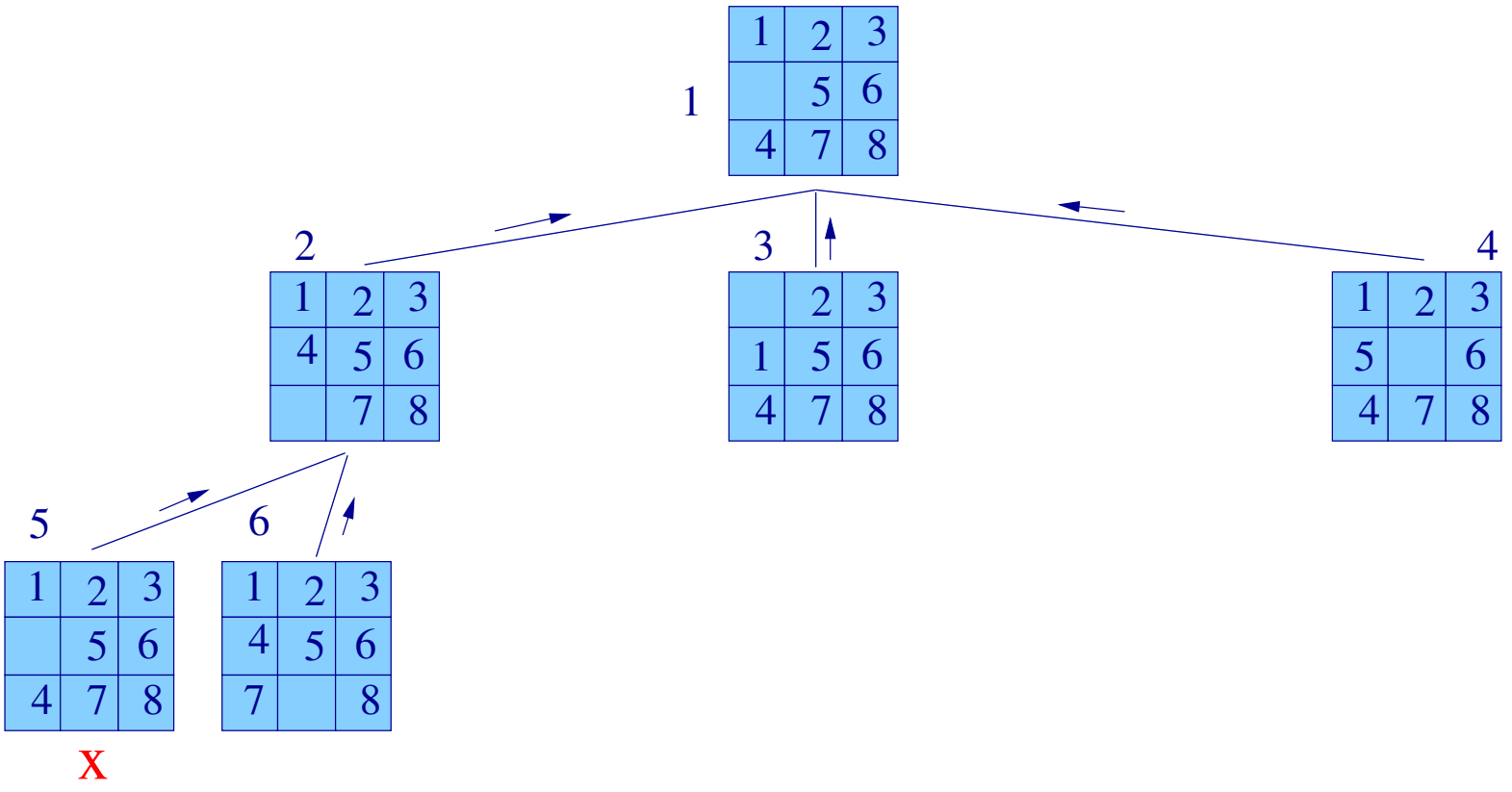
"Se expande el primero de los nodos recién generados hasta que se encuentra una solución..."

"Se expande el primero de los nodos recién generados hasta que se encuentra una solución o se ha alcanzado una profundidad máxima d_{max}"

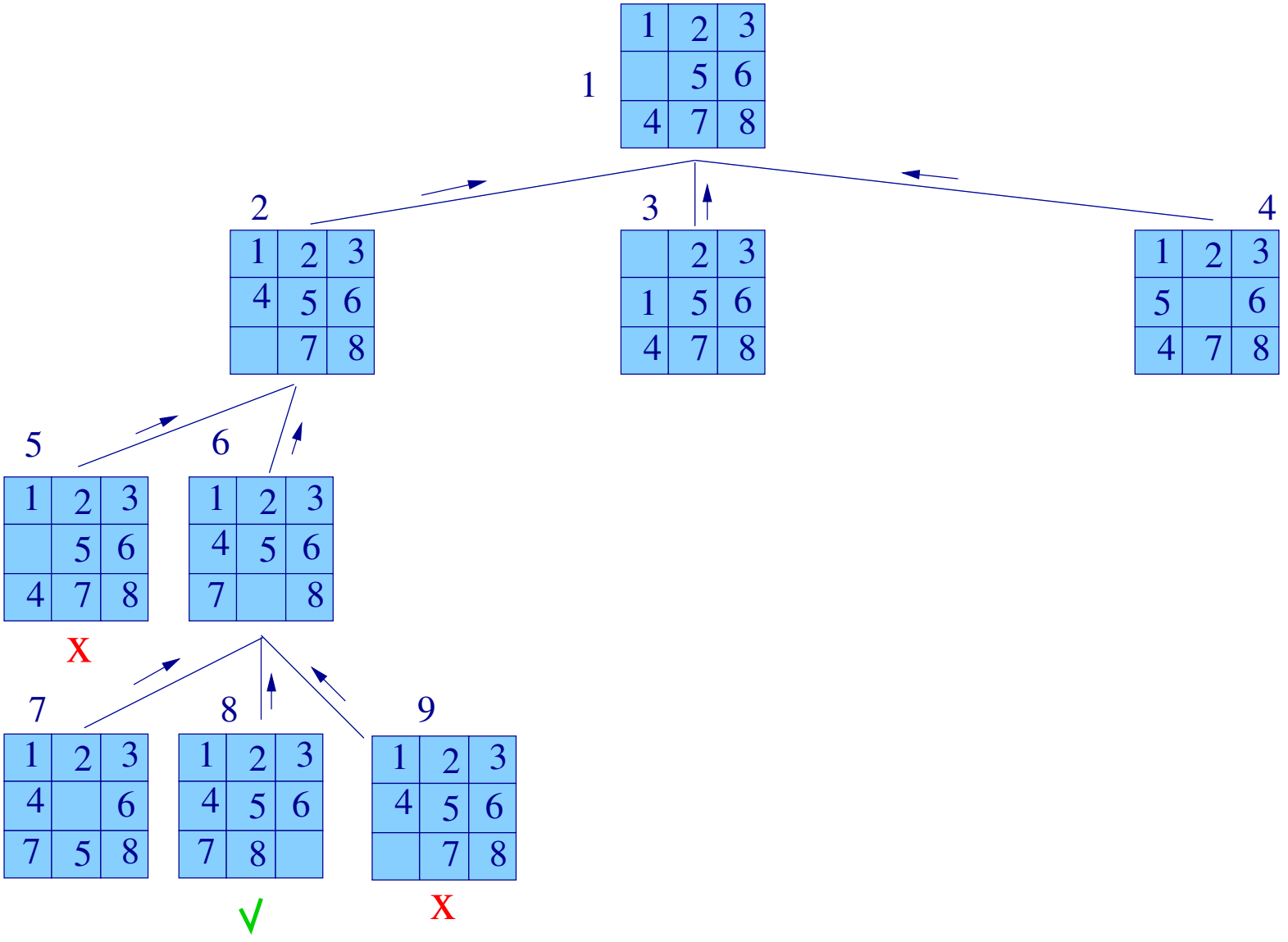
8-Puzle – Profundidad



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

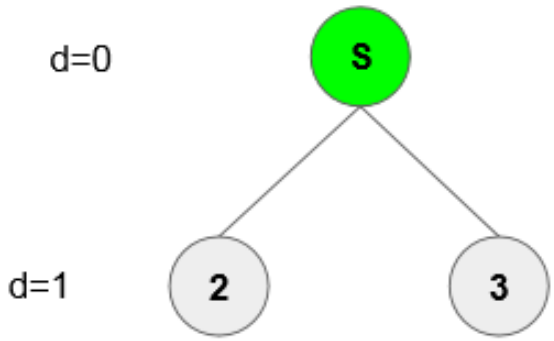
Profundidad

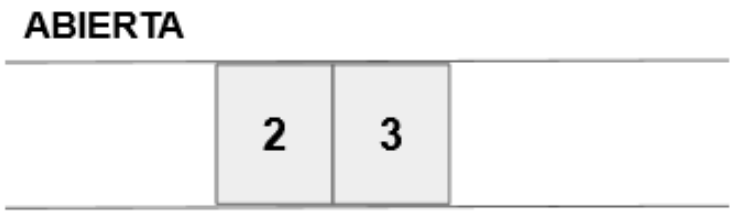
$d_{max} = 3$

d=0

Profundidad

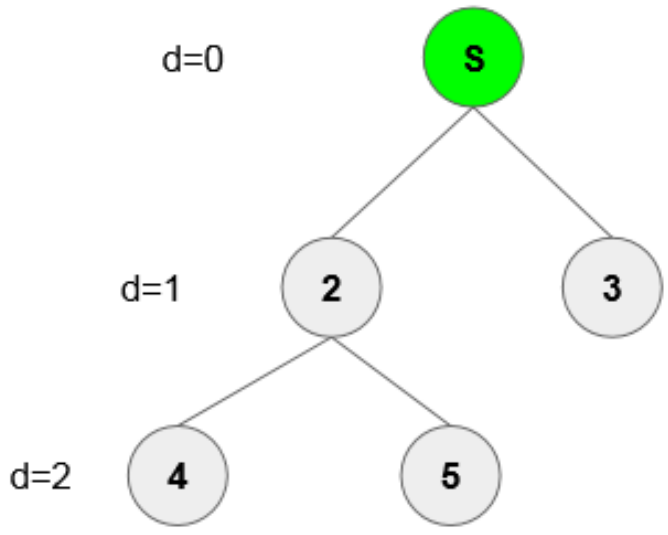
$$d_{max} = 3$$

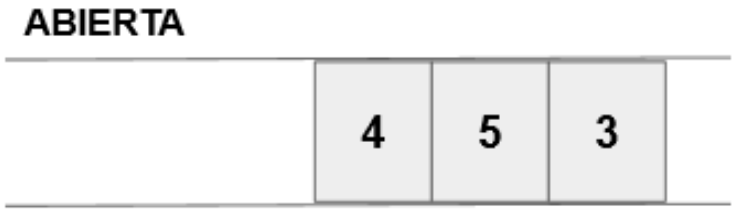




Profundidad

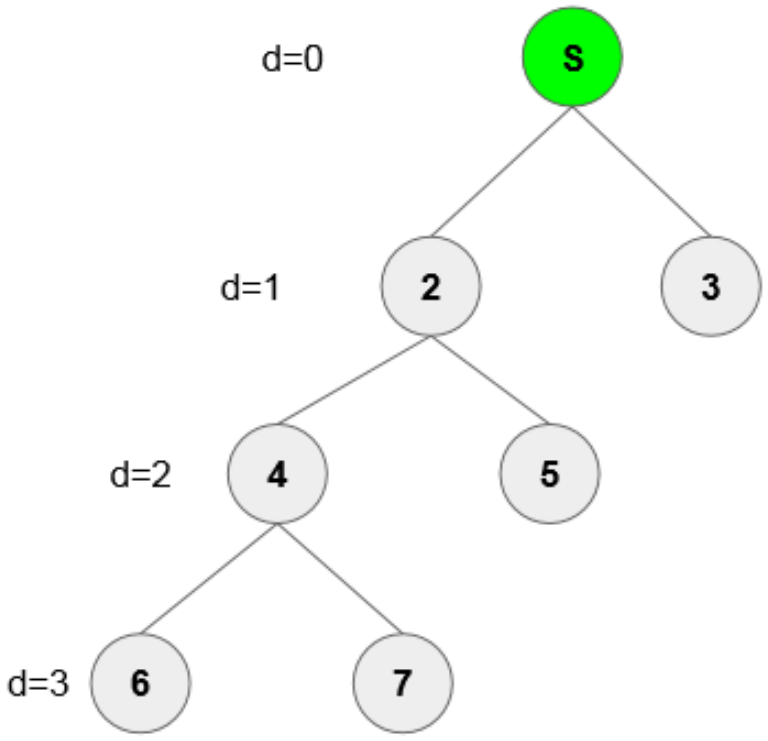
$$d_{max} = 3$$





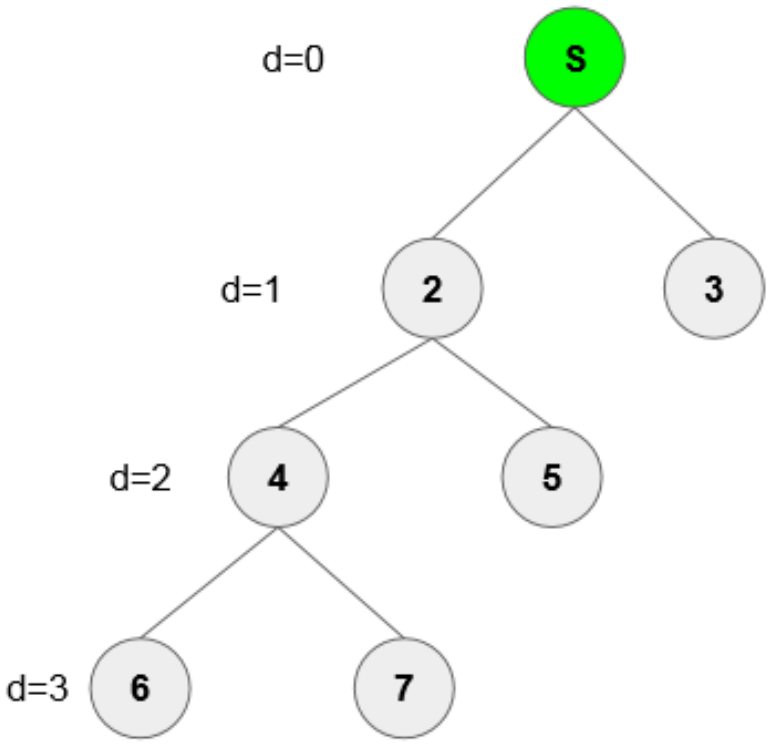
Profundidad

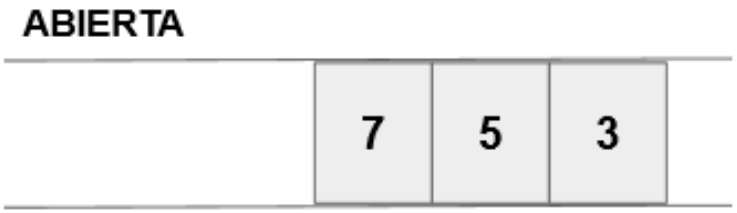
$$d_{max} = 3$$



Profundidad

$$d_{max} = 3$$

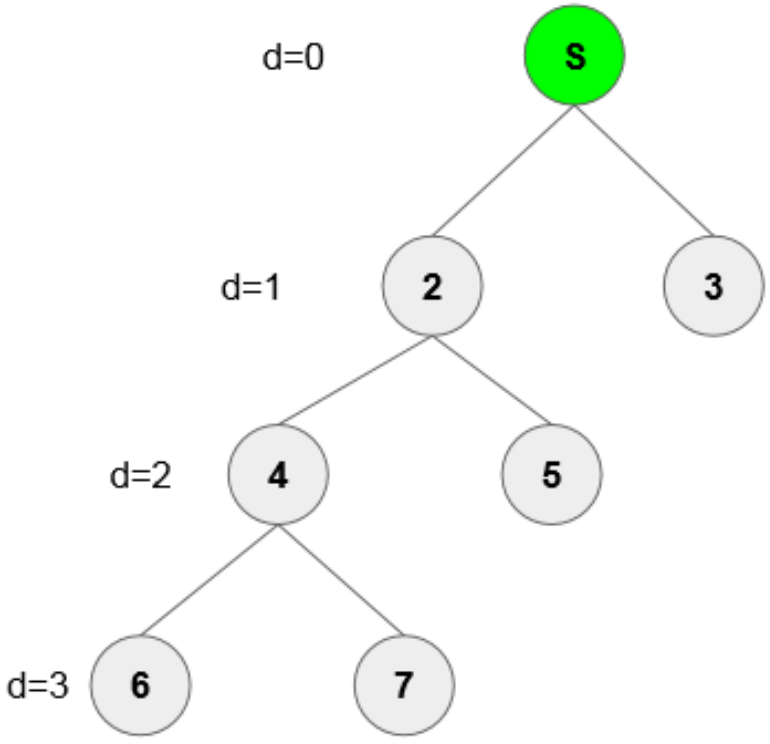




La expansión del nodo 6 no genera nada por el límite de profundidad

Profundidad

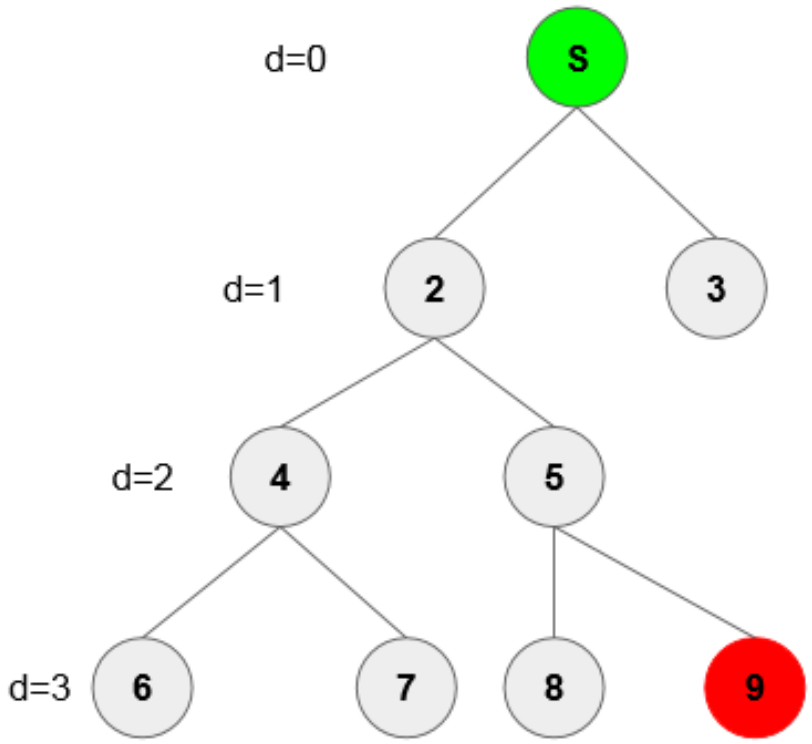
$$d_{max} = 3$$



La expansión del nodo 7 tampoco genera nada por el límite de profundidad

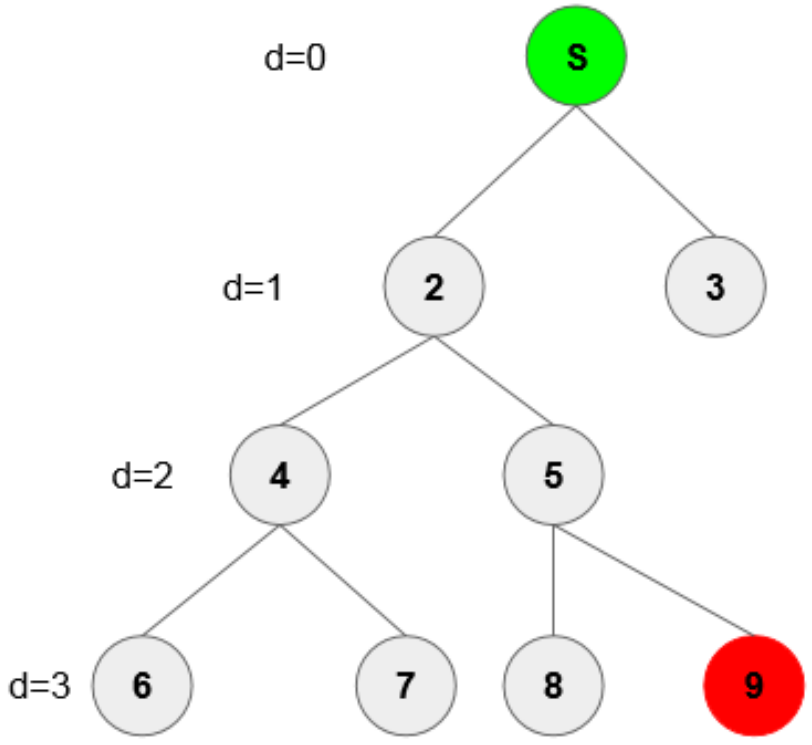
Profundidad

$$d_{max} = 3$$



Profundidad

$$d_{max} = 3$$



ABIERTA

	3

ABIERTA se implementa con una **PILA**

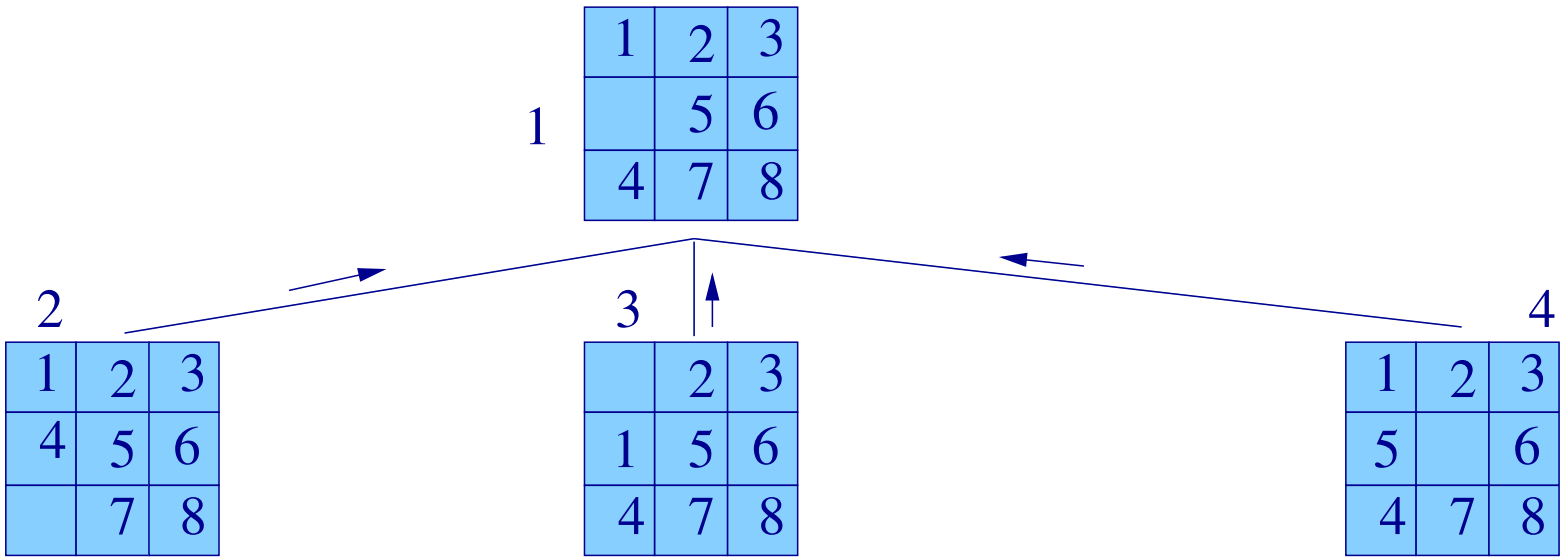
Características de la búsqueda en profundidad

- Requiere técnica de retroceso ("backtracking") que se implementa directamente al utilizar una PILA como estructura de datos
- 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

8-Puzle – Profundidad Iterativa

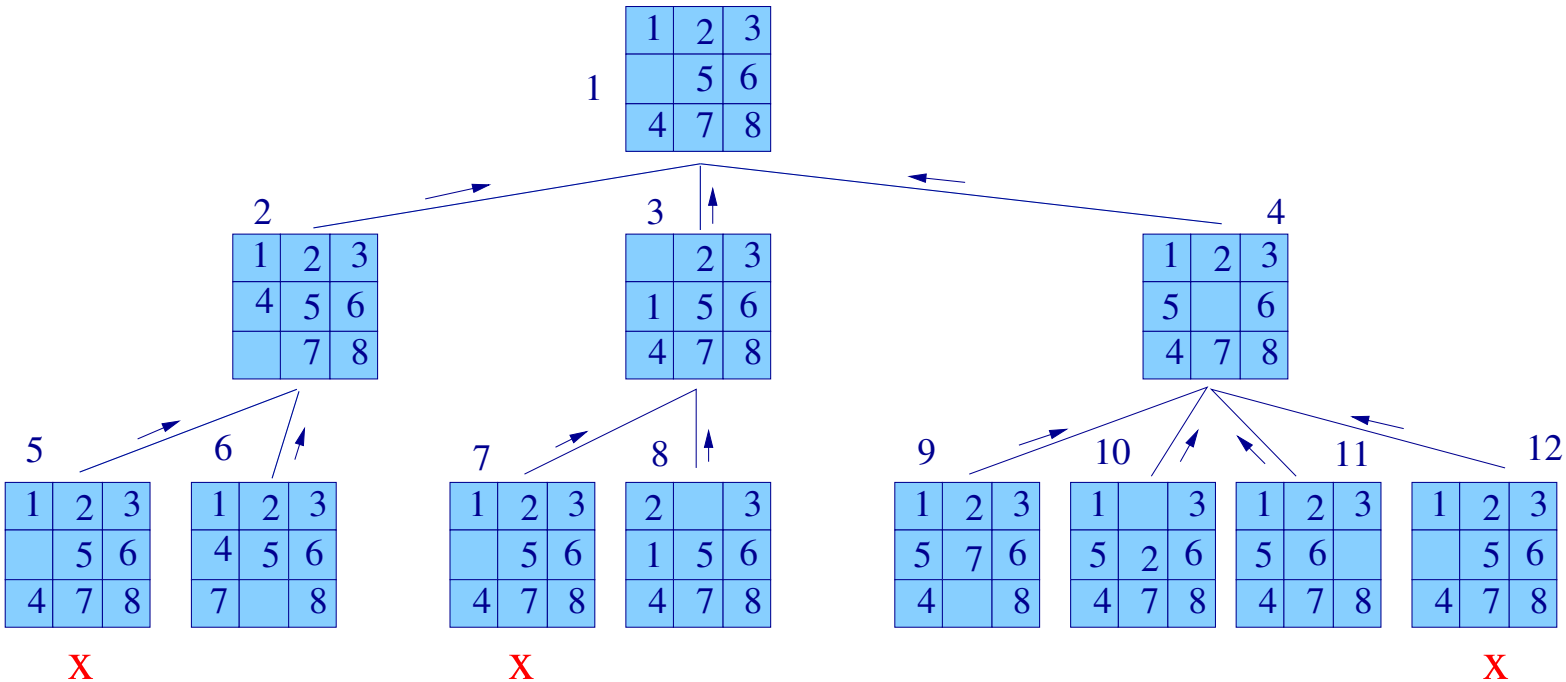
"Consiste en una serie de exploraciones en profundidad donde d_{max} se incrementa en k en cada iteración"

$d_{max} = 1,\ k = 1$



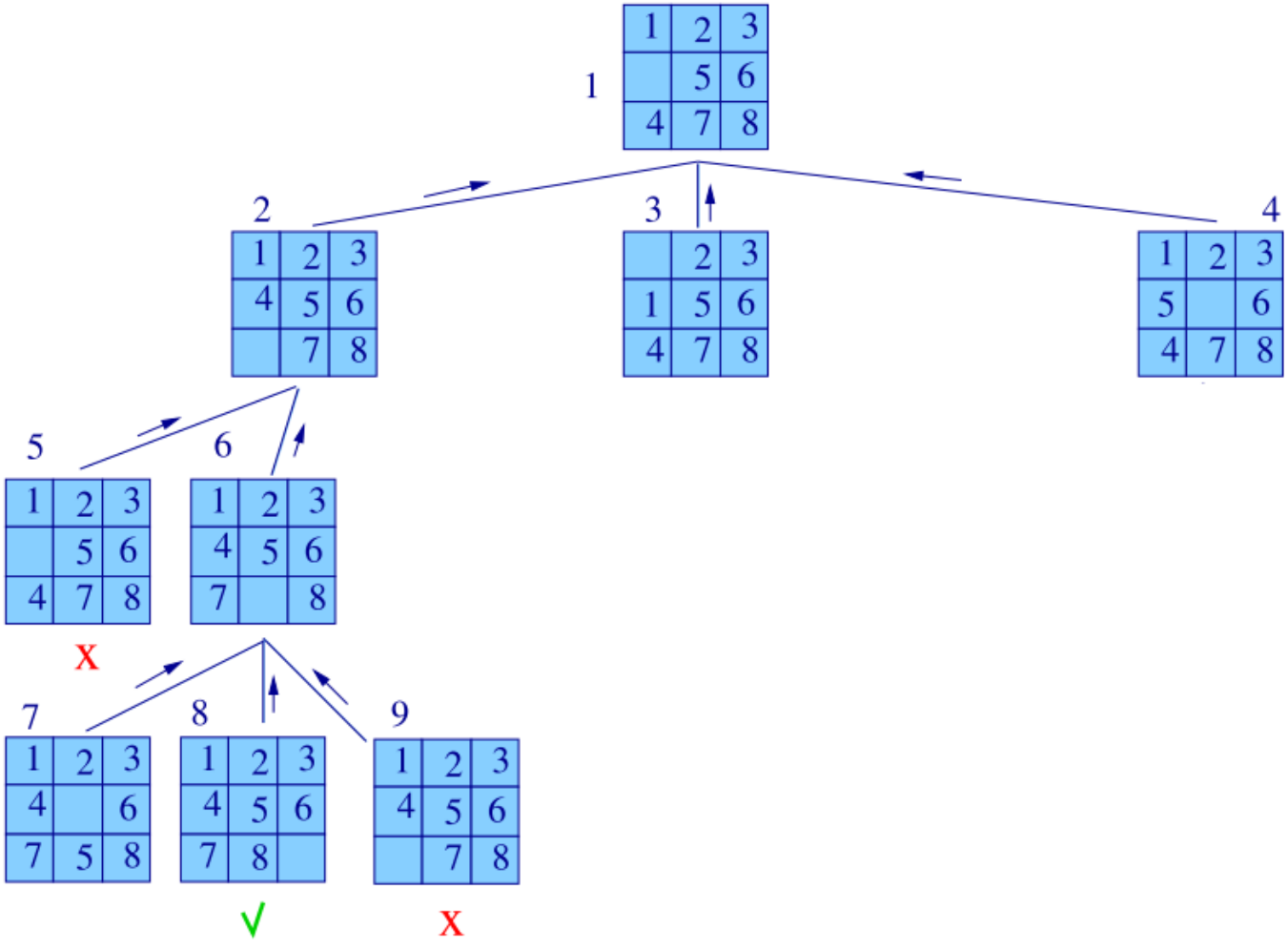
8-Puzle – Profundidad Iterativa

$d_{max} = 2,\ k = 1$



8-Puzle – Profundidad Iterativa

$d_{max} = 3,\ k = 1$



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

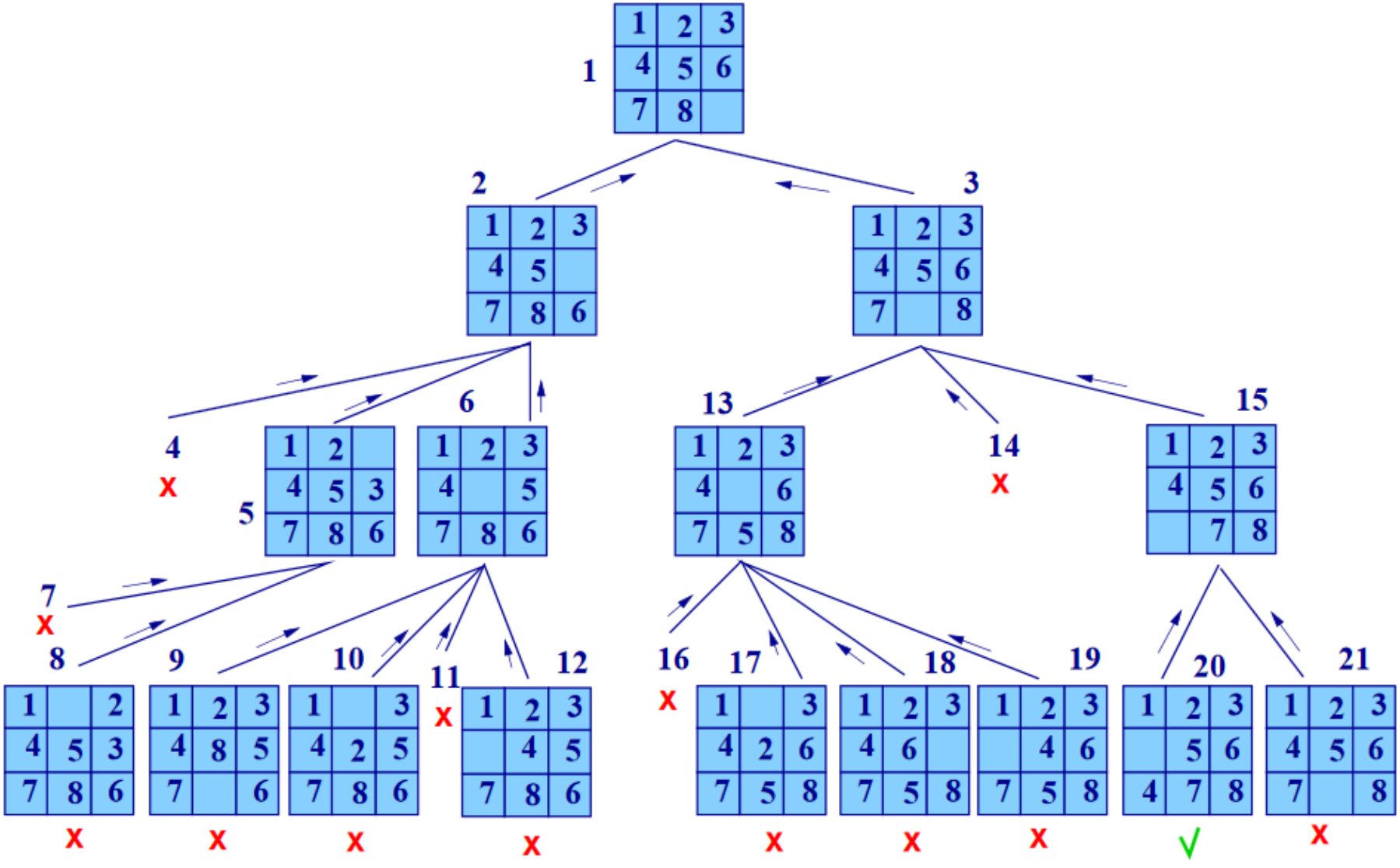
Características de la búsqueda en profundidad iterativa

- **Completitud**: encuentra la solución, si ésta existe
- **Admisibilidad**: encuentra la solución óptima, si $d_{max} = k = 1$
- Problema: puede generar muchos nodos duplicados

Búsqueda hacia atrás

- Problemas búsqueda hacia adelante:
  - Número elevado de de acciones aplicables en cada estado
  - Factor de ramificación demasiado grande
  - No resulta viable para planes con muchos pasos
- Una posible alternativa:
  - Buscar hacia atrás, partiendo del objetivo...
  - Puede ser en amplitud o profundidad
  - Por lo general, se consigue reducir el factor de ramificación

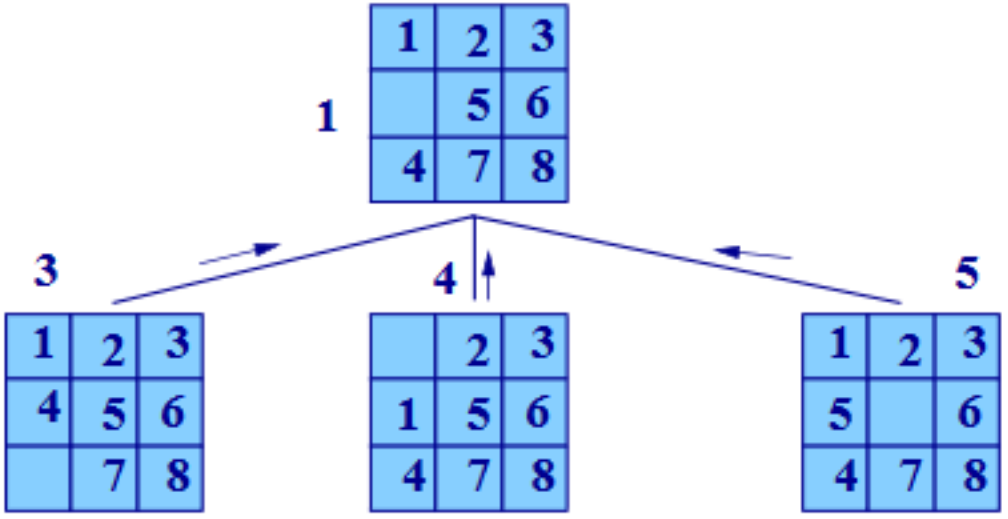
Búsqueda hacia atrás

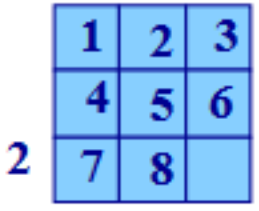


Búsqueda hacia atrás

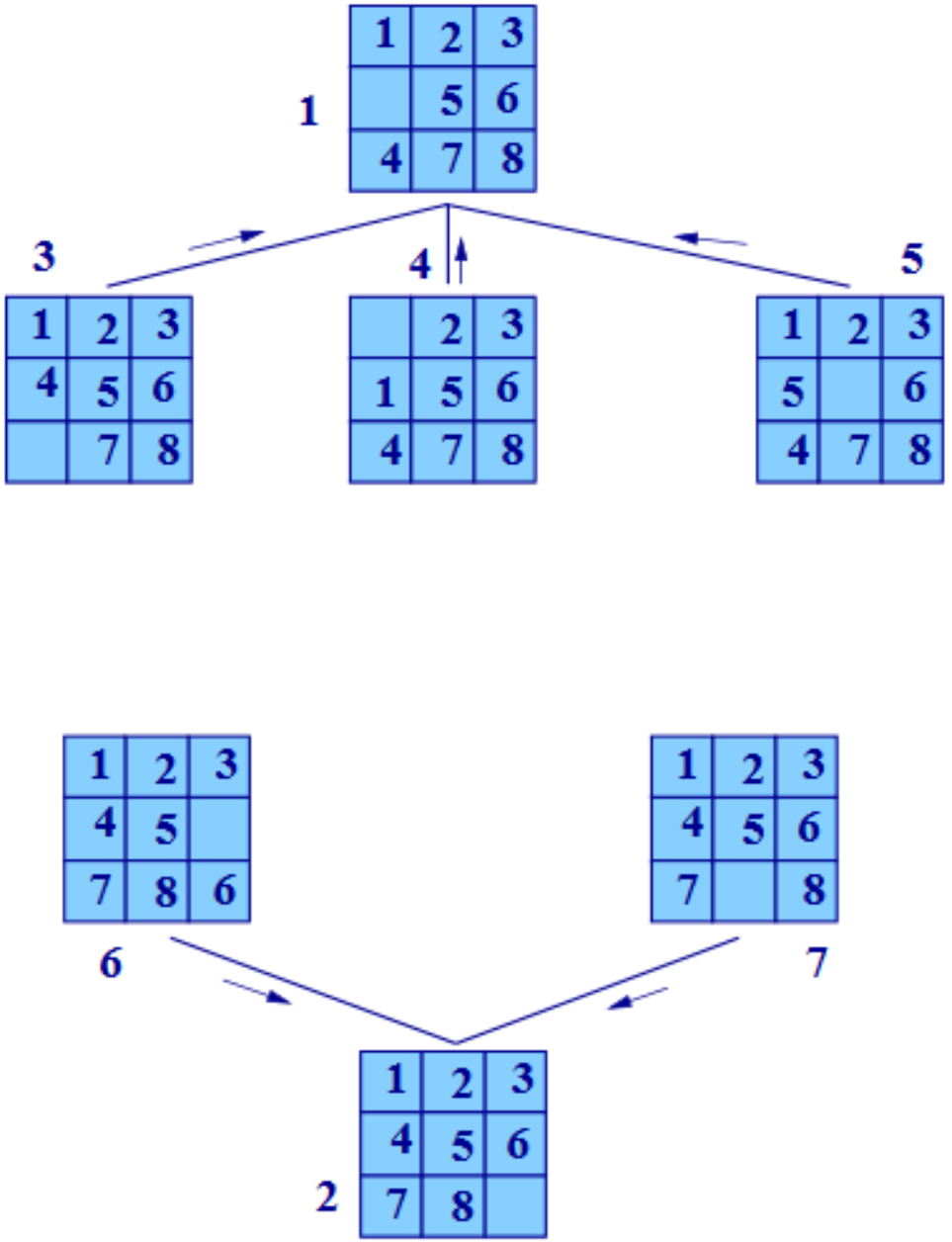
- Requisitos:
  - Operadores inversos (simetrías)
  - Estados iniciales sean alcanzables desde la(s) meta(s)
  - Se pueda generar algún estado final o meta
- Depende de:
  - factor de ramificación
  - número de estados iniciales y metas

Búsqueda bidireccional

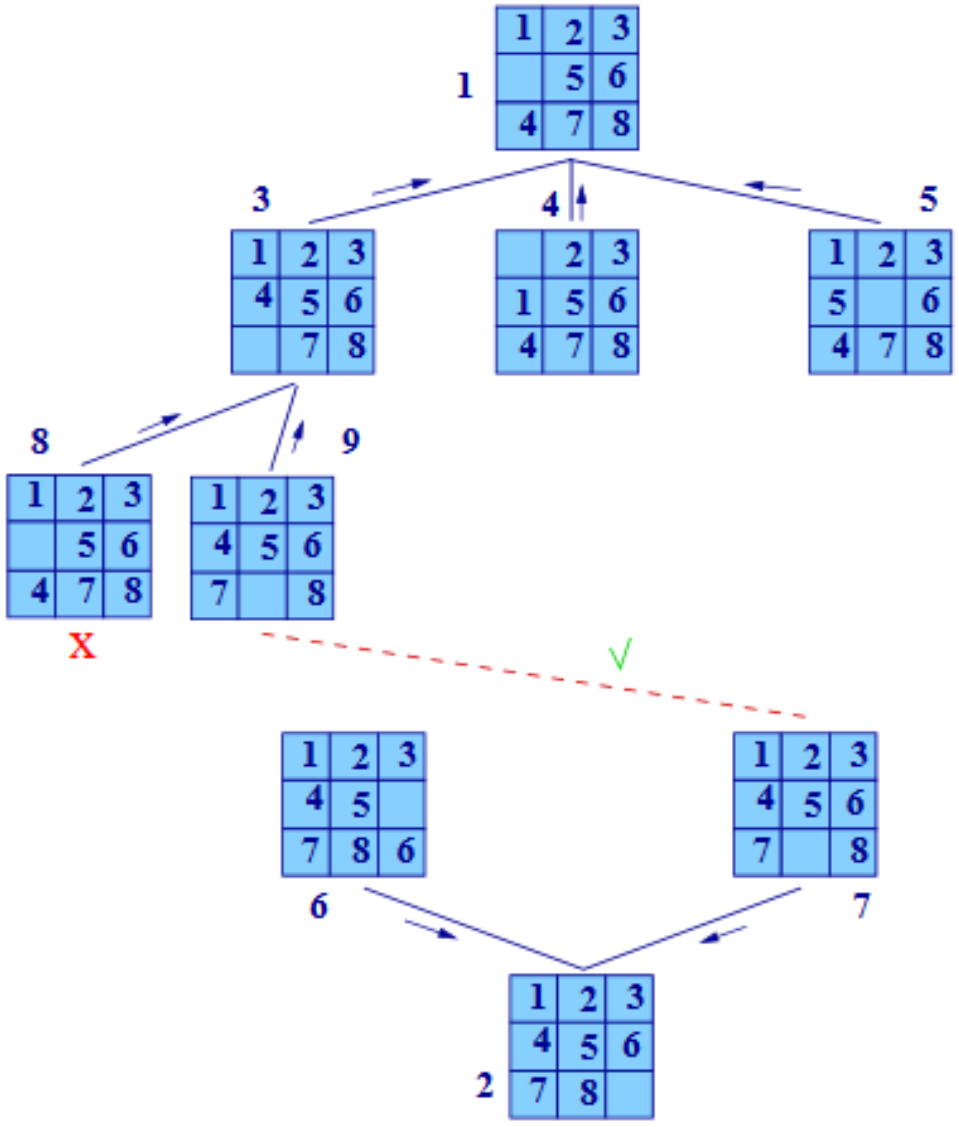




Búsqueda bidireccional



Búsqueda bidireccional



Búsqueda bidireccional

Procedimiento Bidireccional (Estado-inicial Estado-final)

1. ABIERTA-I=Estado-inicial, ABIERTA-M=Estado-final, EXITO=Falso
2. Hasta que alguna de ABIERTA-I o ABIERTA-M estén vacías O EXITO

   Quitar de ABIERTA-I el primer nodo, N
   Generar sucesores de N, introducirlos en ABIERTA-I
   SI algún sucesor de N equipara con algún elemento de ABIERTA-M
   ENTONCES EXITO=Verdadero
   SI NO Quitar de ABIERTA-M el primer nodo M
      Generar sucesores de M, introducirlos en ABIERTA-M
      SI algún sucesor de M equipara con algún elemento de ABIERTA-I
      ENTONCES EXITO=Verdadero
3. Si EXITO
   Entonces Solución=camino desde nodo del Estado-inicial al nodo del Estado-final por los punteros
   Si no, Solución=fracaso

Búsqueda bidireccional

- Completitud: es completo si ambas búsquedas lo son
- Admisibilidad: es admisible si ambas búsquedas lo son
- Eficiencia: reduce el tamaño total del árbol de búsqueda
- Problema: la comprobación de colisión debe implementarse muy eficientemente

Análisis de complejidad

Análisis de complejidad: Problema

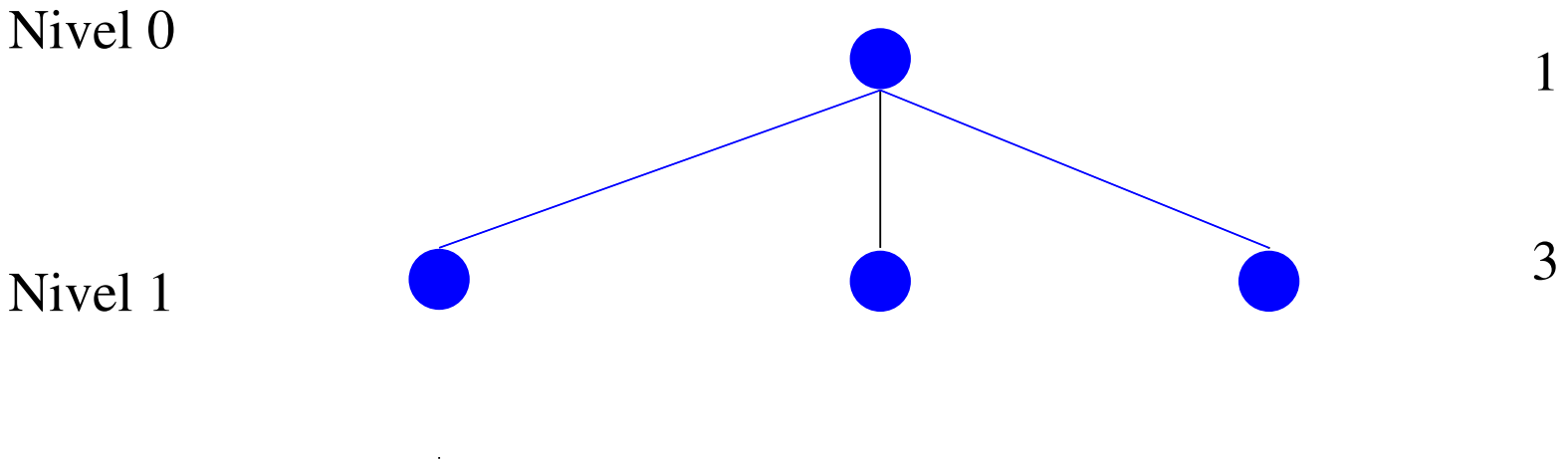
- Si se dispone de:
  - factor de ramificación medio, b
  - profundidad del árbol de búsqueda, d
- ¿Cuál sería, en el peor de los casos, el número de nodos que examinaría cada técnica?
  - Primero en Amplitud (??)
  - Primero en Profundidad (??)
  - Primero en Profundidad Iterativo (??)
  - Bidireccional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

Ejemplo

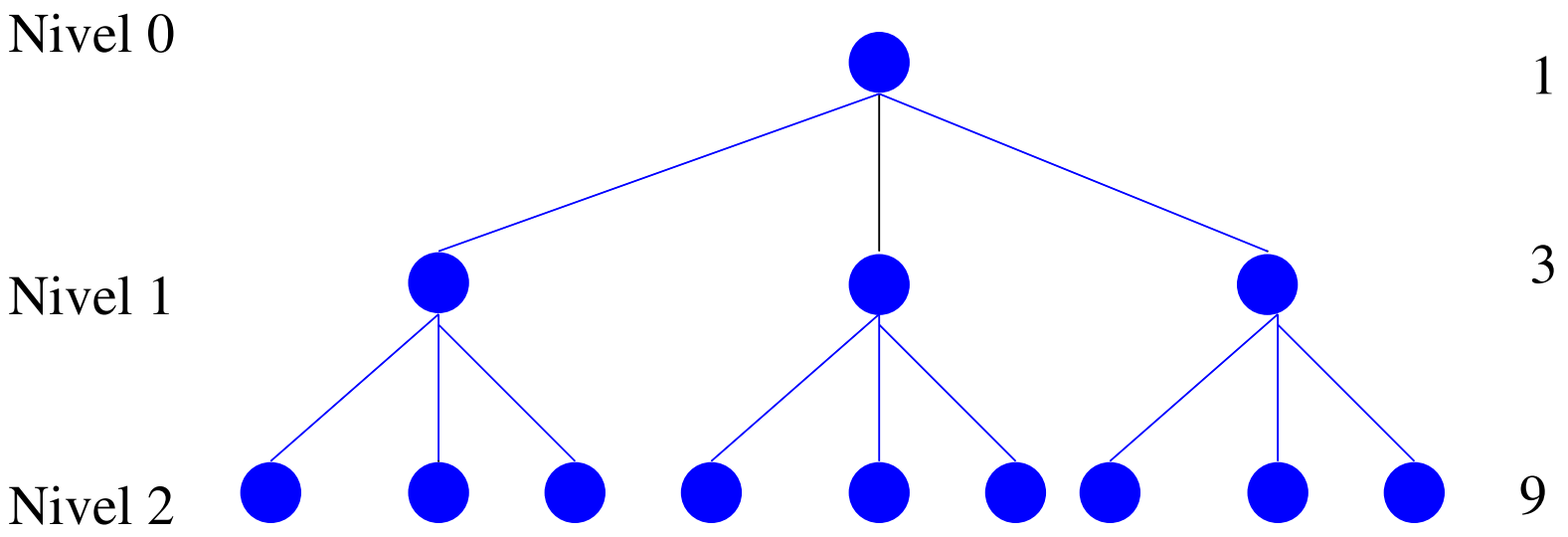
Nivel 0

1

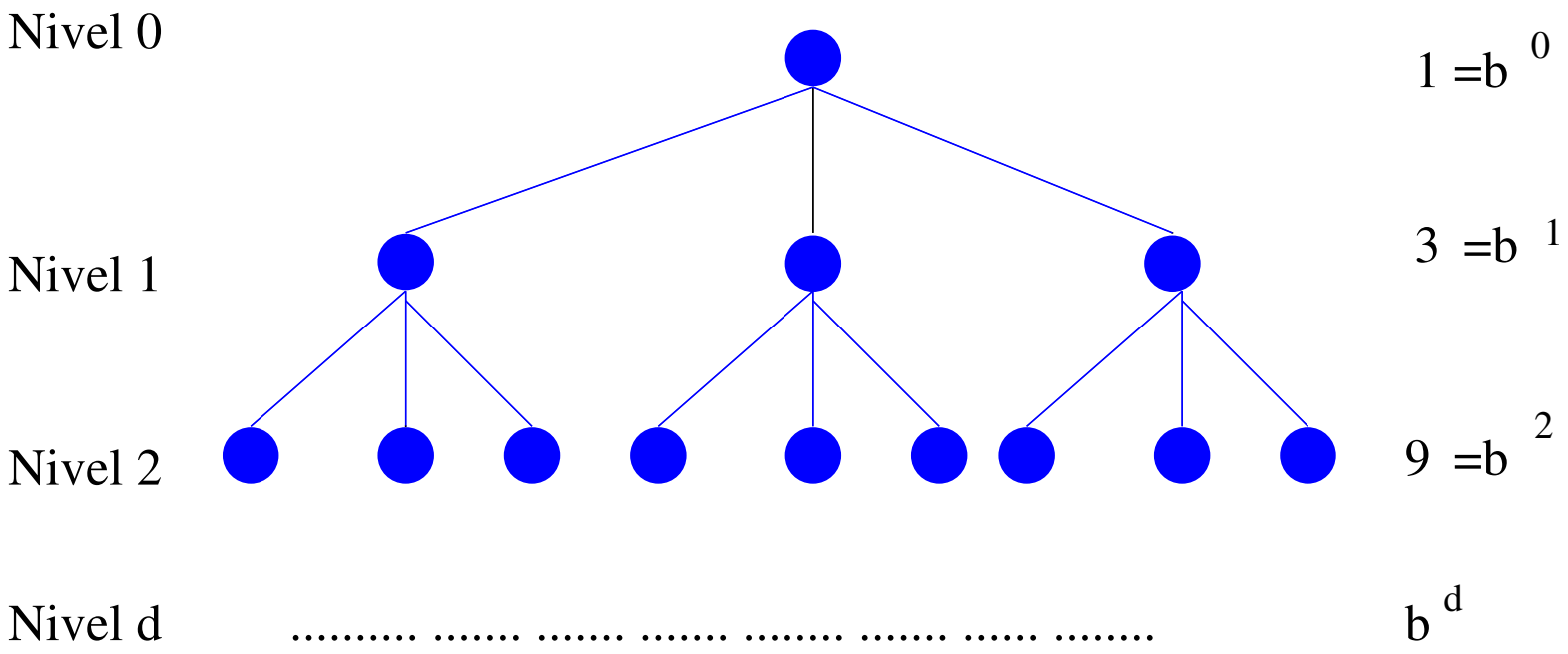
Ejemplo



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	$\sum_{i=0}^{d} (d-i) b^i$
Bidireccional	$2 \sum_{i=0}^{\frac{d}{2}} b^i$

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	$O(b^d)$	$O(d)$
Bidireccional	$O(b^{\frac{d}{2}})$	$O(b^{\frac{d}{2}})$

- Bidireccional es la búsqueda más rápida
- Profundidad es la que menos memoria requiere, y además profundidad iterativa garantiza completitud y adminisibilidad bajo ciertos criterios

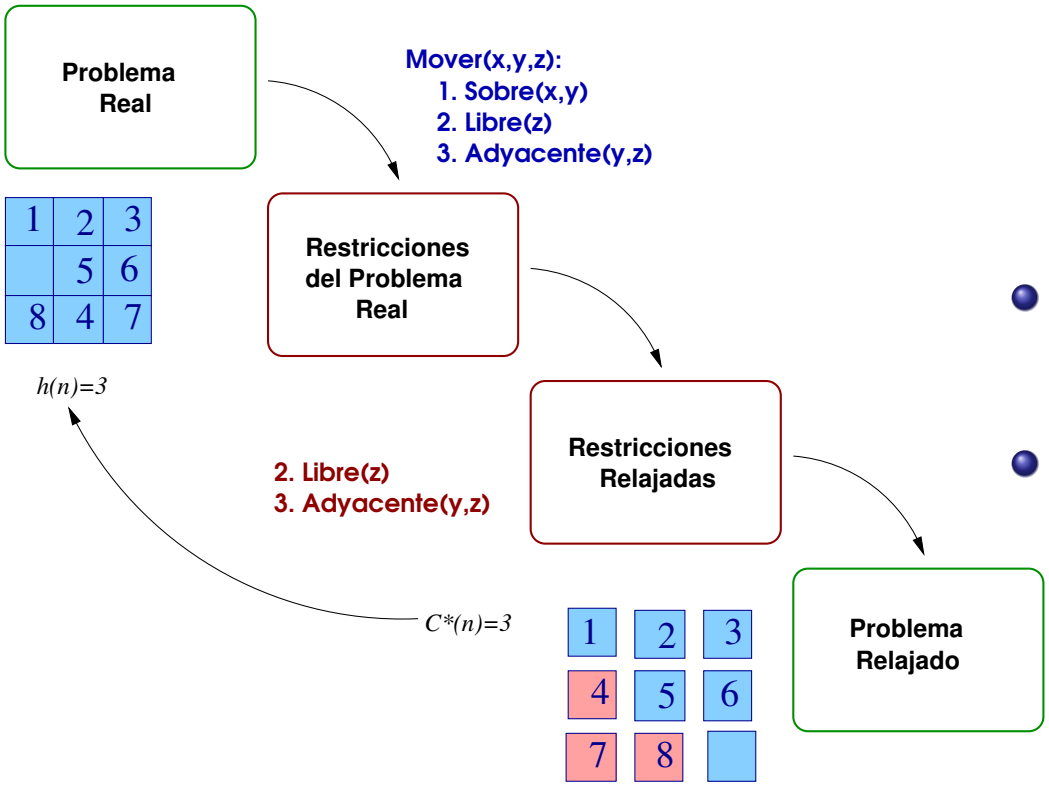
Búsqueda heurística

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
  - Funciones $h(n, t)$: Devuelve una estimación del coste del mejor camino desde el nodo n al nodo t
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Relajación de restricciones

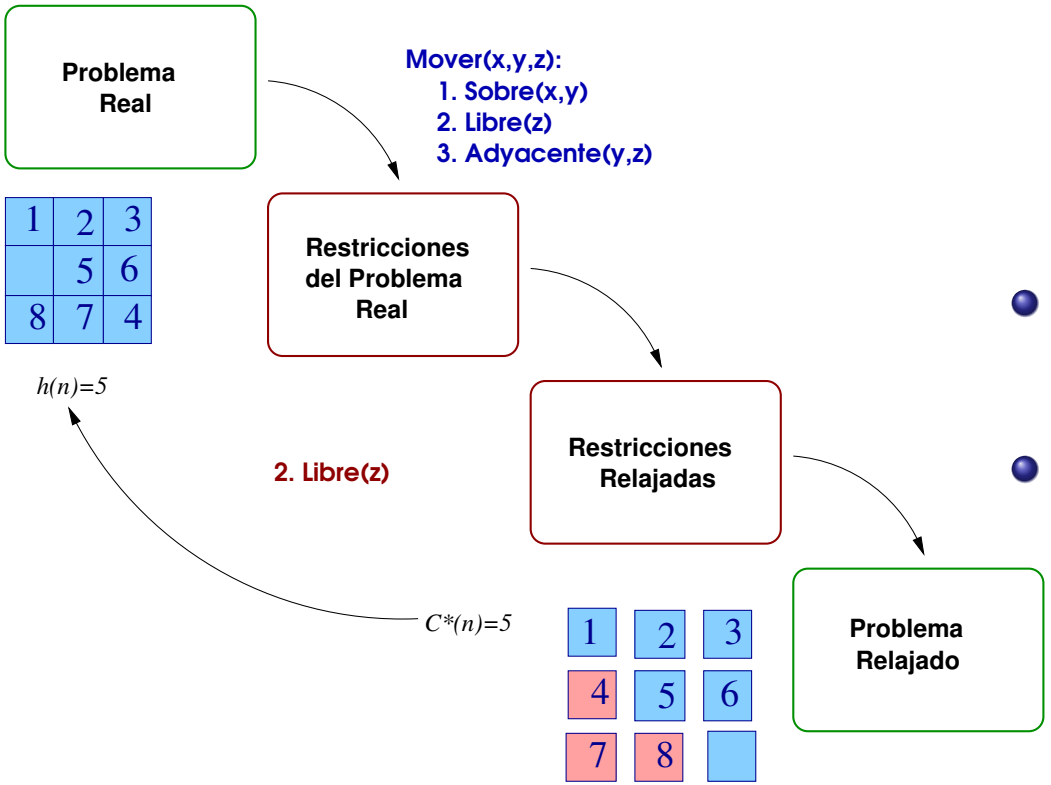
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



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

Relajación de restricciones

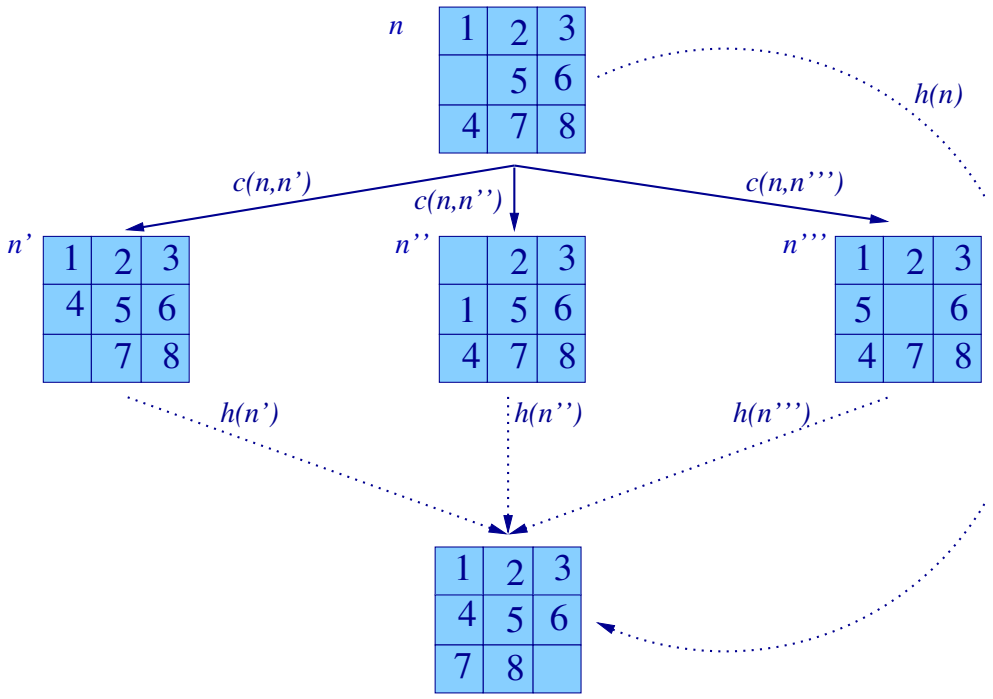
- Típicamente, las funciones heurísticas se obtienen por **relajación de las restricciones** del problema original



- $h(\cdot)$ 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*:

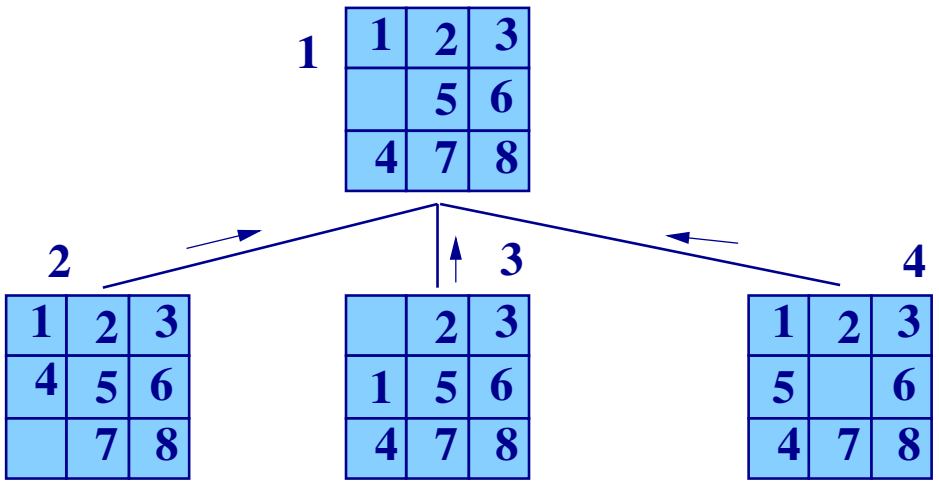
$$h(n) \leq h^*(n)$$

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**

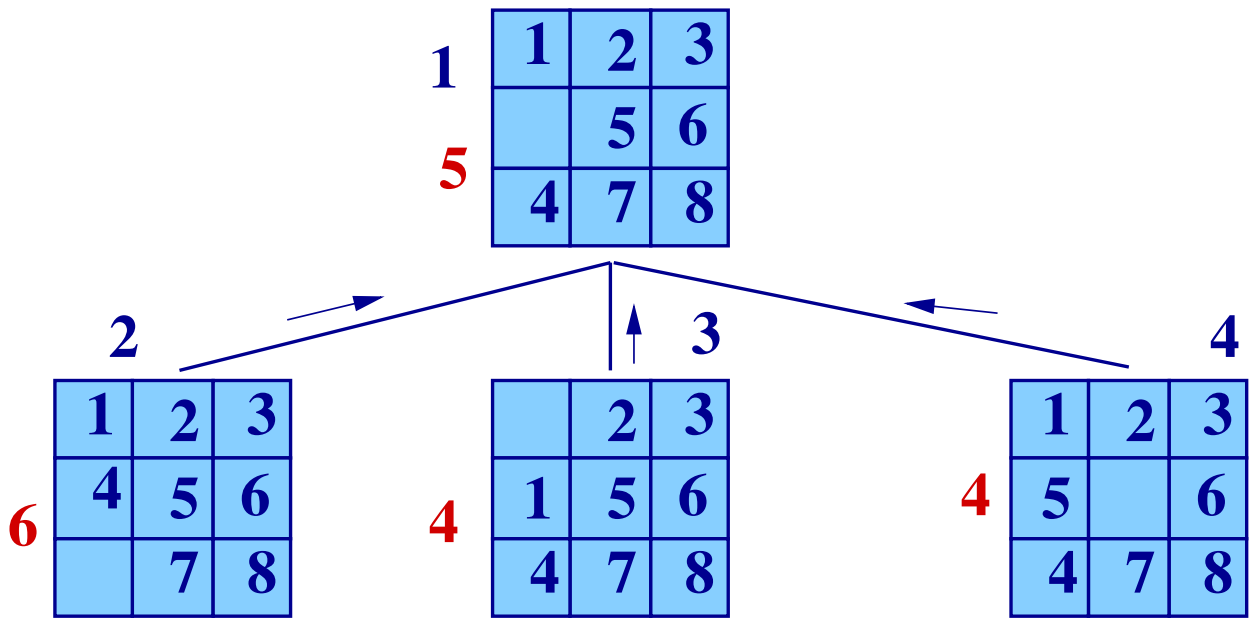
8-Puzle – Búsqueda en escalada

"Se escoge para su expansión el sucesor heurísticamente más prometedor descartando el resto de sucesores"

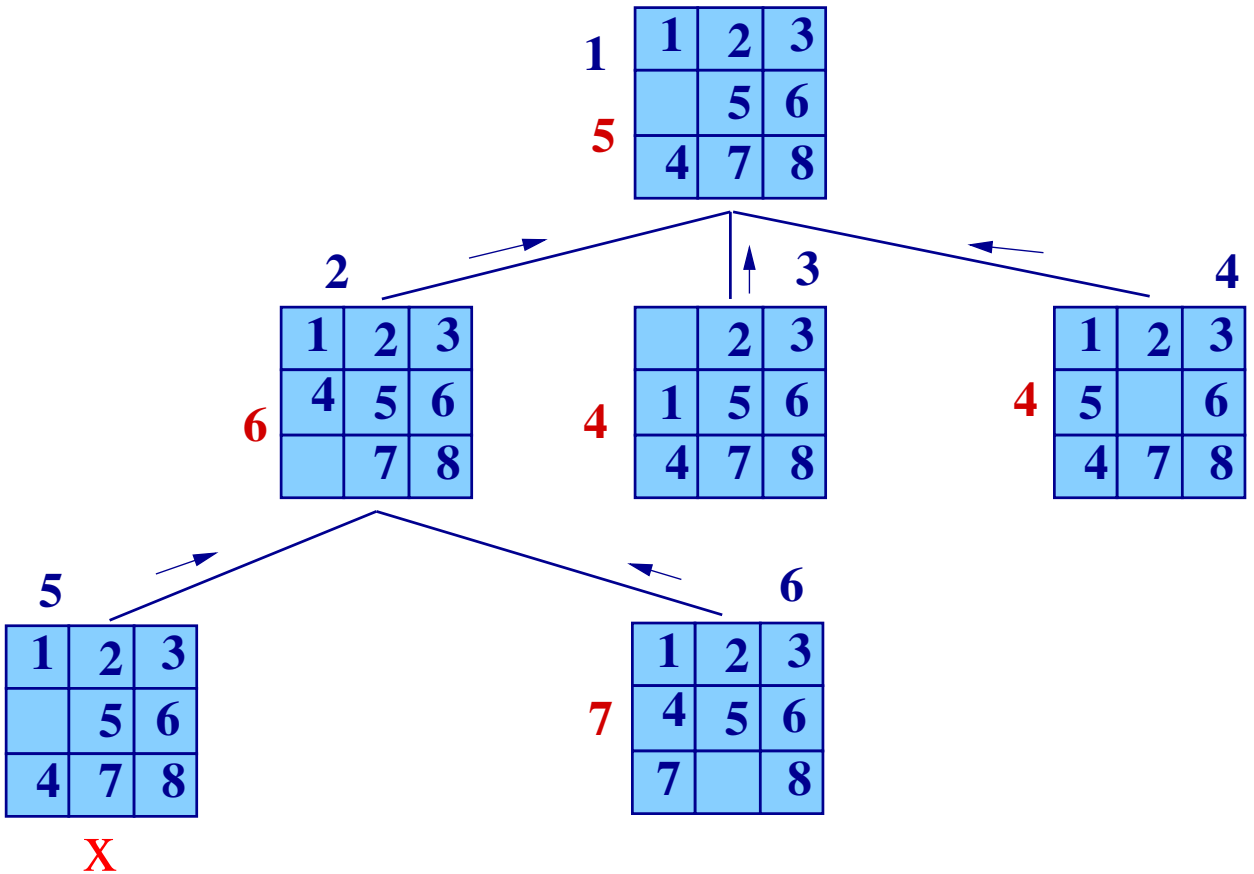


- Necesitamos una función de evaluación $f(n)$ que nos diga cuál es el mejor nodo
  - El valor más bajo (minimizar $f(n)$) utilizando una heurística que nos estime el camino hasta la meta: el número de casillas mal colocadas. En este caso $f(n) = h(n)$
  - El valor más alto (maximizar $f(n)$): El número de casillas bien colocadas (como en este ejemplo)

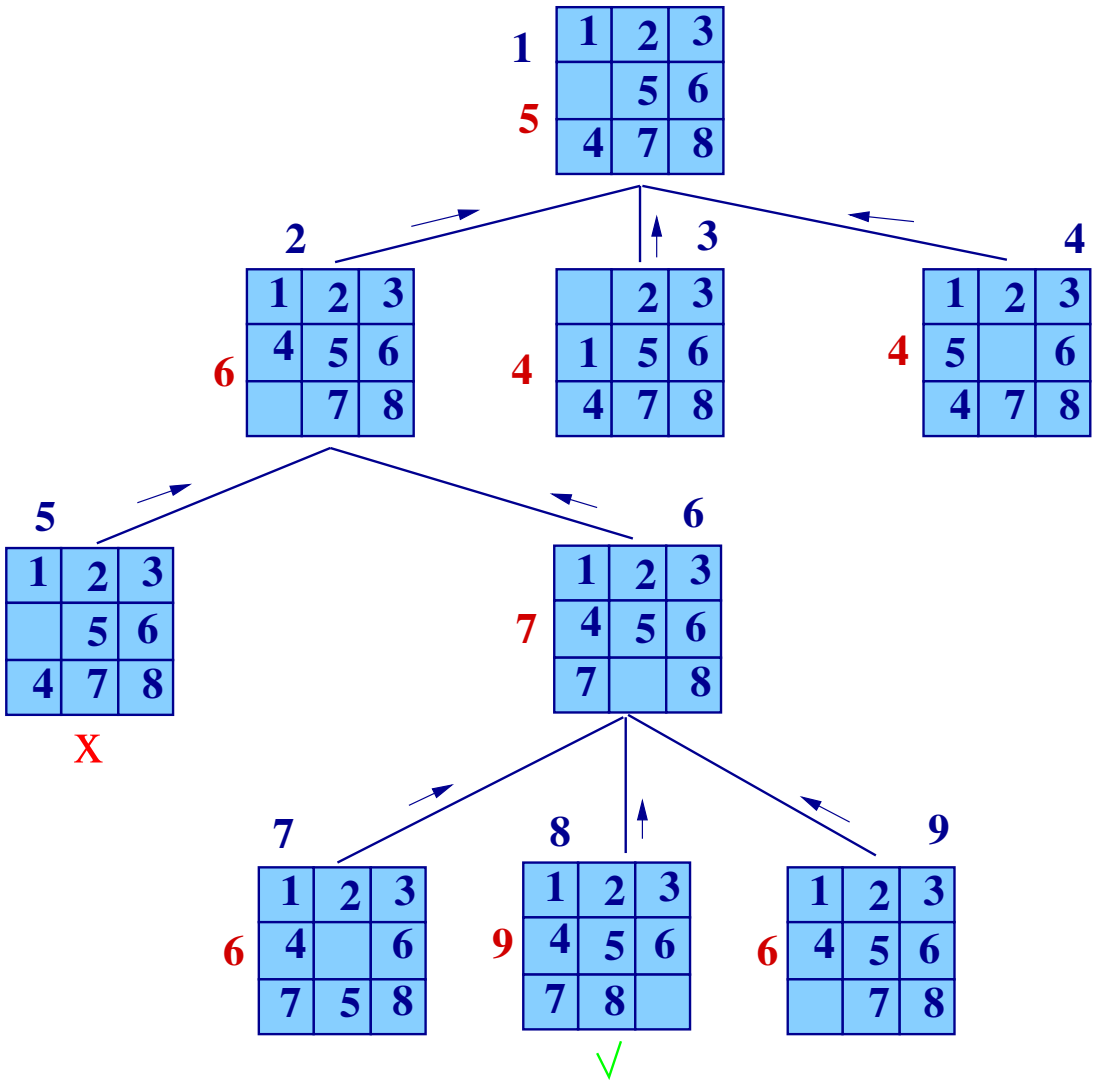
8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



8-Puzle – Búsqueda en escalada



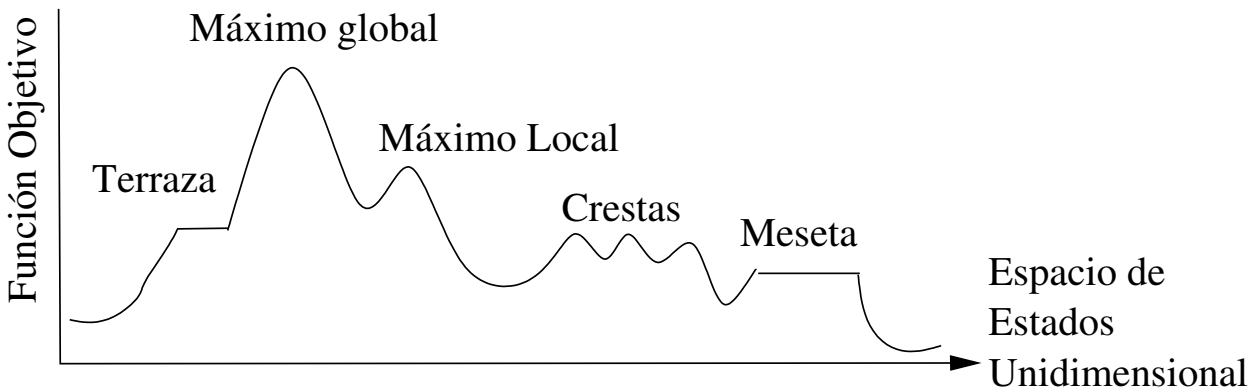
Búsqueda en escalada

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

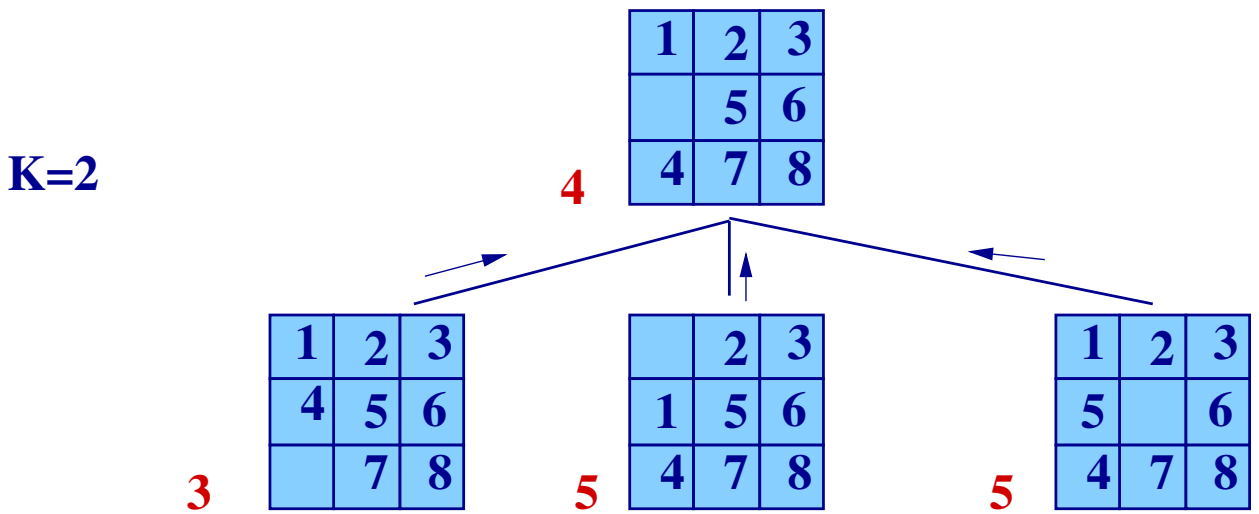


Características

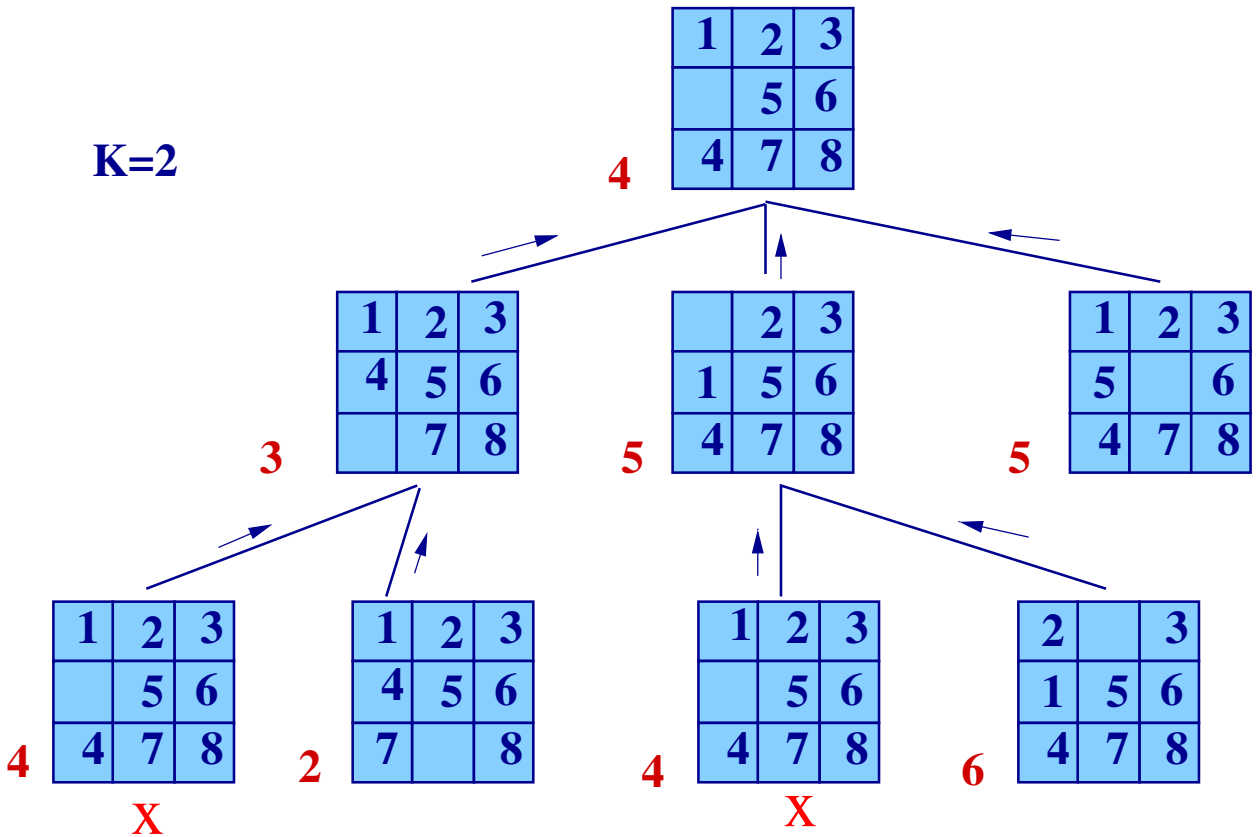
- Soluciones
  - Retroceso
  - Dar más de un paso
  - Reinicio aleatorio
- Método local
  - Completitud: no tiene porqué encontrar la solución
  - Admisibilidad: no siendo completo, aún menos será admisible
  - Eficiencia: rápido y útil si la función es monótona (de)creciente

8 Puzle – Búsqueda en haz

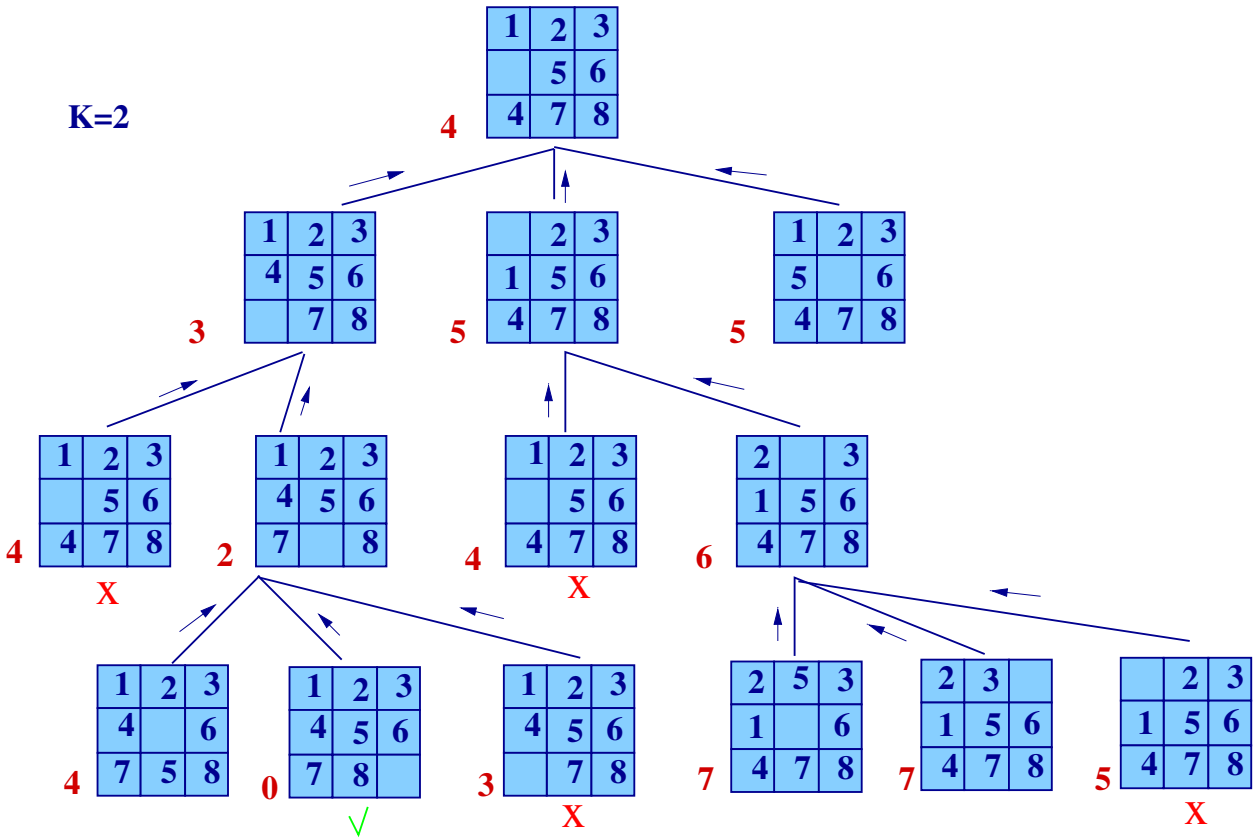
"En cada iteración se expanden SIMULTÁNEAMENTE los K sucesores más prometedores heurísticamente"



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"

- Considerar:
  - Lista ABIERTA: Contiene todos los nodos generados pendientes de ser expandidos, ordenados ascendentemente por $f(n)$
  - Lista CERRADA: Contiene todos los nodos que ya han sido expandidos (detección de duplicados)
  - TERMINACIÓN ocurre cuando procedemos a expandir el nodo meta o final

Búsqueda de "El mejor primero"

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

ABIERTA=Estado-inicial, CERRADA=Vacío, EXITO=Falso
Hasta que ABIERTA esta vacío O EXITO
  Quitar el primer nodo de ABIERTA, N, que no esté en CERRADA
  SI N es Estado-final ENTONCES EXITO=Verdadero
  SI NO Expandir N y meterlo en CERRADA, generando el conjunto S de sucesores de N
    Para cada sucesor s en S
      Se inserta s en orden según f(n) en ABIERTA
Si EXITO Entonces Solución=camino desde N a I a través de los punteros
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

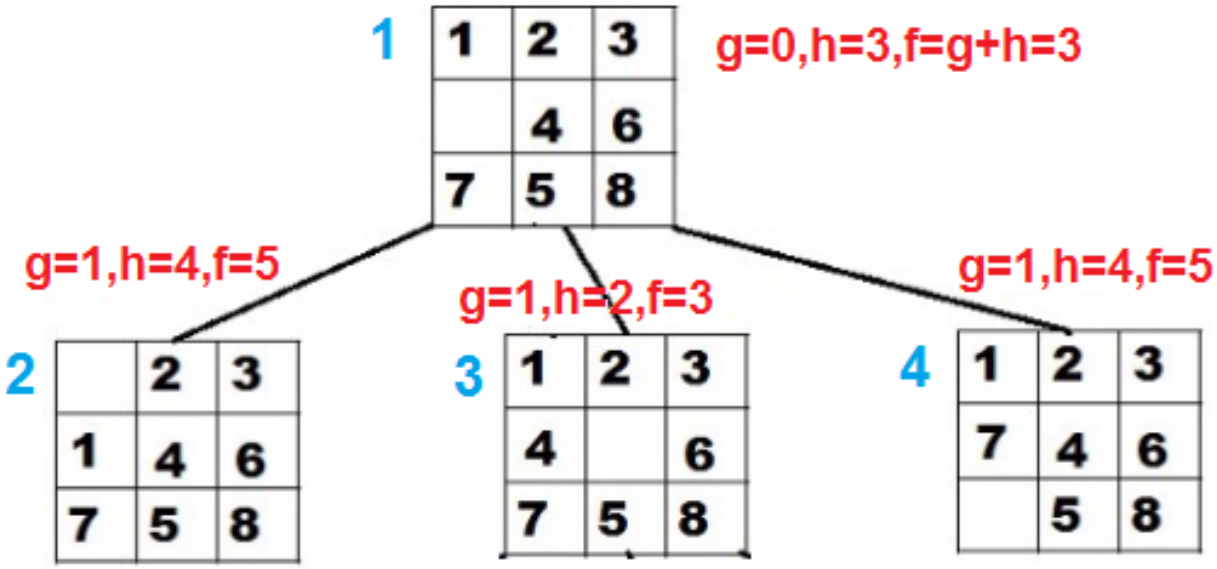
Ejemplo – A*

1

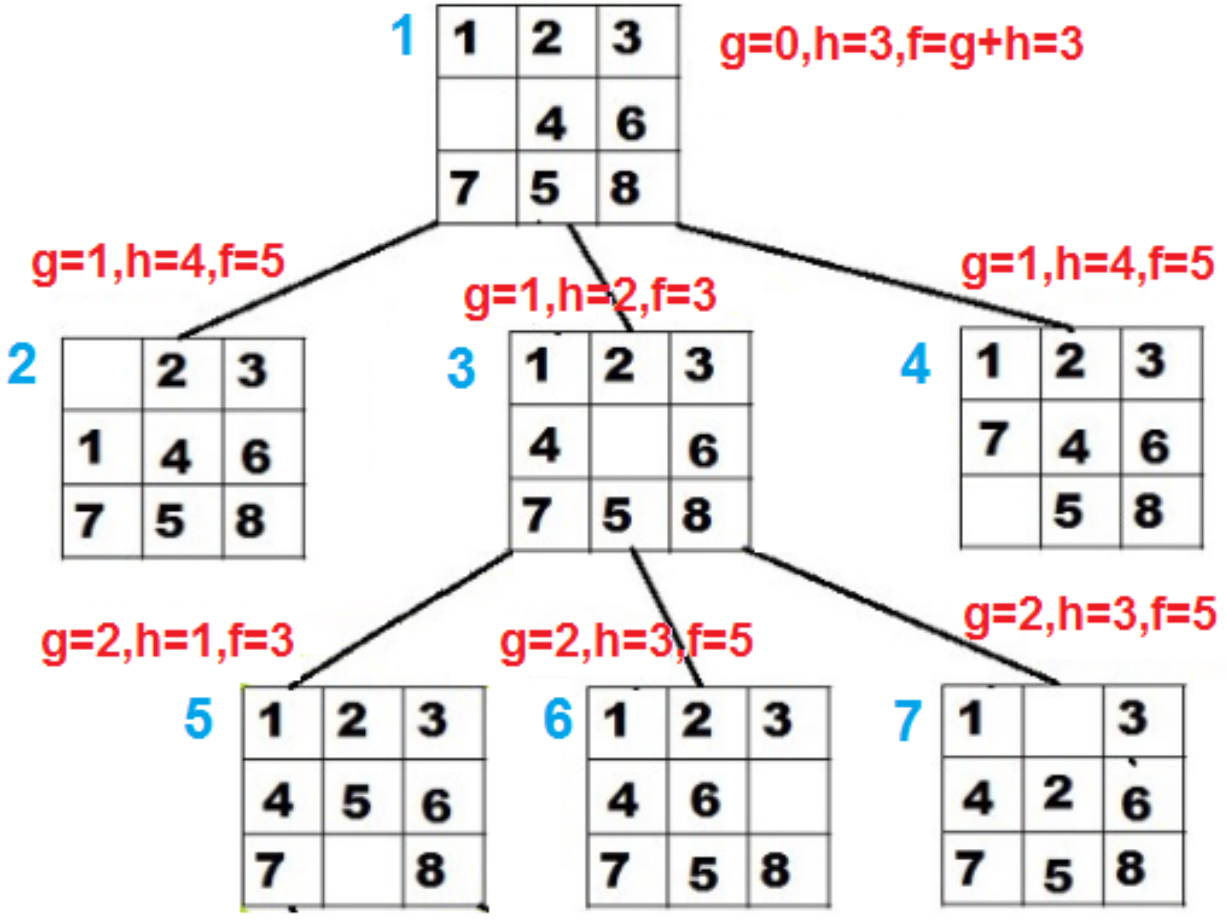
1	2	3
	4	6
7	5	8

g=0,h=3,f=g+h=3

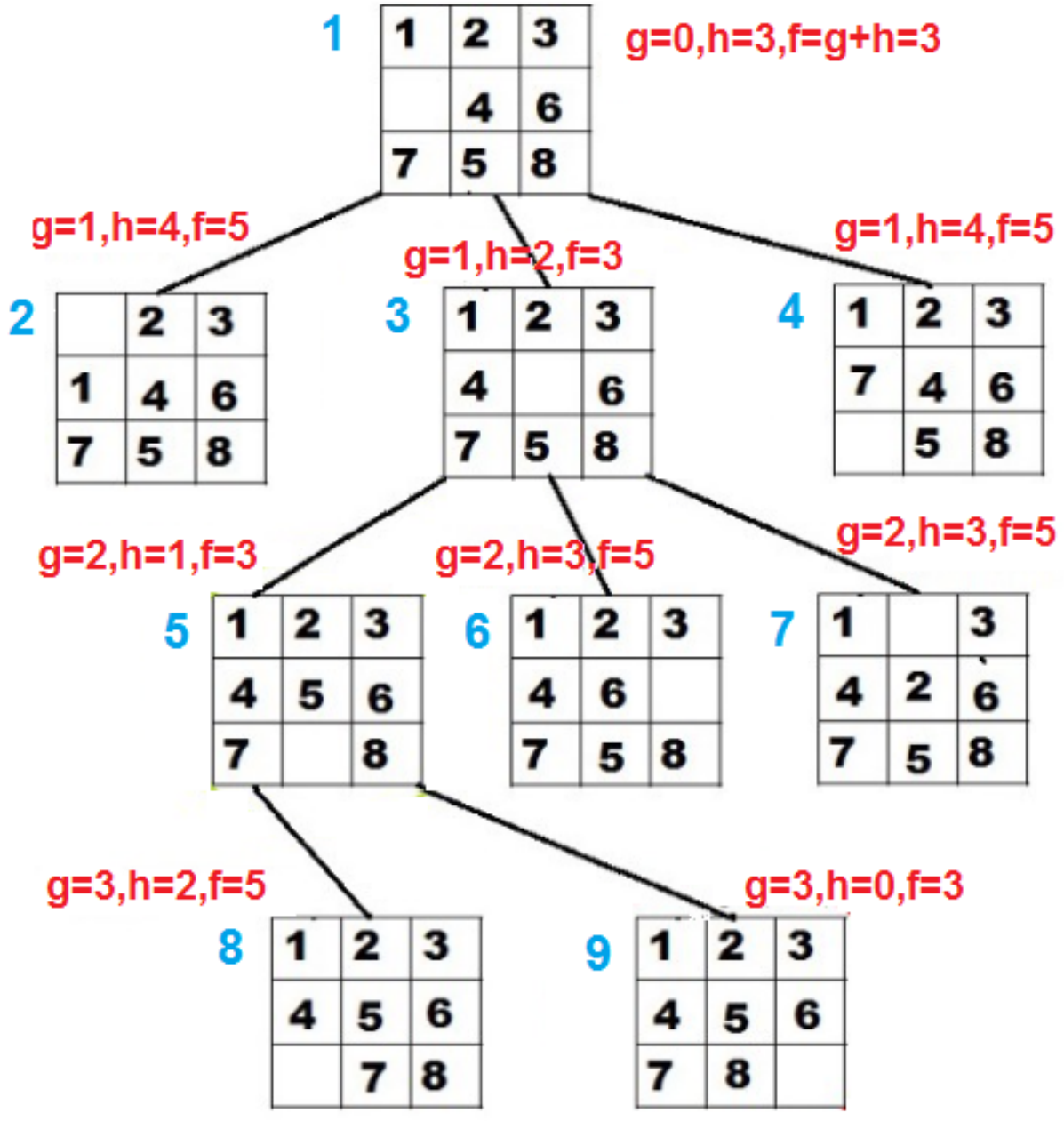
Ejemplo – A*


1
1 2 3
4 6
7 5 8
g=0,h=3,f=g+h=3
g=1,h=4,f=5
2
2 3
1 4 6
7 5 8
g=1,h=2,f=3
3
1 2 3
4 6
7 5 8
g=1,h=4,f=5
4
1 2 3
7 4 6
5 8

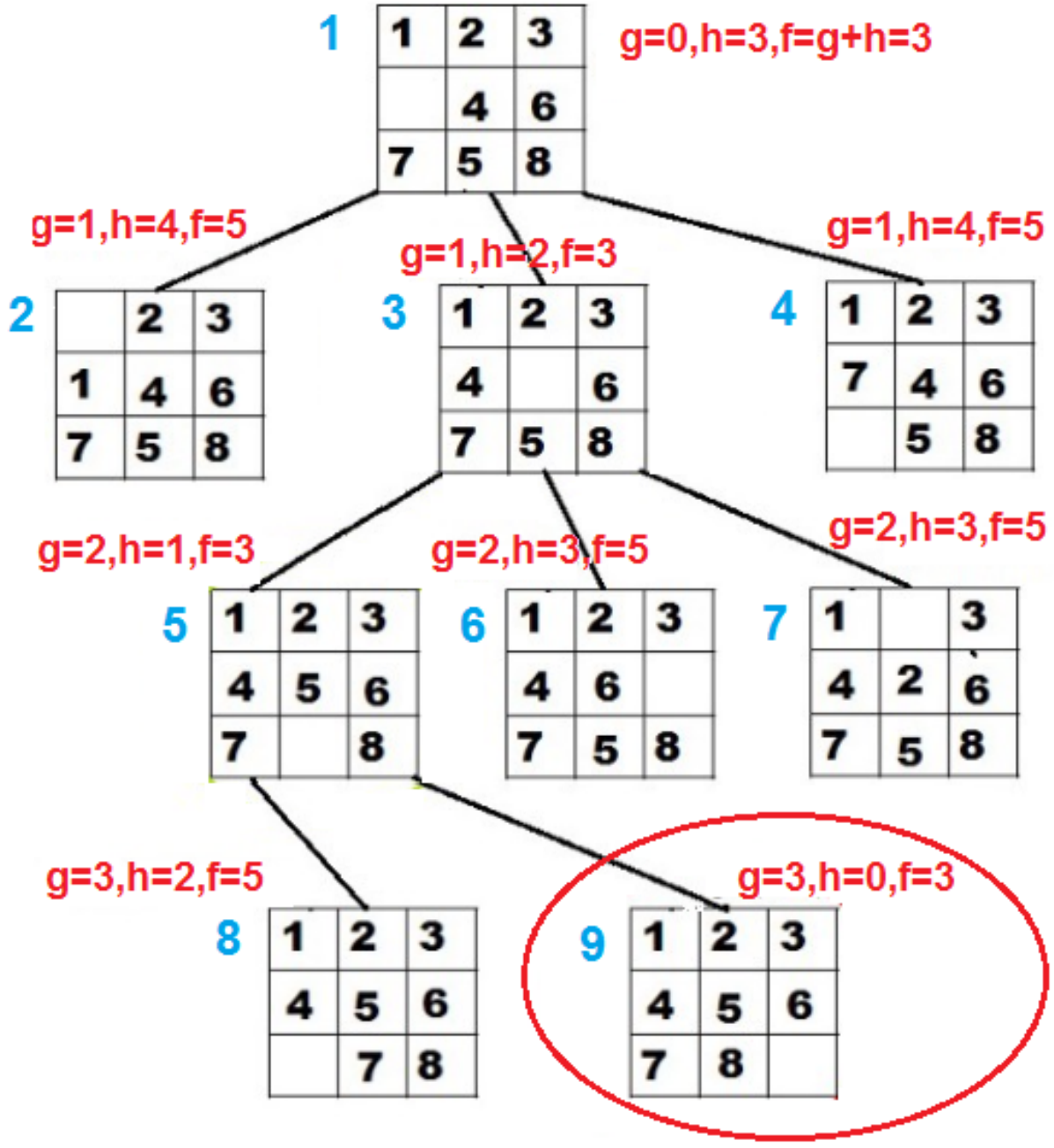
Ejemplo – A*


1
1 2 3
4 6
7 5 8
g=0,h=3,f=g+h=3
g=1,h=4,f=5
g=1,h=2,f=3
g=1,h=4,f=5
2
3
4
g=2,h=1,f=3
g=2,h=3,f=5
g=2,h=3,f=5
5
6
7

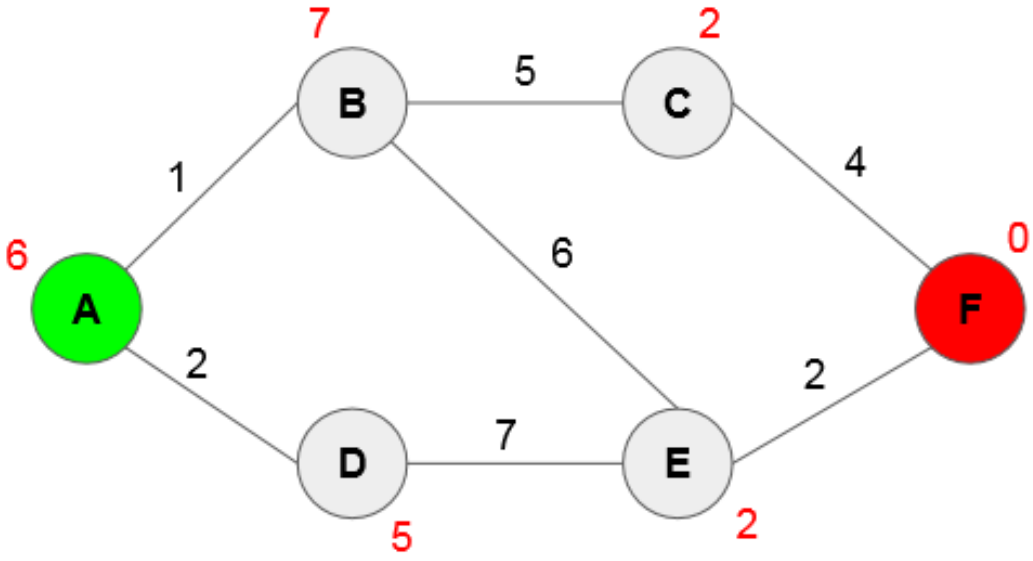
Ejemplo – A*


1
1 2 3
4 6
7 5 8
g=0,h=3,f=g+h=3
g=1,h=4,f=5
g=1,h=2,f=3
g=1,h=4,f=5
2
2 3
1 4 6
7 5 8
3
1 2 3
4 6
7 5 8
4
1 2 3
7 4 6
5 8
g=2,h=1,f=3
g=2,h=3,f=5
g=2,h=3,f=5
5
1 2 3
4 5 6
7 8
6
1 2 3
4 6
7 5 8
7
1 3
4 2 6
7 5 8
g=3,h=2,f=5
g=3,h=0,f=3
8
1 2 3
4 5 6
7 8
9
1 2 3
4 5 6
7 8

Ejemplo – A*


1
1 2 3
4 6
7 5 8
g=0,h=3,f=g+h=3
g=1,h=4,f=5
2
2 3
1 4 6
7 5 8
g=1,h=2,f=3
3
1 2 3
4 6
7 5 8
g=1,h=4,f=5
4
1 2 3
7 4 6
5 8
g=2,h=1,f=3
5
1 2 3
4 5 6
7 8
g=2,h=3,f=5
6
1 2 3
4 6
7 5 8
g=2,h=3,f=5
7
1 3
4 2 6
7 5 8
g=3,h=2,f=5
8
1 2 3
4 5 6
7 8
g=3,h=0,f=3
9
1 2 3
4 5 6
7 8

Ejemplo – A*

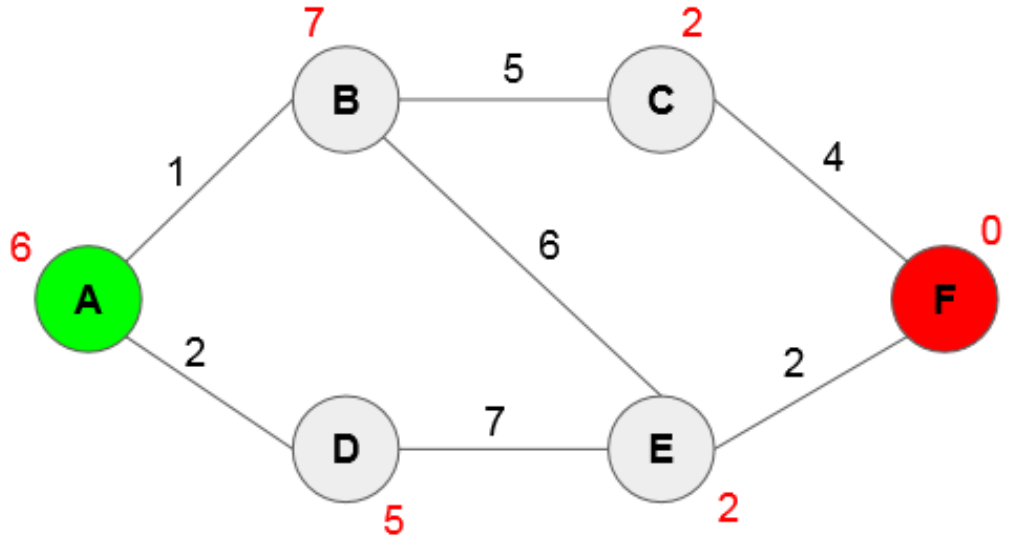


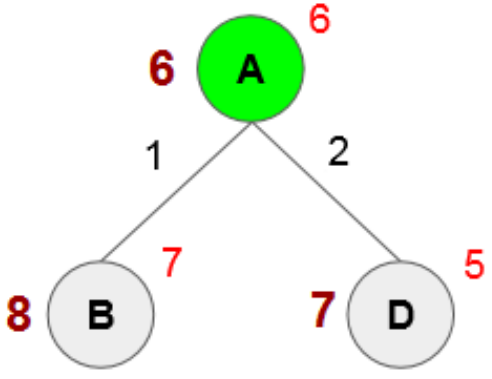
ABIERTA

A 6

CERRADA

Ejemplo – A*





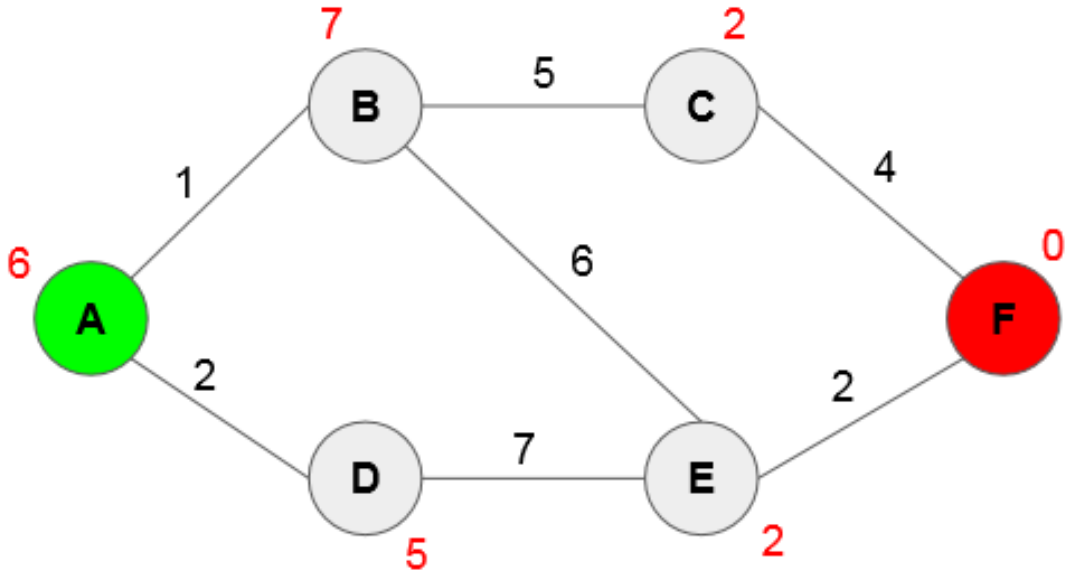
ABIERTA

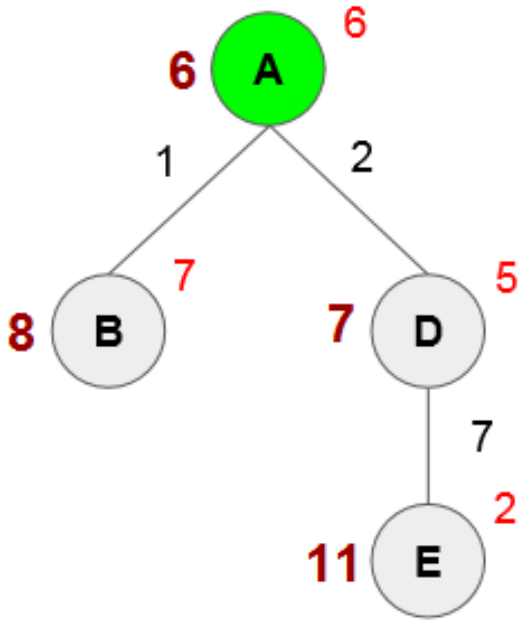
D 7	B 8

CERRADA

A 6

Ejemplo – A*





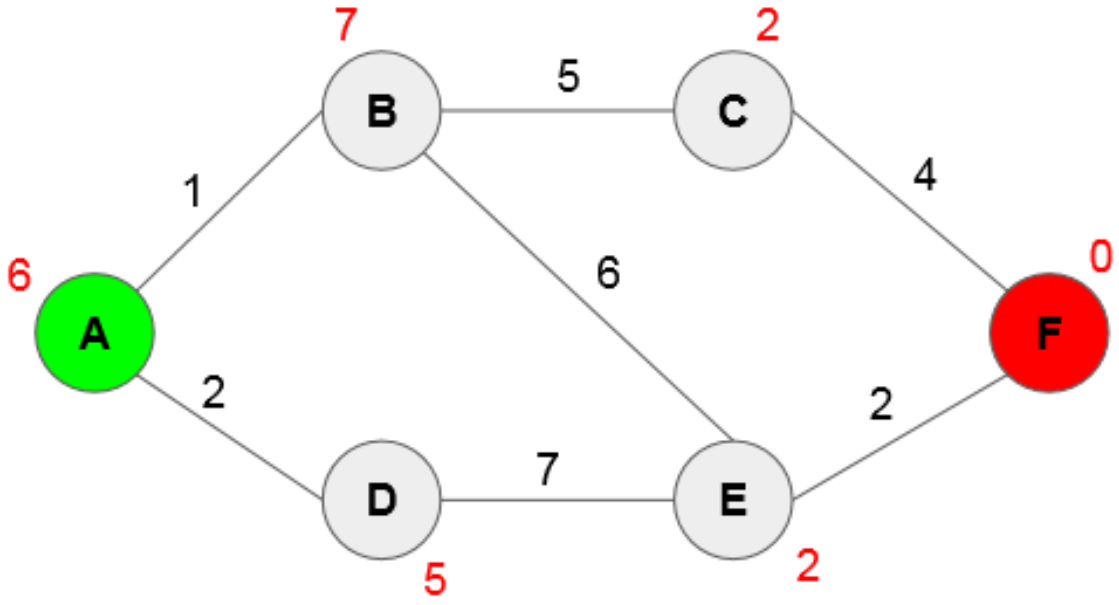
ABIERTA

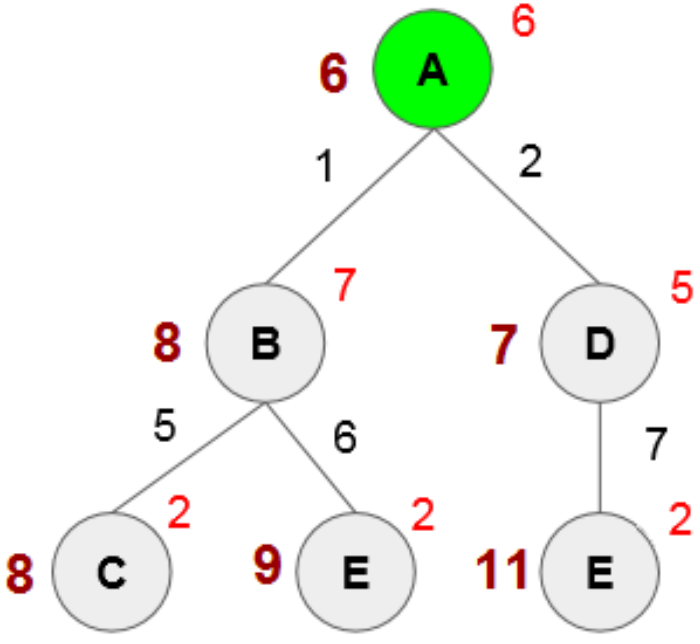
B 8	E 11

CERRADA

A 6	D 7

Ejemplo – A*





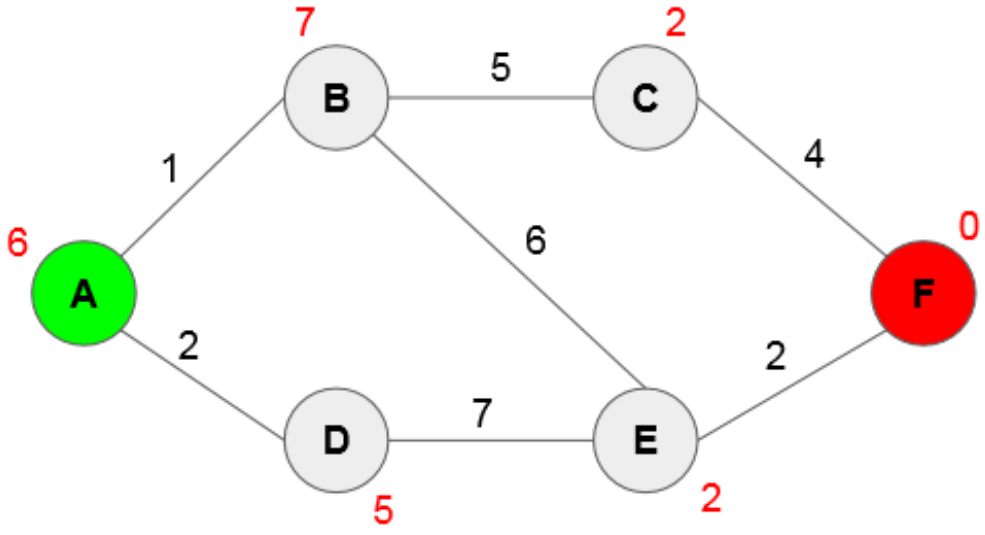
ABIERTA

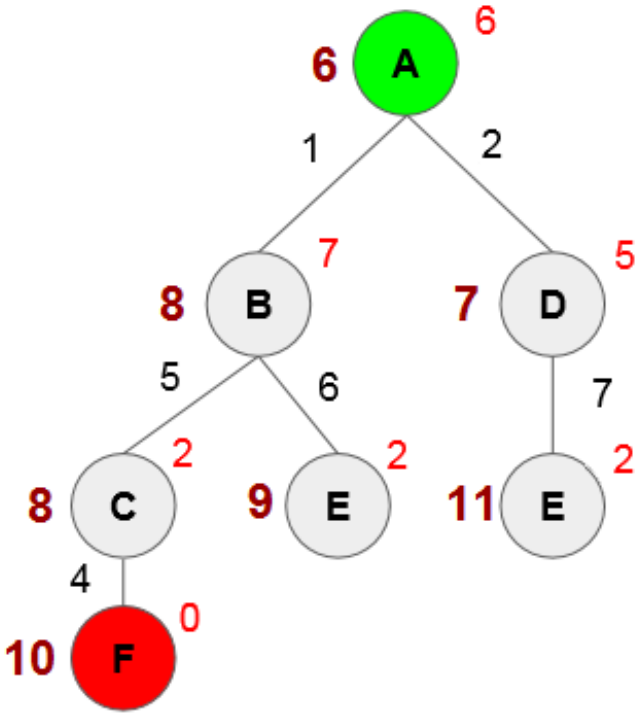
C 8	E 9	E 11

CERRADA

A 6	D 7	B 8

Ejemplo – A*





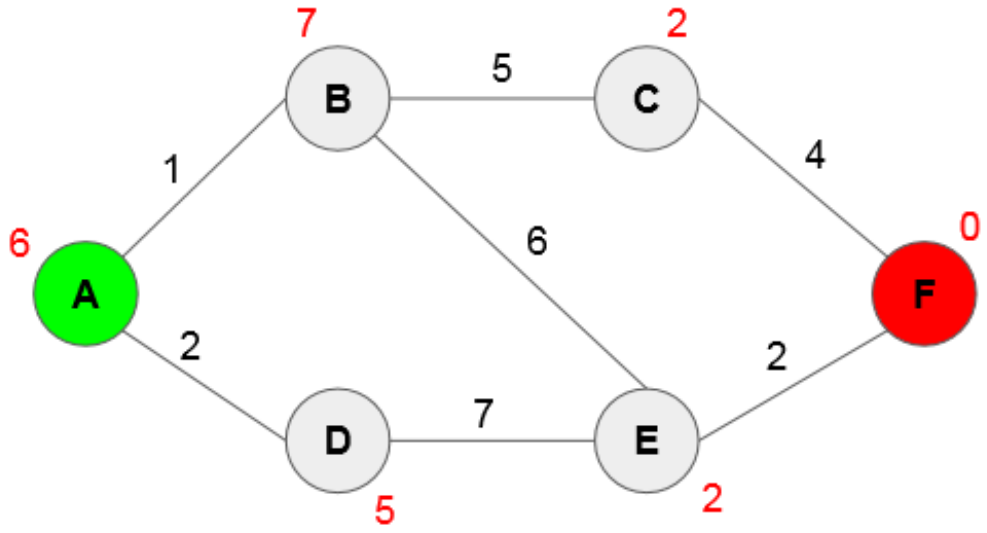
ABIERTA

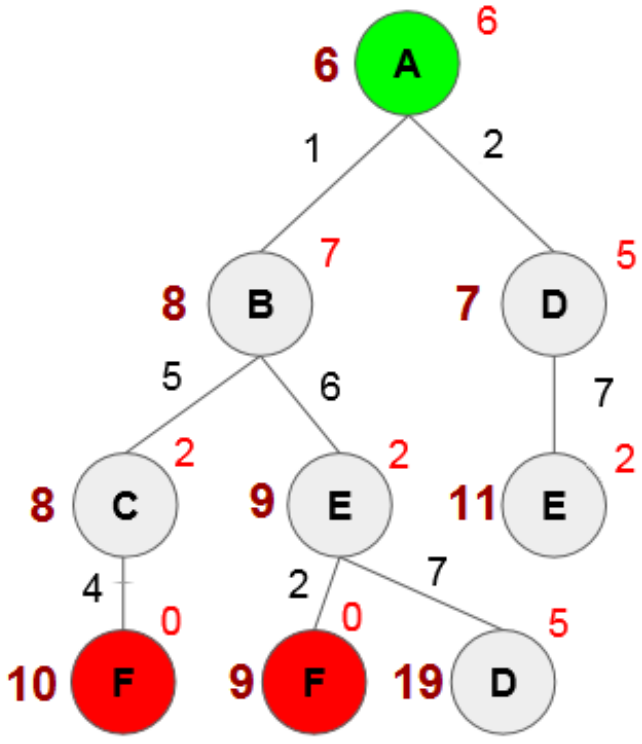
E 9	F 10	E 11

CERRADA

A 6	D 7	B 8	C 8

Ejemplo – A*





ABIERTA

	F 9	F 10	E 11	D 19

CERRADA

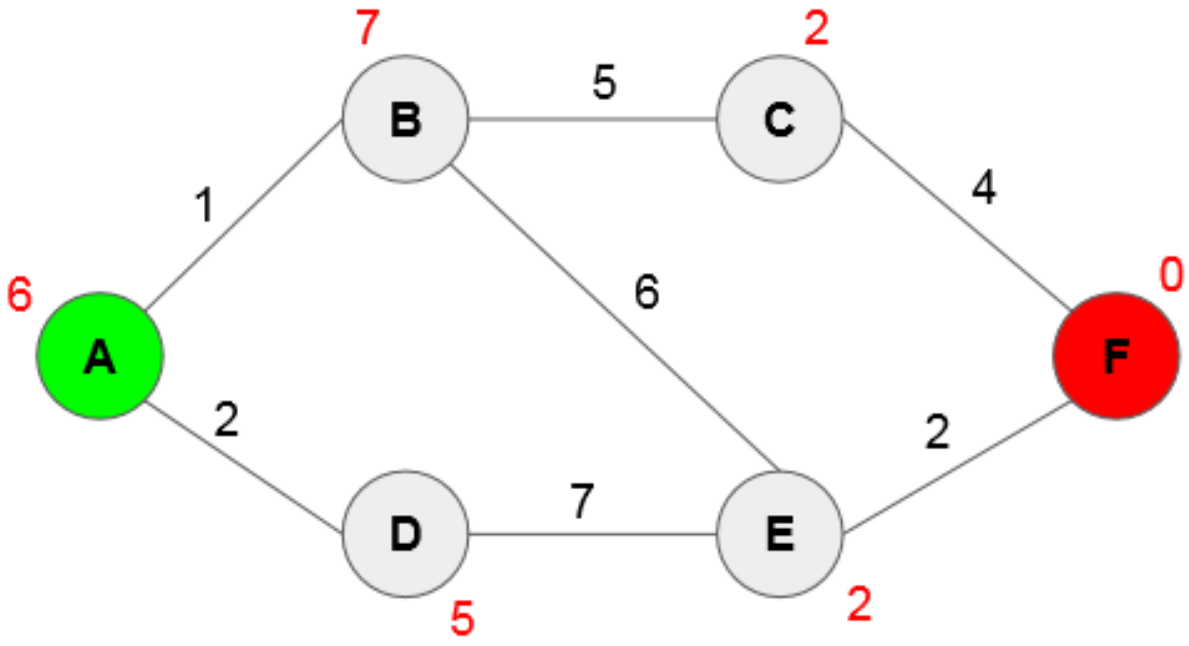
A 6	D 7	B 8	C 8	E 9

Características

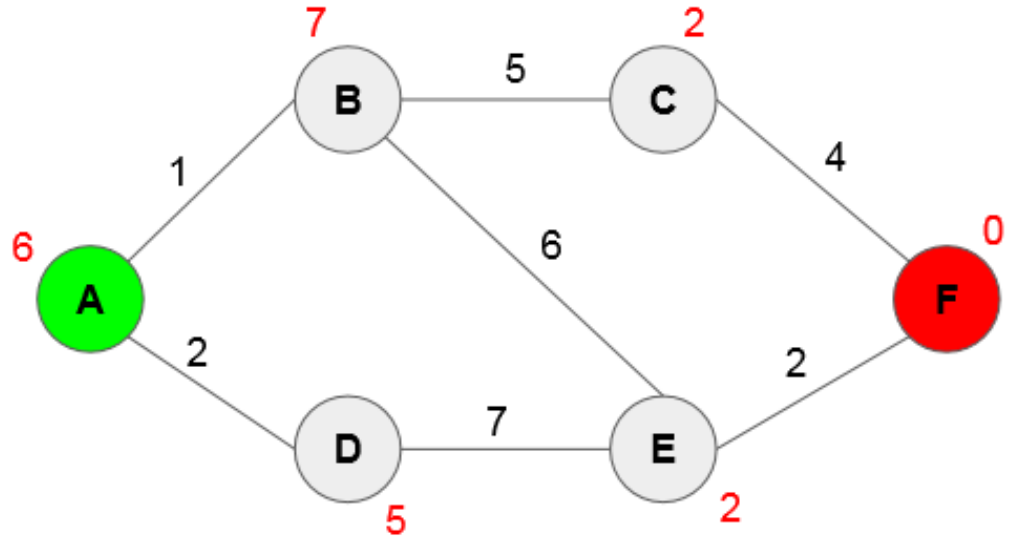
- Completitud: si existe solución, la encuentra
- Admisibilidad: si hay una solución, encuentra la óptima si:
  - el número de sucesores es finito para cada nodo,
  - $k(n_i, n_j) \geq \epsilon > 0$ en cada arco, y
  - La función heurística $h(\cdot)$ es admisible, $h(n) \leq h^*(n) \quad \forall n$
- 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**

IDA* (Korf, 1985)

"Consiste en una serie de recorridos del primero en profundidad hasta que $f(n) > \eta$ o hemos encontrado la solución, incrementando η en cada iteración al menor exceso cometido"

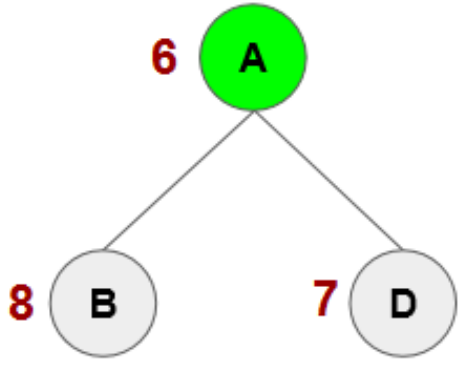


Ejemplo – IDA*

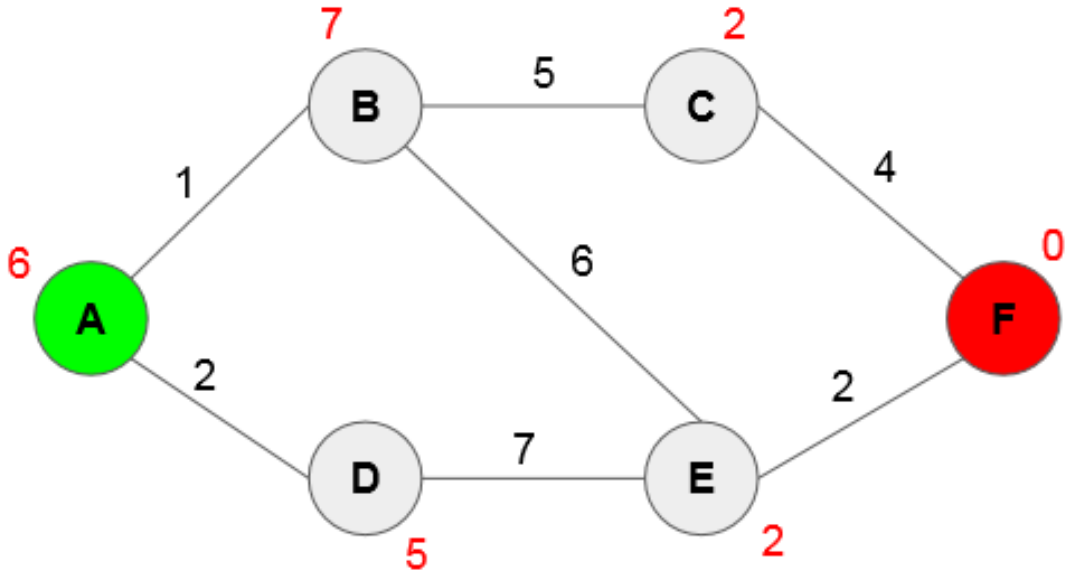


It #0:

η =h(A) = 6

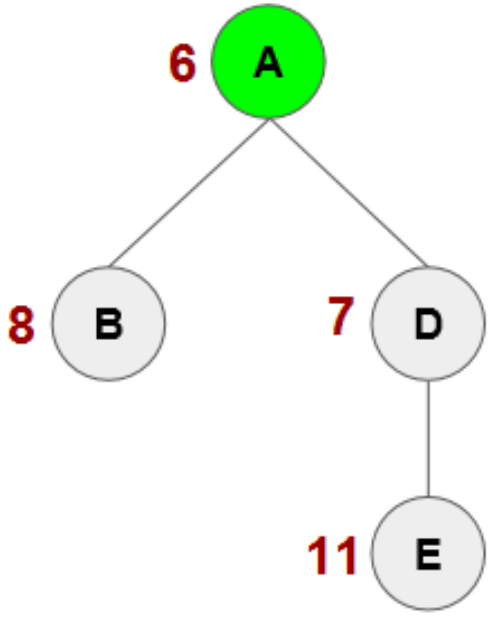


Ejemplo – IDA*

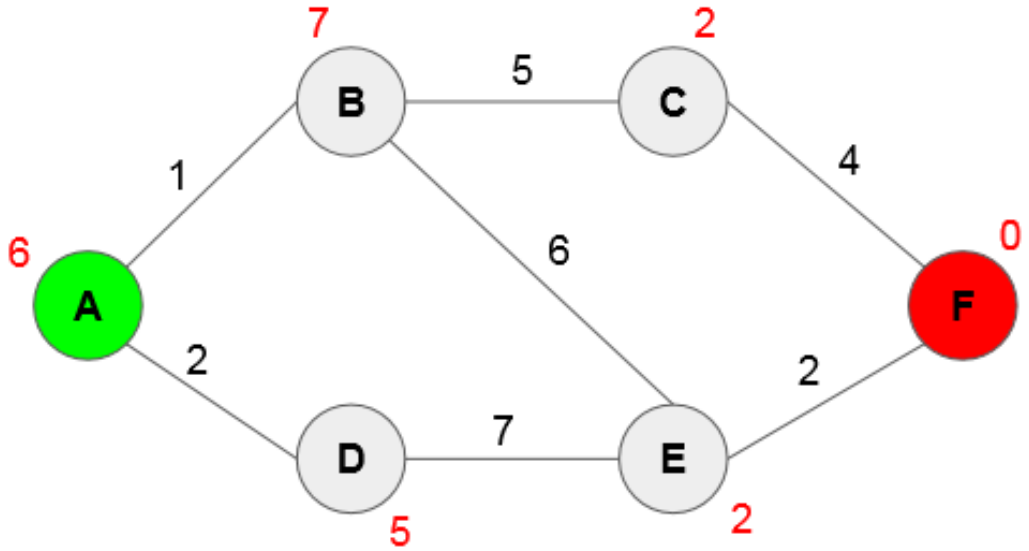


It #1:

$\eta = 7$

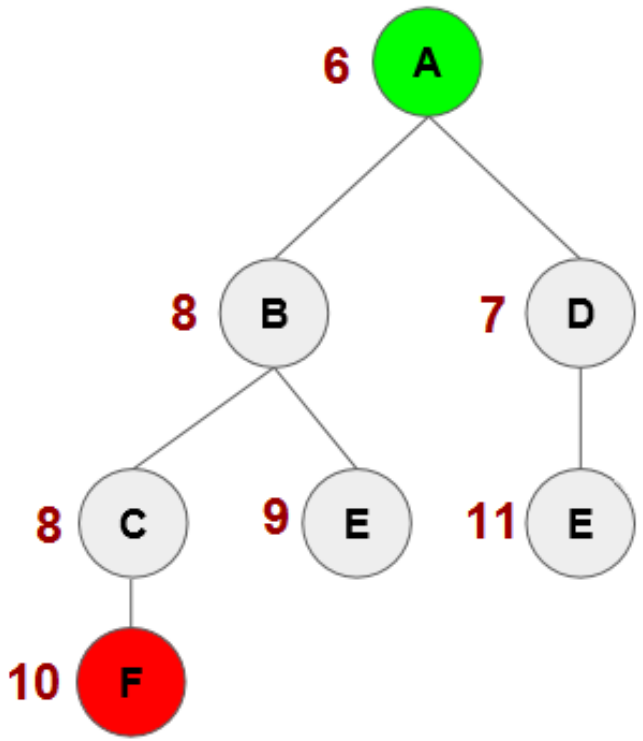


Ejemplo – IDA*

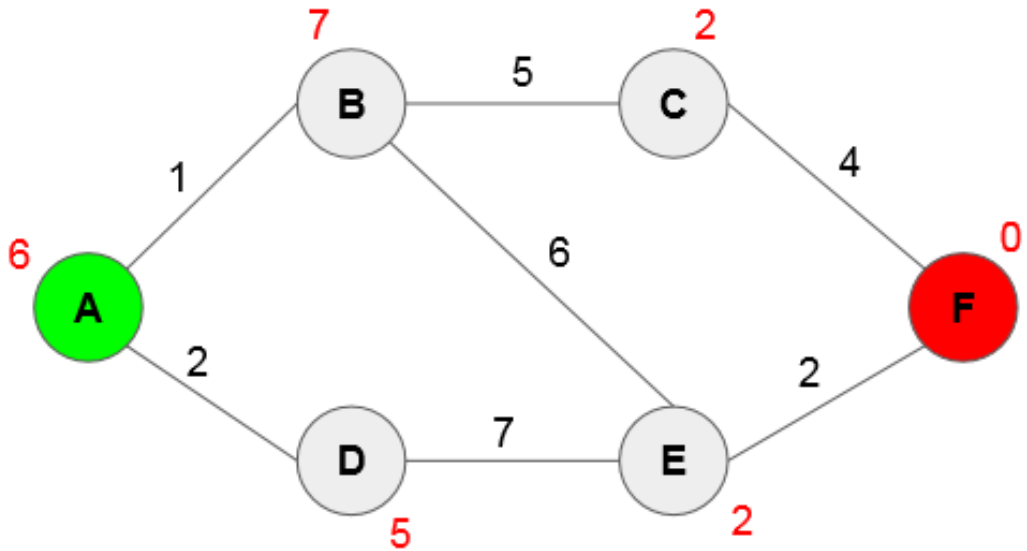


It #2:

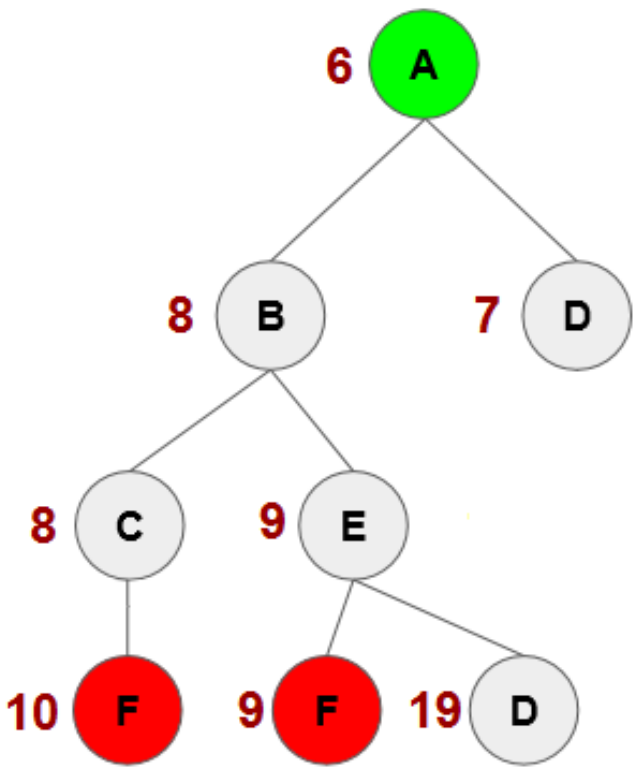
η =8



Ejemplo – IDA*



It #3:
$\eta = 9$



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